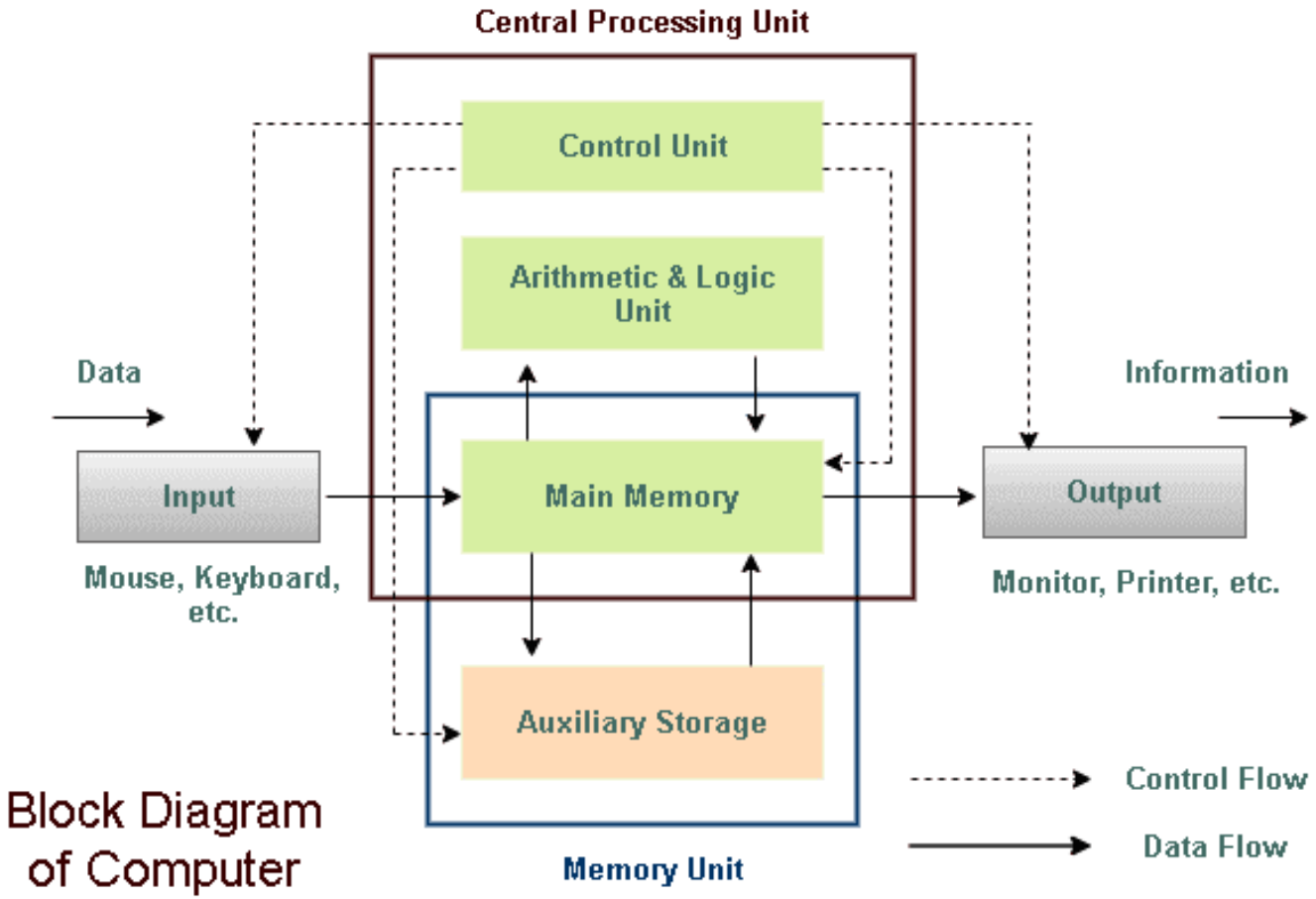
Fundamental of Microprocessor

UNIT – 1

Block Diagram of Computer



8085 Microprocessor Architecture


intel
Core™ i7

Definition

- Microprocessor is
  - Multipurpose
  - Programmable
  - Clock driven
  - Register based
  - Electronic device
  - Reads binary instructions from storage device called memory
  - Accepts data as binary input
  - Processes data according to instructions
  - Provides results as output

- Microprocessor is multipurpose, programmable, Clock driven, Register based Electronic device that reads binary instructions from storage device called memory and accepts data as binary input, processes data according to instructions provides results as output.

- Microprocessor operates in binary 0 or 1
- Each processor recognizes and processes a group of bits called word
- Microprocessor are classified according to their **word length** such as 8 bits, 16 bits etc

Applications

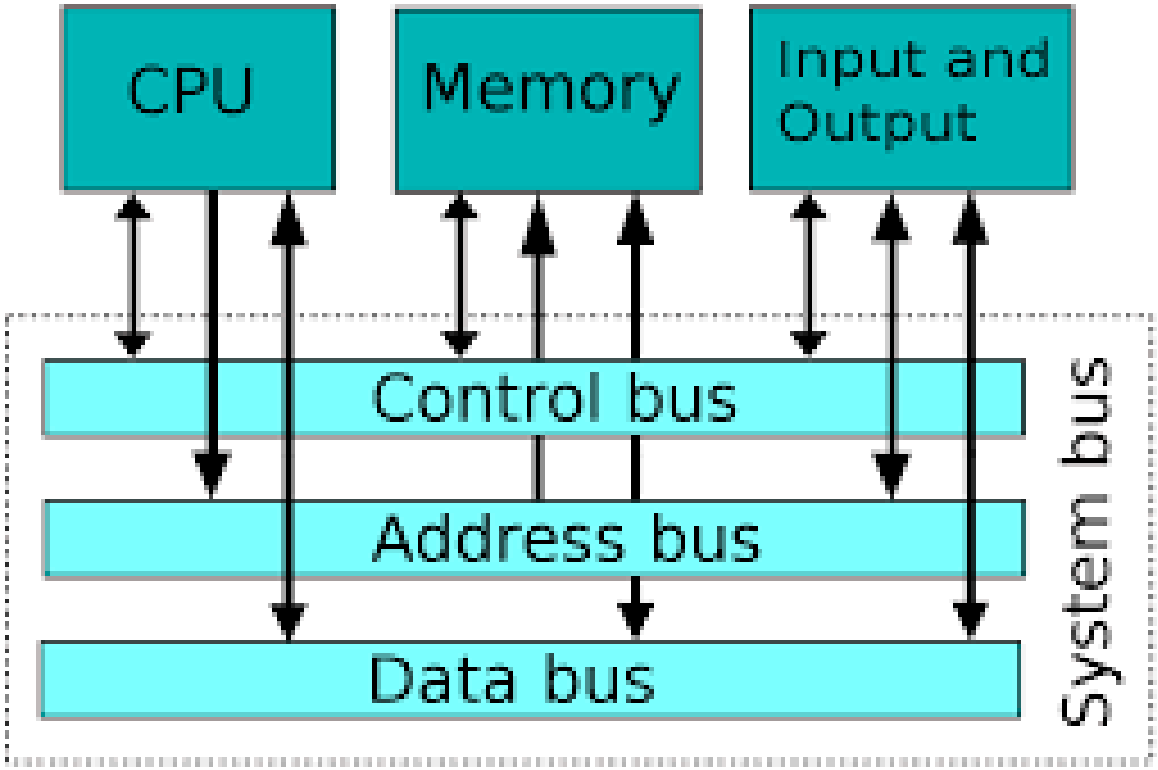
- Microcontrollers
- Measurement and testing equipment
  - Blood group analyzers
  - X-ray analyzer
  - Signal generators
- Washing machine
- Microwave oven
- Scientific and engineering research
- Industry
- Security system: smart cameras, CCTV, smart door
- Traffic light control

Advantage of microprocessor

- Computational speed is high
- Intelligence has been brought to system
- Automation of industrial process and office automation
- Flexible
- Compact in size
- Maintenance is easier

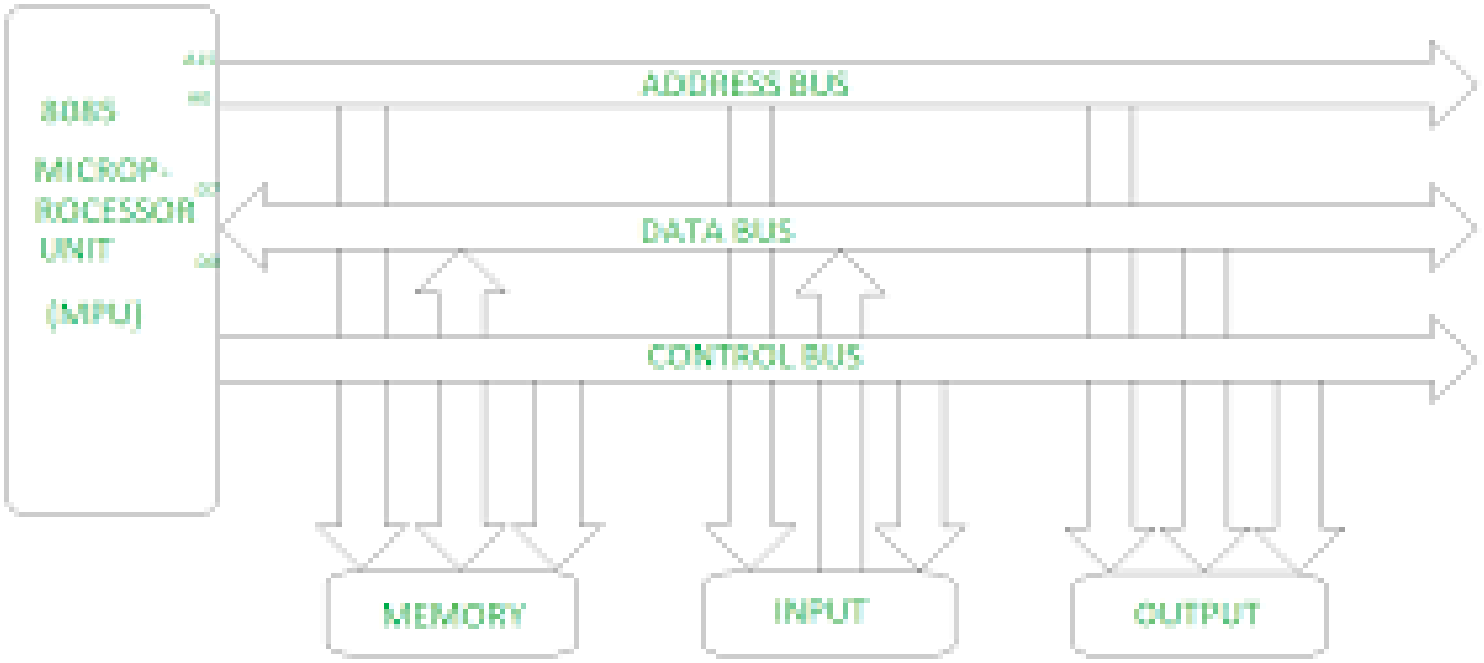
System Bus

- A bus is a subsystem that is used to **connect computer component** and transfer data between them.
- A **system bus** is single computer bus that connects the major components of a computer system combining the functions of a **Data bus** to carry information, an **Address bus** to determine where it should be sent and **Control bus** to determine its operation.


CPU
Memory
Input and Output
Control bus
Address bus
Data bus
System bus

- Design of the system bus **varies from system to system** and can be specific to particular computer design.
- System bus **characteristics** are dependent on need of the processor, the speed, and the wordlength of data and instructions.
- **Size of bus** :- known as width, determines how much data can be transferred at a time and indicates the number of available wires.
- 32 bit bus refers 32 parallel wires that can simultaneously transfer 32 bits.

Microprocessor with bus organization



Bus organization system of 8085 Microprocessor

Microprocessor with Bus organization

- Bus is a group of conducting wires which carries information.
- All peripherals are connected to microprocessor through bus.
- Three types of buses
- Address bus
- Control Bus
- Data Bus

- Address Bus
  - It is group of conducting wires which carries address only.
  - Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to input /output devices.
  - Length of address bus of 8085 is 16 bit or 4 hexa digits, from 0000 H to FFFF H.
  - It can address 65536 different memory locations

- Data Bus
  - Group of conducting wires which carries data only.
  - Data bus is bidirectional because data flow both way from microprocessor to memory or I/O devices and from memory or Input/output devices to microprocessor.
  - 8085 have 8 bit data bus means 2 hexadecimal ranging from 00 H to FF H.
  - ***Write operation***: processor will put data to be written in the data bus.
  - ***Read operation***: memory controller will put data into data bus from specific memory location.

- Width of data bus is directly related to largest number that the bus can carry, such as 8 bit can carry from 0 to 255.

- Control bus
  - It is group of conducting wires which is used to generate timing and control signals to all associated peripherals.
  - Some control signals are:
    - Memory Read
    - Memory write
    - I/O read
    - I/O write
    - Opcode fetch

Microprocessor Architecture and operations

- Microprocessor is programmable digital device, designed with registers, flip-flops and timing elements.
- The microprocessor has set of instructions designed internally to manipulate data and communicate with peripherals.
- The process of data manipulation and communication is determined by the logic design of the microprocessor, called architecture.

- Microprocessor can be programmed to perform functions on given data by selecting necessary instructions from its set.
- This instructions are given to the microprocessor by writing them into memory.
- Writing or entering instructions and data are given by input device.

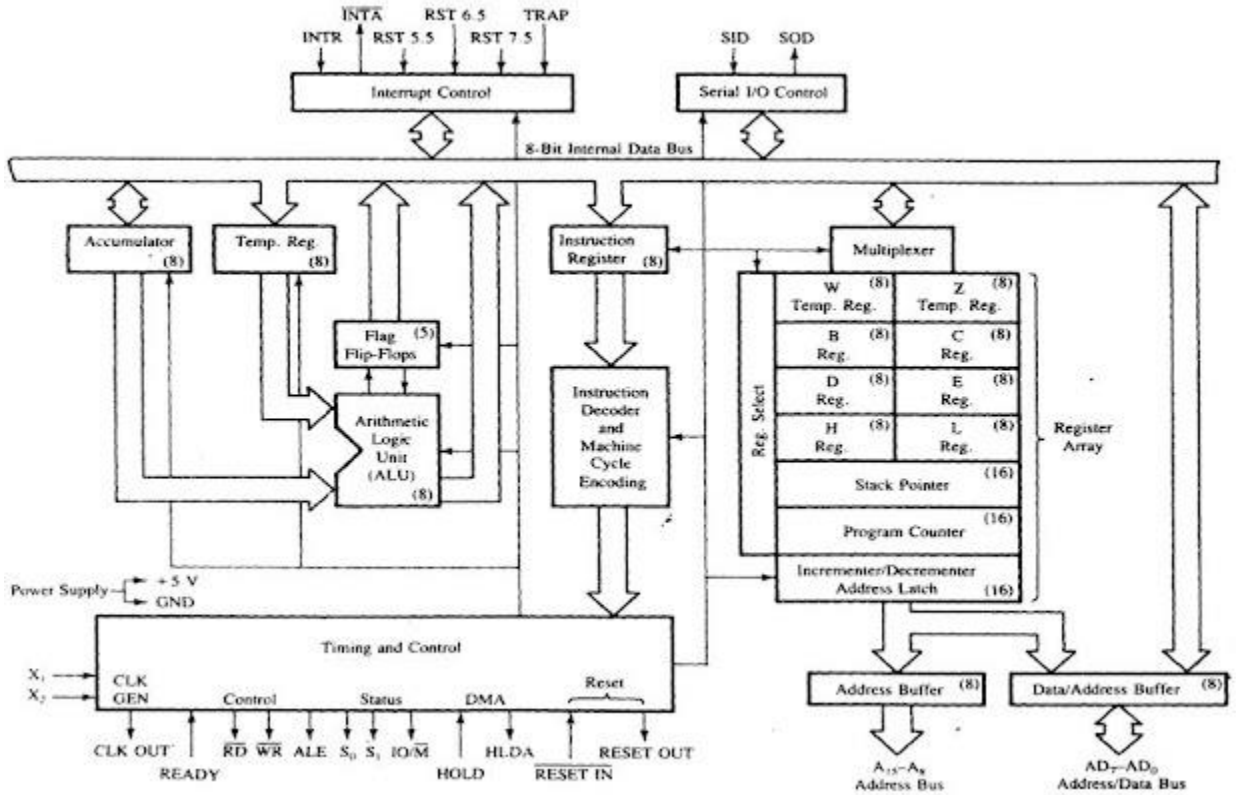
- The microprocessor reads or transfer one instructions at time, matches it with its instructions set and performs data manipulation indicated by instructions.
- The result can be store into the memory or send to output device.

- Microprocessor can respond to external signals.
- It can be interrupted ,reset, or asked to wait to synchronize with slower peripherals.

8085

- 8085 is complete 8 bit parallel CPU.
- Main components are:-
  - Array of registers
  - ALU
  - Encoder/Decoder
  - Timing and control circuits
- All components are linked by internal data bus.

8085 microprocessor functional block diagram



- ALU
  - Performs computing functions
  - Includes accumulator, temporary registers, arithmetic and logic circuits and five flags.
  - Temporary registers hold data during Arithmetic and logic operations.
  - Result is stored in accumulator.
  - Flag (flip-flops) are set or reset according to the result of operations.

- Accumulator(register A):
  - 8 bit register which is part of ALU
  - stores 8 bit data
  - 8085 is accumulator based microprocessor
  - When data is read from input port it is moved to accumulator and when data is send to output port, it must be first place in accumulator.

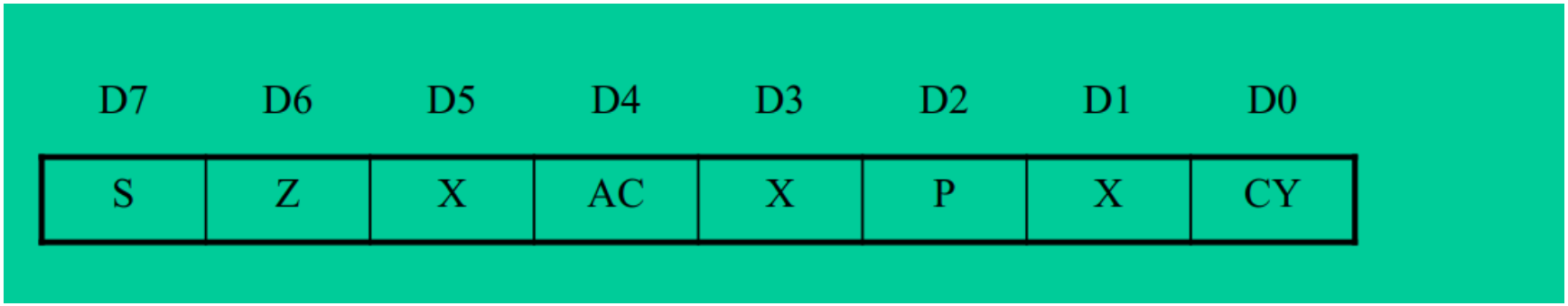
- Temporary Register(W and Z)
  - 8 bit registers not accessible to the programmer
  - During program execution 8085 places the data for short period
- Instruction Register(IR)
  - 8 bit register not accessible to the programmer
  - It receive the operation code from internal data bus and passes to the instruction decoder which decodes so that microprocessor knows which type of operation to perform.

- Registers Array
  - General Purpose Registers
    - B, C, D, E, H & L (8 bit registers)
    - Can be used singly
    - Or can be used as 16 bit register pairs
      - BC, DE, HL
    - H & L can be used as a data pointer (holds memory address)
  - Data can be directly added or transferred from one to another.
  - Contain can be incremented or decremented or combined logically with the contain of the accumulator
  -

- Stack Pointer
  - It is 16 bit register known as memory pointer
  - It points to the memory location in RAM called stack
  - The beginning of the stack is defined by loading a 16 bit address in the stack pointer.
- Program Counter
  - This is a register that is used to control the sequencing of the execution of instructions.
  - This register always holds the address of the next instruction.
  - PC is incremented by 1to point next memory location.

- Flag register
  - Registers consists of five flip-flops, each holding the status of different states separately.
  - Known as flag registers or flags
  - 8085 set or reset one or more flags.
  - S (sign flag), Z (zero flag), AC (auxillary carry flag), P (parity flag) & CY (carry flag)

- **Carry(CY)**- If the last operation generates a carry, its status will be 1 otherwise 0
  - It can handle the carry or borrow from one word to another
- **Zero(Z)**-If the result of last operation is zero, Its status will be 1 otherwise 0
  - It is often used in loop control and in searching for particular data value
- **Sign(S)**-If the MSB of the result of the last operation is 1(negative) then its status will be 1 otherwise 0
- **Parity (P)**-If the result of last operation has even number of 1's its status will be 1 otherwise 0
- **Auxillary Carry(AC)**- If the last operation generates carry from the lower half word its status will be 1otherwise 0. Used for performing BCD arithmetic.


D7 D6 D5 D4 D3 D2 D1 D0
S Z X AC X P X CY

- Timing and control unit
  - This unit synchronizes all the microprocessor operations with the clock and generates necessary control signals for communications between microprocessor and peripherals.
- Interrupt controls
  - The various interrupt control signals are used to interrupt microprocessor.
  - INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP
- Serial I/O control
  - Two serial I/O control SID and SOD are used for serial data transmission.

Microprocessor operations

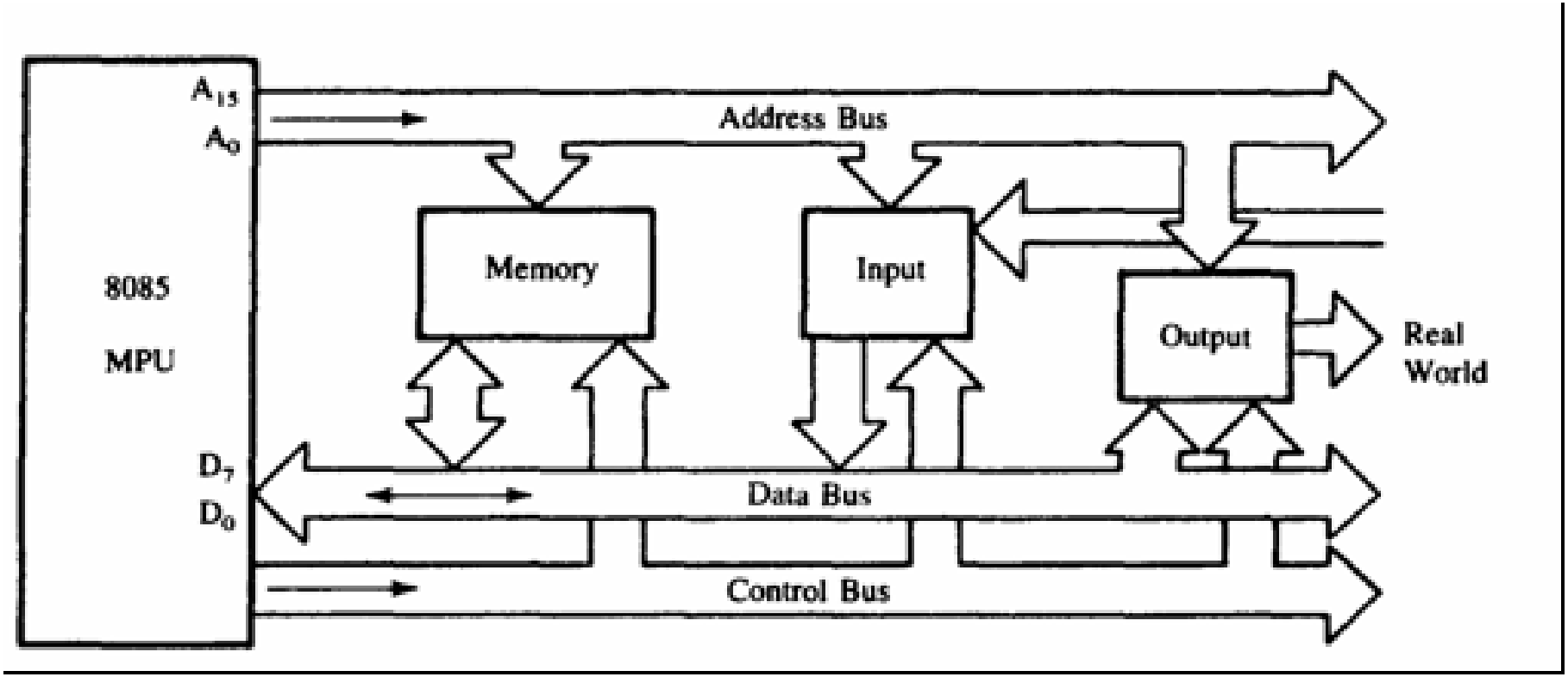
- Functions performed by microprocessor can be classified in three general category.
  - Microprocessor initiated operations
  - Internal operations
  - Peripheral(or externally initiated) operations

Microprocessor initiated operations

- Microprocessor performs primarily four operations:
  - **Memory Read**:-Read data or instructions from memory
  - **Memory Write**:-Write data or instructions from memory
  - **I/O read** :-Accepts data from input devices
  - **I/O write**:-sends data to output devices
- To communicate with a peripherals or a memory microprocessor needs to perform following steps:-
  - Step1 identify the peripherals or memory location with its address
  - Step2 transfer binary information(data or instructions)
  - Step 3 provide timing or synchronization signals

- 8085 perform this using set of communications lines called buses.
- Address bus, data bus and control bus.

8085 Bus structure



- **Address Bus**
  - 8085 Microprocessor has 16 bit address bus.
  - The bus over which the CPU sends out the address of the memory location is known as Address bus.
  - The address bus carries the address of memory location or peripherals devices to be written or to be read from.
  - The address bus is ***unidirectional.*** It means bits flowing occurs only in one direction, only from microprocessor to peripheral devices.

NOTE:-We can find that how much memory location is needed ,by using the formula 2^N. where N is the number of bits used for address lines.

In 8085 the memory size is $2^{16} = 65536$bytes or 64Kb, So it can access upto 64 kb memory location.

- **Q.**>If a processor has 4 GB memory then how many address lines are required to access this memory?
- **Ans**: 4GB= 4 * 1GB

$4 = 2^2$

$1GB = 2^{30}$

So, $4GB = 2^2 * 2^{30} = 2^{32}$

So 32 address lines are required to access the 4 GB memory.

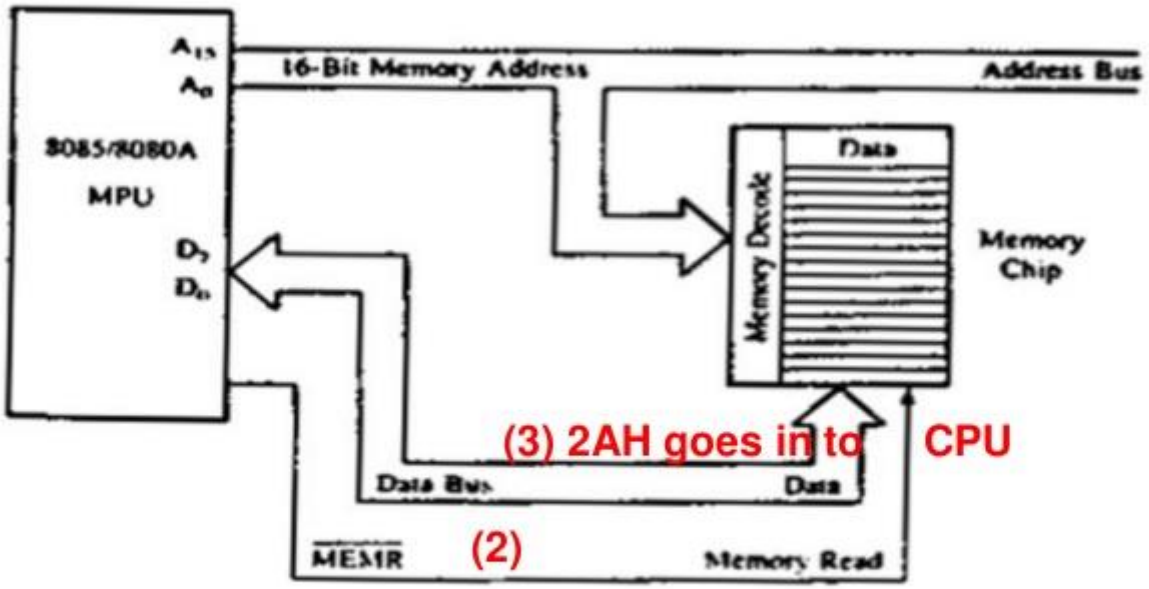
- **Data bus**
  - 8085 Microprocessor has 8 bit data bus.
  - So it can be used to carry the 8 bit data starting from 00000000H(00H) to 11111111H(FFH). Here 'H' tells the Hexadecimal Number.
  - It is bidirectional. These lines are used for data flowing in both direction means data can be transferred or can be received through these lines.
  - The data bus also connects the I/O ports and processor.
  - The largest number that can appear on the data bus is 11111111.
  - It has 8 parallel lines of data bus. So it can access upto 2^8 = 256 data

- **Control Bus**
  - Control bus is comprised of various single lines that carry synchronization signals.
  - These are not group of lines as in data bus and address bus, it is a individual line that provides a pulse to indicate microprocessor operations.
  - The microprocessor generates specific control signals for every operations.

Memory read operation

Read Operation:

(1): 0010 0000 0101 0000 = 2050H



Memory Read Operation

Internal Data operations

- Internal architecture of microprocessor define what and how the operations can be performed with the data. These operations are:
  - Store 8 bit data
  - Perform arithmetic and logical operations
  - Test for conditions
  - Sequence the execution of instructions
  - Store data temporarily during execution in the defined R/W memory locations called stack

- To perform all operations the microprocessor requires registers , ALU and control unit, and internal buses.

Peripherals or Externally initiated operations

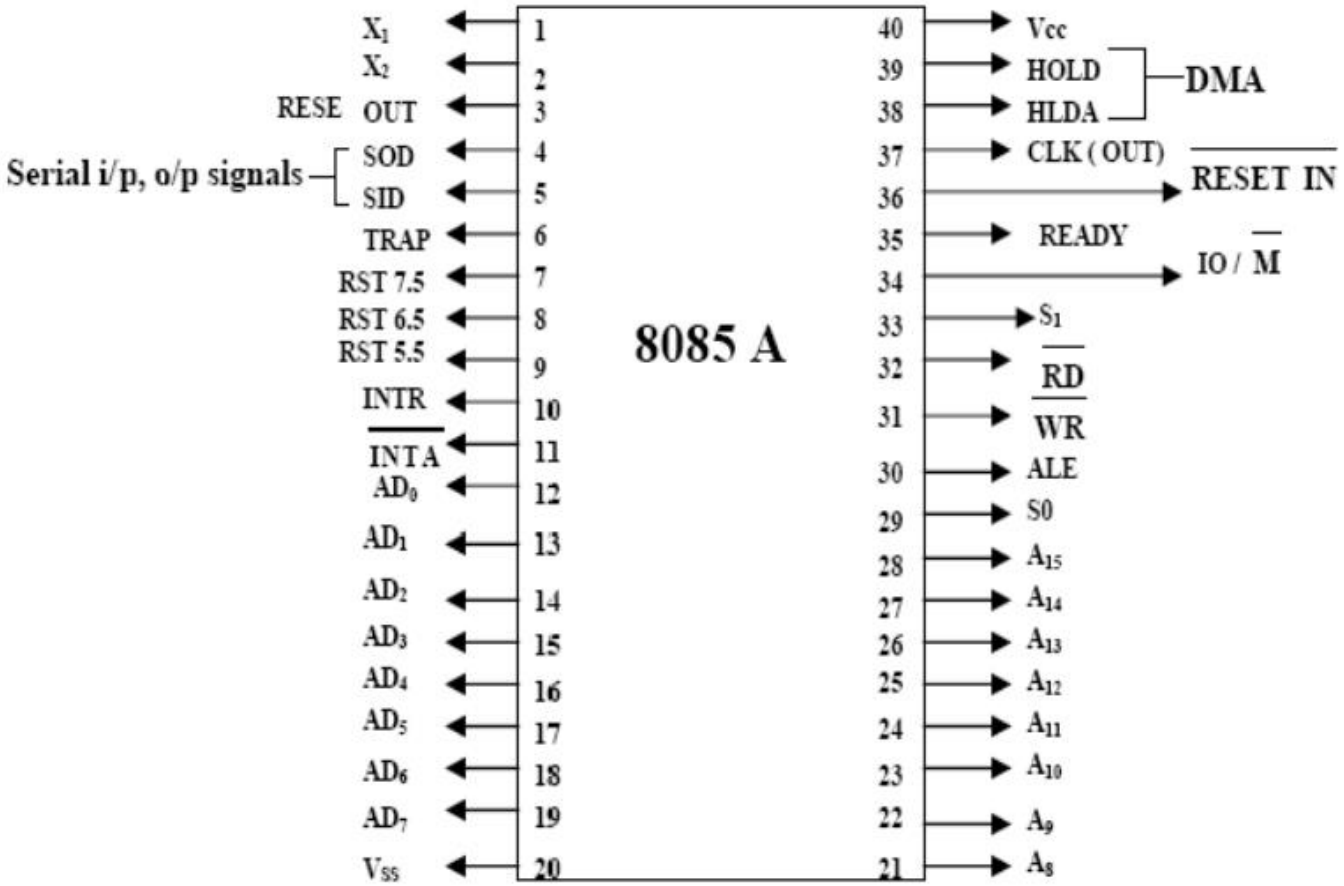
- External devices can initiate the following operations, for which individual pins on the microprocessor chip are assigned: Reset, Interrupt, Ready, Hold.
- **Reset**:
  - When reset pin is activated by an external key, all internal operations are suspended and the program counter is cleared to 0000H. Now the program execution can again begin at the zero memory address.
- **Interrupt:**
  - The microprocessor can be interrupted from the normal execution of instructions and asked to execute some other instructions called service routine. The microprocessor resumes its operation after completing the service routine.

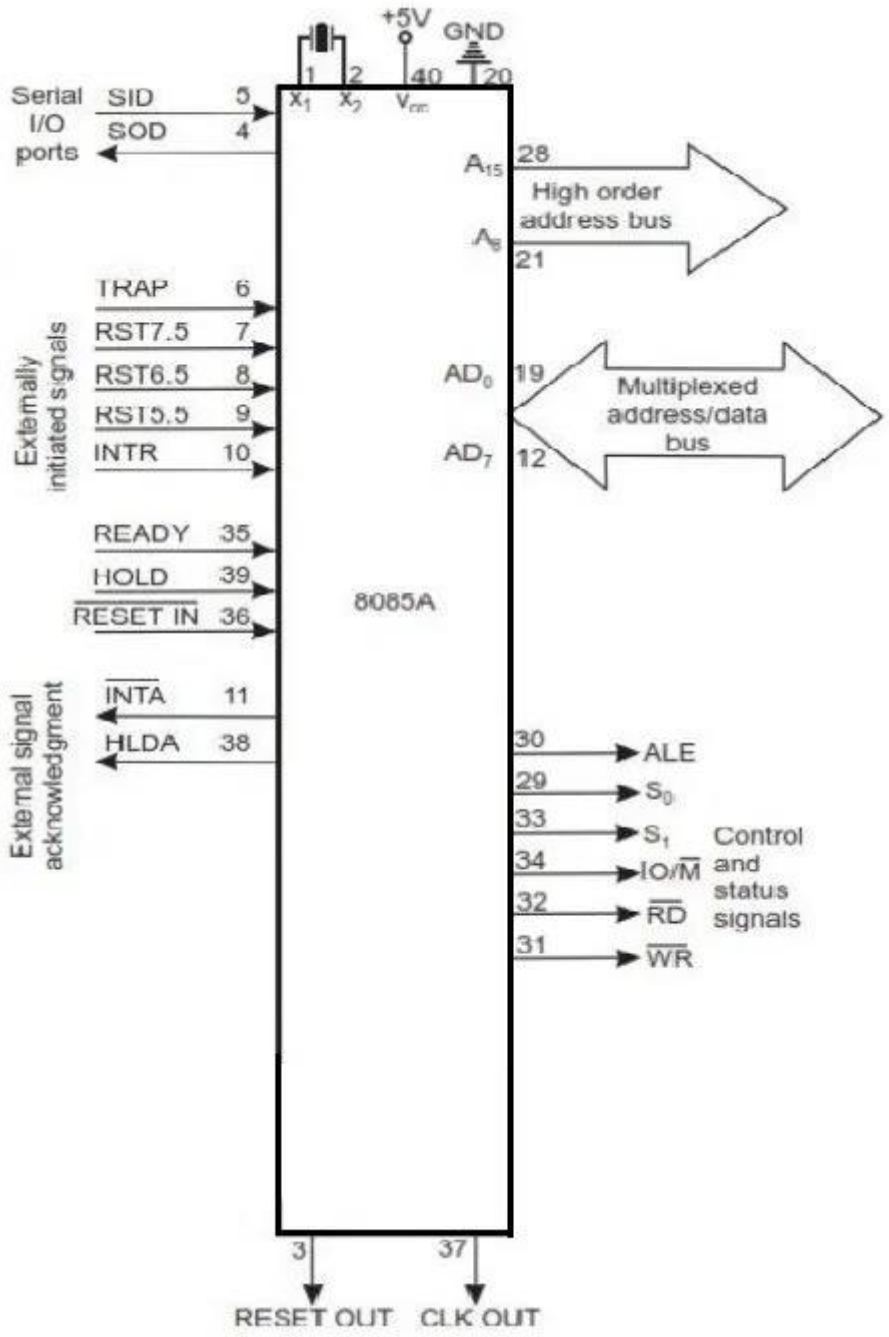
- Ready:
  - 8085 has pin called ready. When this ready pin is low, the microprocessor enters into Wait state. This signal is used primarily to synchronize slower peripherals with microprocessor.
- Hold:
  - When the HOLD pin is activated by an external signals, the microprocessor relinquishes control of buses and allows the external peripheral to use them.

8085 Pin configuration

- 8085 is 8 bit microprocessor.
- It has 16 bit address bus with 64 KB addressing capabilities.
- 8085 is 40 pin DIP package.
- The pins can be grouped as
  - 1. power supply and clock signals
  - 2. Address bus
  - 3. Data Bus
  - 4. Control and status bus
  - 5. Interrupt and externally initiated signals
  - 6. Serial I/O port

8085 Pin configuration




+5V
GND
1
2
40
20
X1
X2
Vcc
Serial I/O ports
SID
SOD
5
4
A15
A8
28
21
High order address bus
TRAP
RST7.5
RST6.5
RST5.5
INTR
6
7
8
9
10
Externally initiated signals
AD0
AD7
19
12
Multiplexed address/data bus
READY
HOLD
RESET IN
35
39
36
8085A
INTA
HLDA
11
38
External signal acknowledgment
30
29
33
34
32
31
ALE
S0
S1
IO/M
RD
WR
Control and status signals
3
37
RESET OUT
CLK OUT

1. Power supply and clock frequency

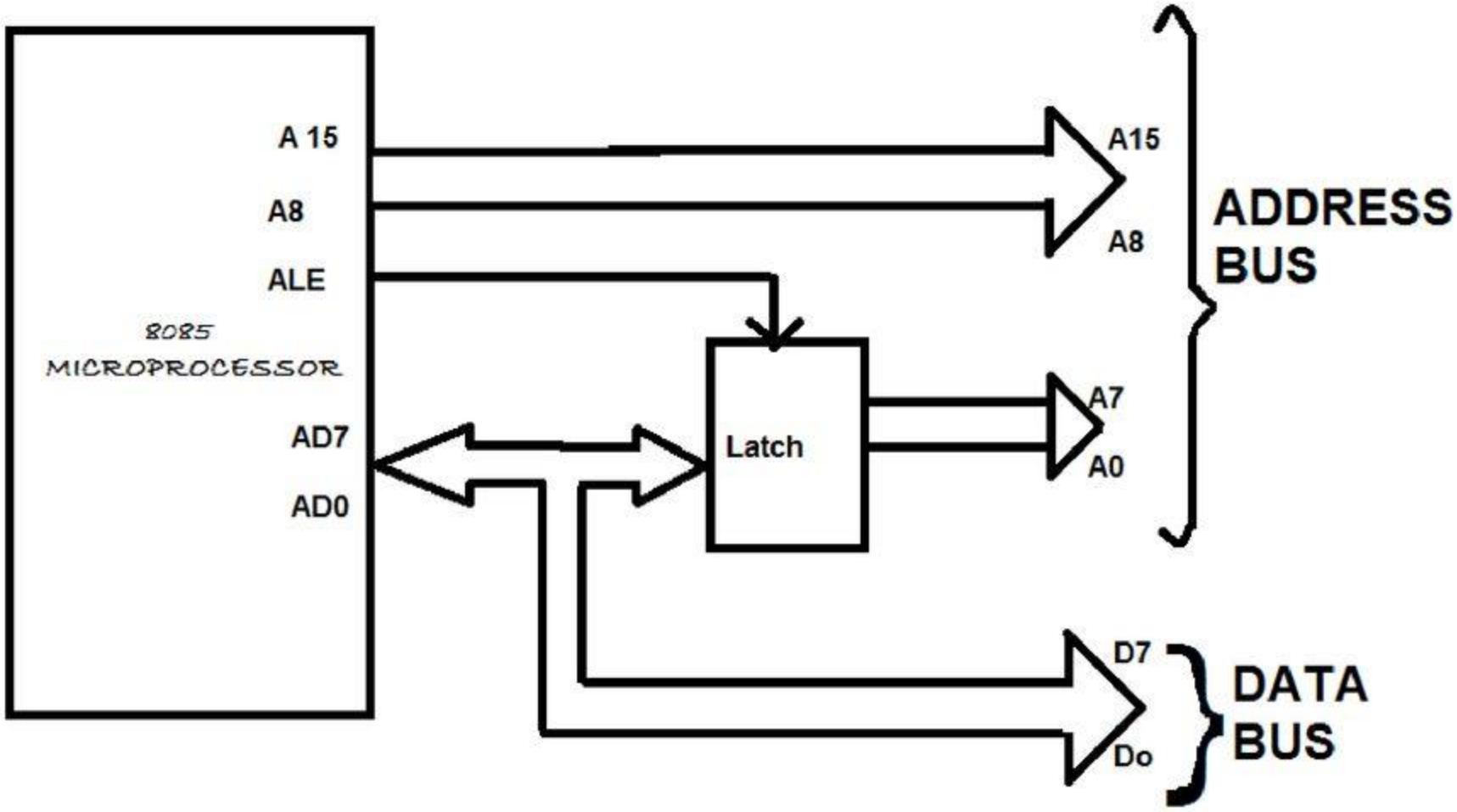
- V_{cc} = +5 volt power supply
- V_{ss} = Ground
- X1,X2: crystal or R/C network or LC network connections to set the frequency of internal clock generator.
- Frequency 3 MHz
- CLK (output)- clock output is used as system clock for peripherals and devices interfaced with the microprocessor.

2. Address Bus

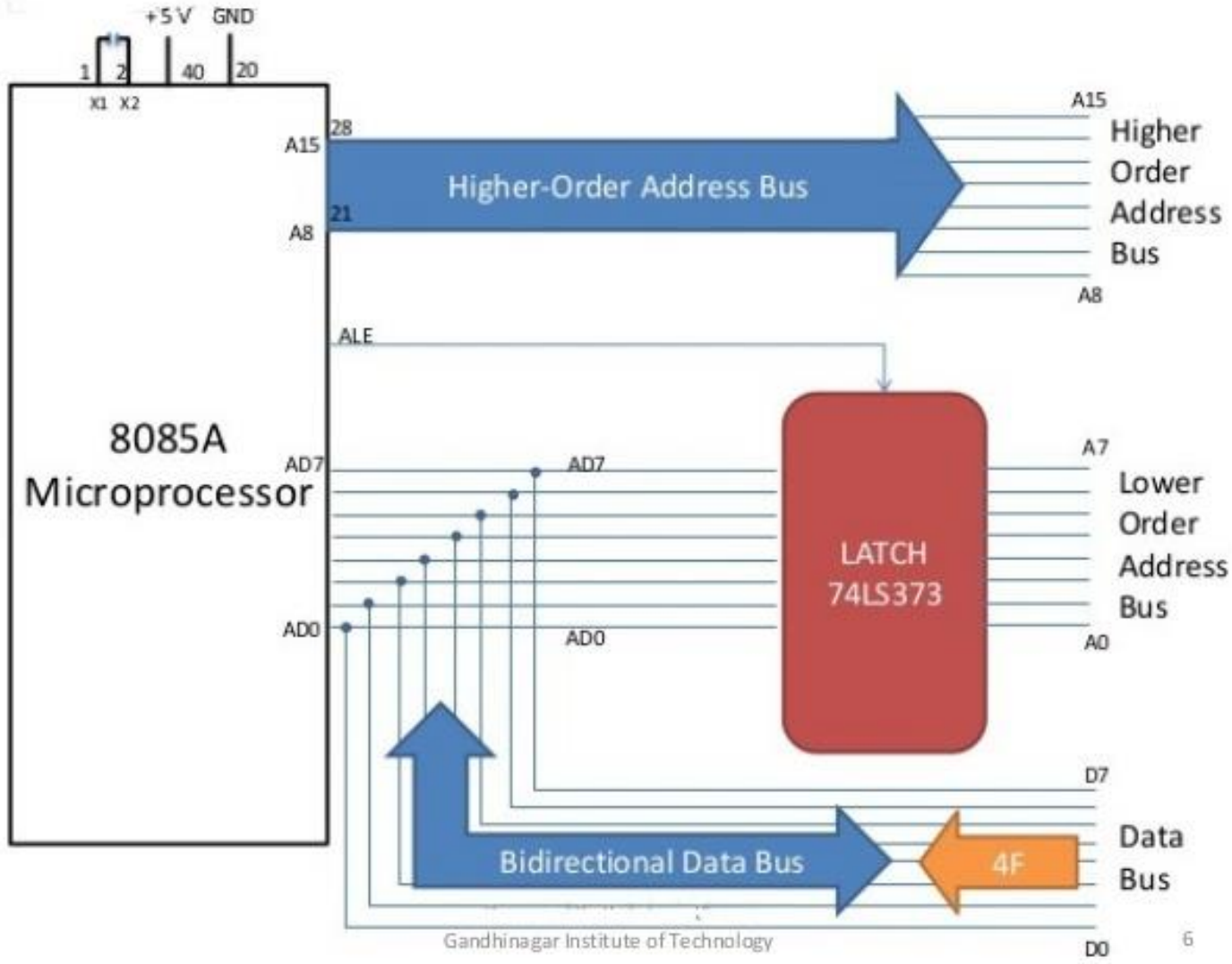
- A8–A15
- It carries most significant bits of memory address or the 8 bits of the I/O address.

3. Multiplexed Address /Data Bus

- AD0-AD7
- These multiplexed set of lines used to carry the lower order 8 bit address as well as data.
- During the opcode fetch operation, in the first clock cycle, the lines deliver the lower order address A0-A7.
- In subsequent IO/Memory, Read/Write clock cycle the lines are used as data bus.
- The CPU may read or write out data through this lines.



[FIG : DE MULTIPLEXING OF MULTIPLEX ED ADDRESS DATA BUS.]


+5 V
GND
1
2
40
20
X1
X2
8085A
Microprocessor
A15
28
A8
21
Higher-Order Address Bus
A15
Higher
Order
Address
Bus
A8
ALE
AD7
AD0
AD7
AD0
LATCH
74LS373
A7
Lower
Order
Address
Bus
A0
D7
Bidirectional Data Bus
4F
Data
Bus
D0

4. Control and status signal

- These signals contains two control signal (RD and WR) , three status signal(IO/M,S1 and S0) to identify the nature of operations and one special signal (ALE) to indicate the beginning of operations.
  - **ALE (Output):- Address Latch Enable**
    - The signal helps to capture the lower order address presented on the multiplexed address/ data bus.
  - **RD(Active Low):-Read memory or IO device**
    - It means that the selected memory location or I/O device is to be read and that the data bus is ready to accept data from memory or I/O device.

- **WR (Active Low):- Write memory or I/O device**
  - This indicates that the data on the data bus is to be written into the selected memory location or I/O device.
- **IO/M(output):- Select memory or I/O device**
  - The status signal indicates that the read/ write operation relates to whether the memory or I/O device. It goes high to indicate an I/O operation and low for ,memory operation.

$IO/\bar{M}$	S_1	S_0	States
0	0	1	Memory Write
0	1	0	Memory Read
1	0	1	I/O write
1	1	0	I/O read
0	1	1	Opcode fetch

5. Interrupt and externally initiated signals

- Interrupt are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt TRAP, RST 7.5,RST6.5,RST5.5 and INTR.
  - **INTA**:-It is interrupt acknowledgement signal.
  - **RESET IN**:-When the signal on this pin is low, the PC is set to 0, and microprocessor is reset.
  - **RESET OUT**:- This signal indicates that the processor is being reset. The signal can be used to reset other connected devices.

- **READY**:- This means device is ready to send and receive data. When this signal is low, the processor waits for an integral number of clock cycles until it goes high.
- **HOLD**:-this signal indicates that other master is requesting the use of address and data buses.
- **HLDA(HOLD Acknowledge)**:-It indicates that CPU has received HOLD request and it will relinquish the bus in next clock cycle. HLDA is set to low after HOLD signal is removed.

6. Serial I/O signals

- SOD and SID:- These lines are used for serial communication.
  - **SOD(Serial output data line)**:- The output SOD is set/reset as specified by SIM (Set Interrupt Mask) Instructions.
  - **SID(Serial input data line):-** The data on this line is loaded into accumulator whenever a RIM (Read Interrupt Mask) instruction is executed.

Interrupts in 8085

- Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are five types of interrupt i.e RST5.5,RST6.5,RST7.5, INTR and TRAP. Interrupt are classified into following groups on basis on their parameter:
- **Vector Interrupt:-**
  - In this type of interrupt, the interrupt address is known to the processor. For example RST7.5, RST6.5,RST5.5 ,TRAP.

- **Non vector Interrupt**
  - In this type of interrupt the interrupt address is not known to processor. So, address need to be send externally by the device to perform interrupts.
  - Example: INTR
- **Maskable interrupt**
  - In this type of interrupt, interrupt can be disable by writing some instructions in a program. For example RST5.5,RST6.5,RST7.5
- **Non maskable Interrupt**
  - In this type of interrupt we cant disable the interrupt by writing some instructions in the program. For Example TRAP

- **Software interrupt**
  - In this type of interrupt the programmer had to add the instructions into the program to execute the interrupt . There are 8 software interrupt in 8085, i.e RST 0-7
- **Hardware interrupt**
  - There are 5 hardware interrupt in 8085 used as hardware interrupt i.e Trap,RST5.5,RST6.5,RST7.5,INTA

Interrupt Service Routine(ISR)

- A small program or a routine that when executed, services the corresponding interrupting source is called ISR.
- TRAP
  - Non maskable interrupt
  - Has highest priority among all interrupt
  - By default it is enabled until it get acknowledged
  - This transfers control to 0024H

- RST 7.5
  - Maskable interrupt
  - Have second priority
  - When interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH
- RST 6.5
  - Maskable interrupt
  - Have third priority
  - When interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H

- RST 5.5
  - Maskable interrupt
  - When interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH

- INTR
  - Maskable interrupt
  - Have lowest priority
  - It can be disable by resetting microprocessor
  - When INTR goes high following event occurs:
    - The microprocessor checks the status of INTR signal during the execution of each instruction.
    - When INTR signal is high then microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
    - When instructions are received then microprocessor saves the address of the next instruction on stack and executes the received instruction.

Addressing modes of 8085

- The different ways in which a processor can access data are referred as its addressing modes.
- The addressing mode is indicated in instructions itself.
- The various addressing modes are:
  - Register Addressing Mode
  - Immediate Addressing Mode
  - Direct Addressing mode
  - Register Indirect Addressing Mode
  - Implied Addressing Mode

- Register Addressing Mode
  - It is most common form of data addressing.
  - Transfer data from one register to other.
  - It is carried out with same size registers.
  - ***E.g MOV A,B*** (move data from source register B to Destination Register A)

- Immediate Addressing Mode
  - Data is specified in instructions itself.
  - Data immediately follows the hexadecimal opcode.
  - *E.g MVI C,3AH* (I implies immediate data and Data 3AH is moved to register C)
- Direct Addressing mode
  - The Address of data is defined in instruction itself.
  - *E.g LDA 2000H*(Load Accumulator with content of 2000H memory.)

1

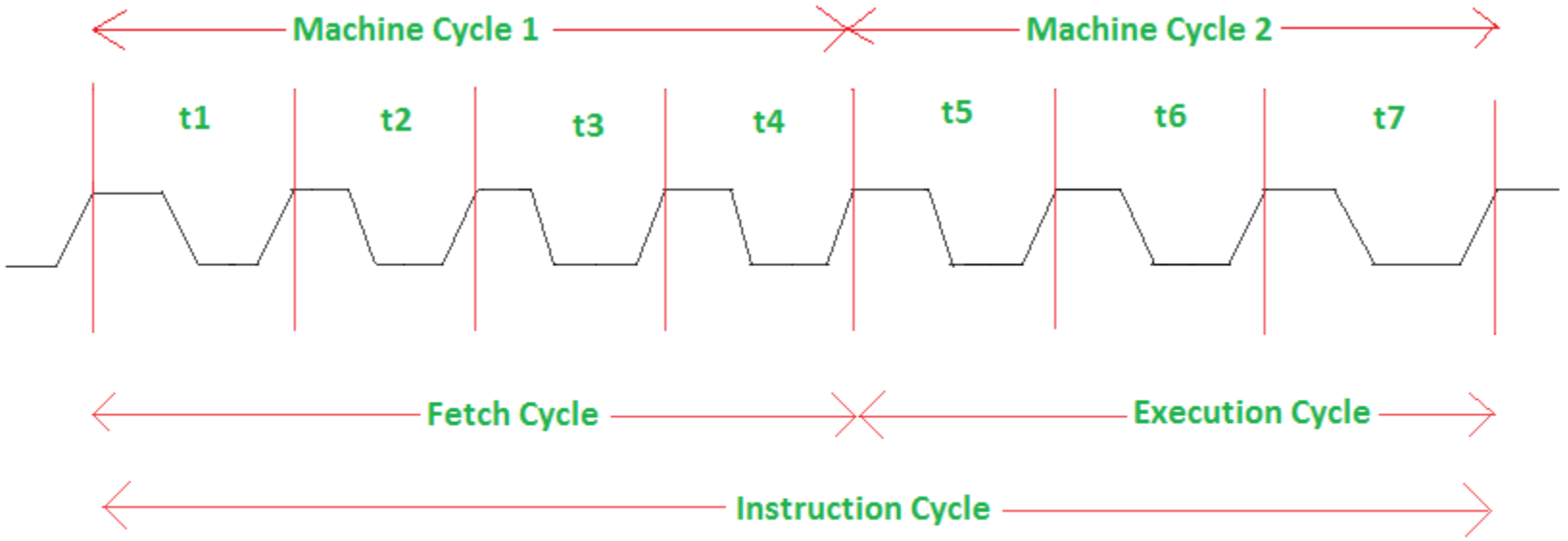
- Register Indiect Addressing Mode
  - In this mode address of operand is specified by register pair.
  - H pair, B pair or D pair.
  - ***E.g MOV C,M*** (M is memory location specified by H-L pair and address of operand is in H-L pair register)
- Implied Addressing Mode
  - The Addressing mode of certain instructions is implied by the instructions function.
  - ***E.g STC*** (set carry flag)
  - ***CMC*** (complement carry flag)

Instruction cycle

- The necessary steps that the CPU carries out to fetch an instruction and necessary data from memory and to execute it constitute an ***instruction cycle.***
- Instruction cycle is defined as time required to complete the execution of an instruction.
- An instruction cycle constitute of fetch and execute cycle.
- The necessary steps which are carried out to fetch an opcode from memory constitute **fetch cycle.**
- The necessary steps which are carried out to get data if any from memory and to perform the specific operation specified in the instruction constitute of ***execute cycle.***

Total time to execute the instruction

- IC=Fc+Ec



Instruction cycle in 8085 microprocessor

Fetch cycle

- First byte of instruction is its opcode.
- The program counter keeps the memory address of the next instruction to be executed in the beginning of fetch cycle.
- The content of PC, which is address of instruction to be fetch is send to memory.
- The memory places opcode to data bus so as to transfer to CPU.
- The entire process takes 2 clock cycle and in next one cycle instruction is decode.

Execution cycle

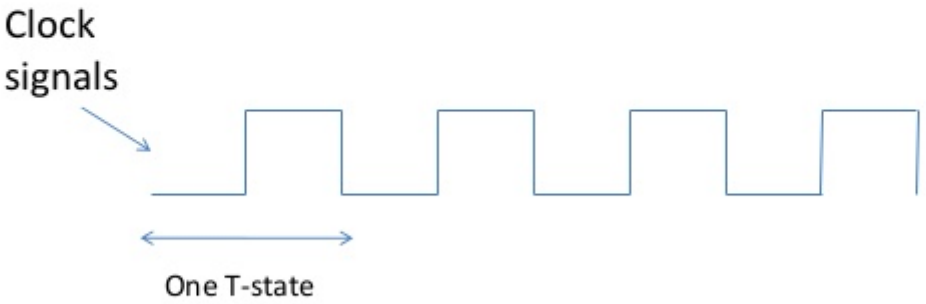
- The opcode fetch from memory goes to IR from IR it goes to decoder which decodes instruction after instruction is decoded execution begins.
- If operand is in register, the execution begins immediately and is completed in one clock cycle.
- If instruction contains data or operand address then CPU has to perform some read operation to get desired data.
- In some instruction write operation is performed. In write cycle data are send from CPU to memory or o/p device.
- In some cases execute cycle may involve one or more read or write cycle or both.

Machine Cycle

- It is defined as time required to complete one operation of accessing memory, I/O or acknowledging external request.
- This cycle may consist of 3 to 6 T states
- **T States**: It is defined as one subdivision of operation in one clock period. These subdivisions are internal states synchronized with system clock and each T states precisely equal to one clock period.

T-States

One clock period is called a T-state.



Time period of clock

Machine Cycles of 8085

- The 8085 has following basic machine cycle
  - opcode Fetch Cycle(4T)
  - Memory Read Cycle(3T)
  - Memory Write Cycle(3T)
  - I/O read Cycle(3T)
  - I/O write Cycle(3T)
  - Interrupt

Machine Cycle	Status			Control Signals		
	$IO/\overline{M}$	S1	S0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/0 Read	1	1	0	0	1	1
I/O Write	1	0	1	1	0	1
Interrupt Acknowledge	1	1	1	1	1	0
HALT	Z	0	0	Z	Z	1
HOLD	Z	X	X	Z	Z	1
RESET	Z	X	X	Z	Z	1

Where Z is tri state (pin neither connected to supply nor ground. High impedance) and X represents do not care.

8085 machine cycle status and control signals

Timing Diagram of opcode fetch machine cycle

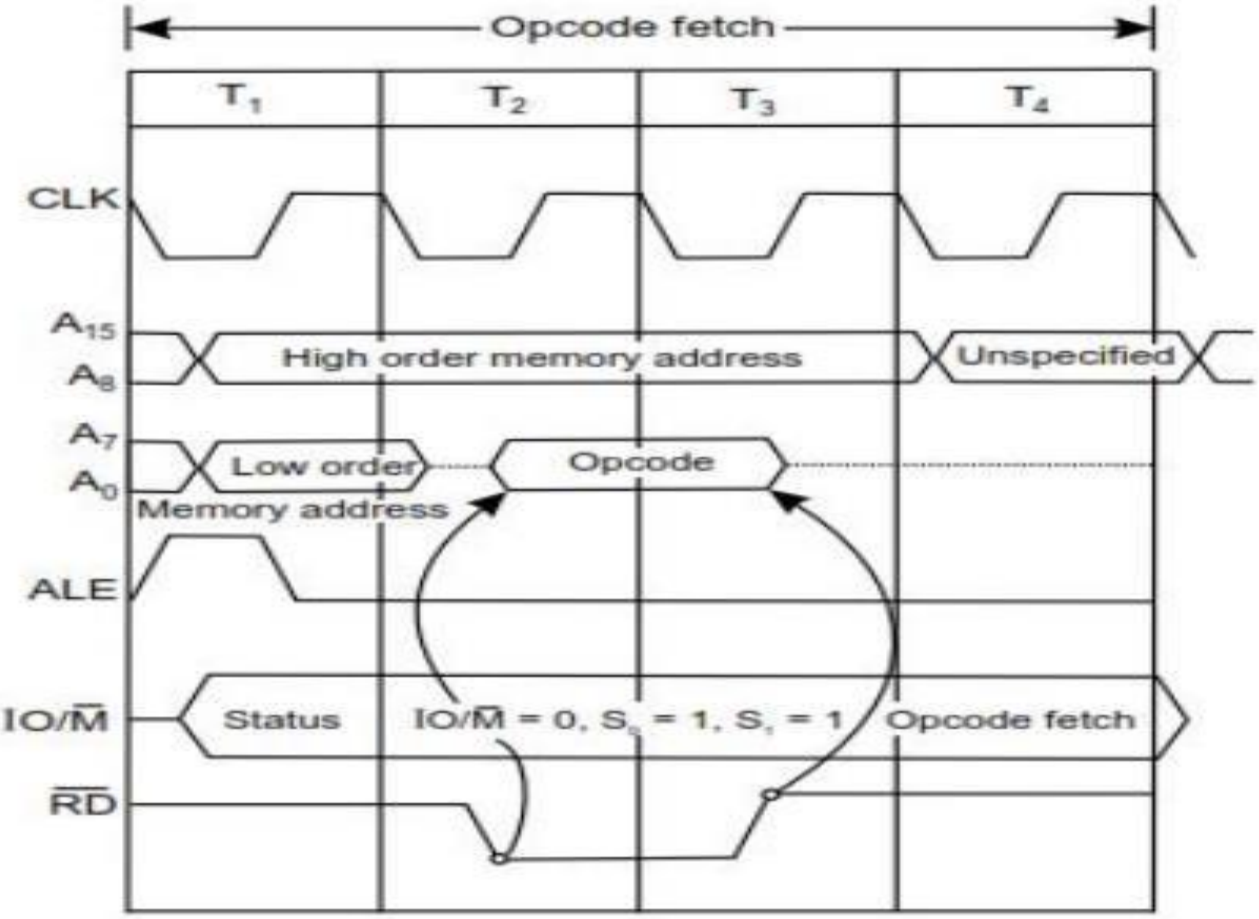


Fig 1.8 Opcode fetch machine cycle

OPCODE FETCH

- The Opcode fetch cycle, fetches the instructions from memory and delivers it to the instruction register of the microprocessor
- Opcode fetch machine cycle consists of **4 T-states**.

T1 State:

During the T1 state, the contents of the program counter are placed on the 16 bit address bus. The higher order 8 bits are transferred to address bus (A8-A15) and lower order 8 bits are transferred to multiplexed A/D (AD0-AD7) bus.

ALE (address latch enable) signal goes **high**. As soon as ALE goes high, the memory latches the AD0-AD7 bus. At the middle of the T state the **ALE goes low**

T2 State:

During the beginning of this state, the **RD' signal goes low** to enable memory. It is during this state, the selected memory location is placed on D0-D7 of the Address/Data multiplexed bus.

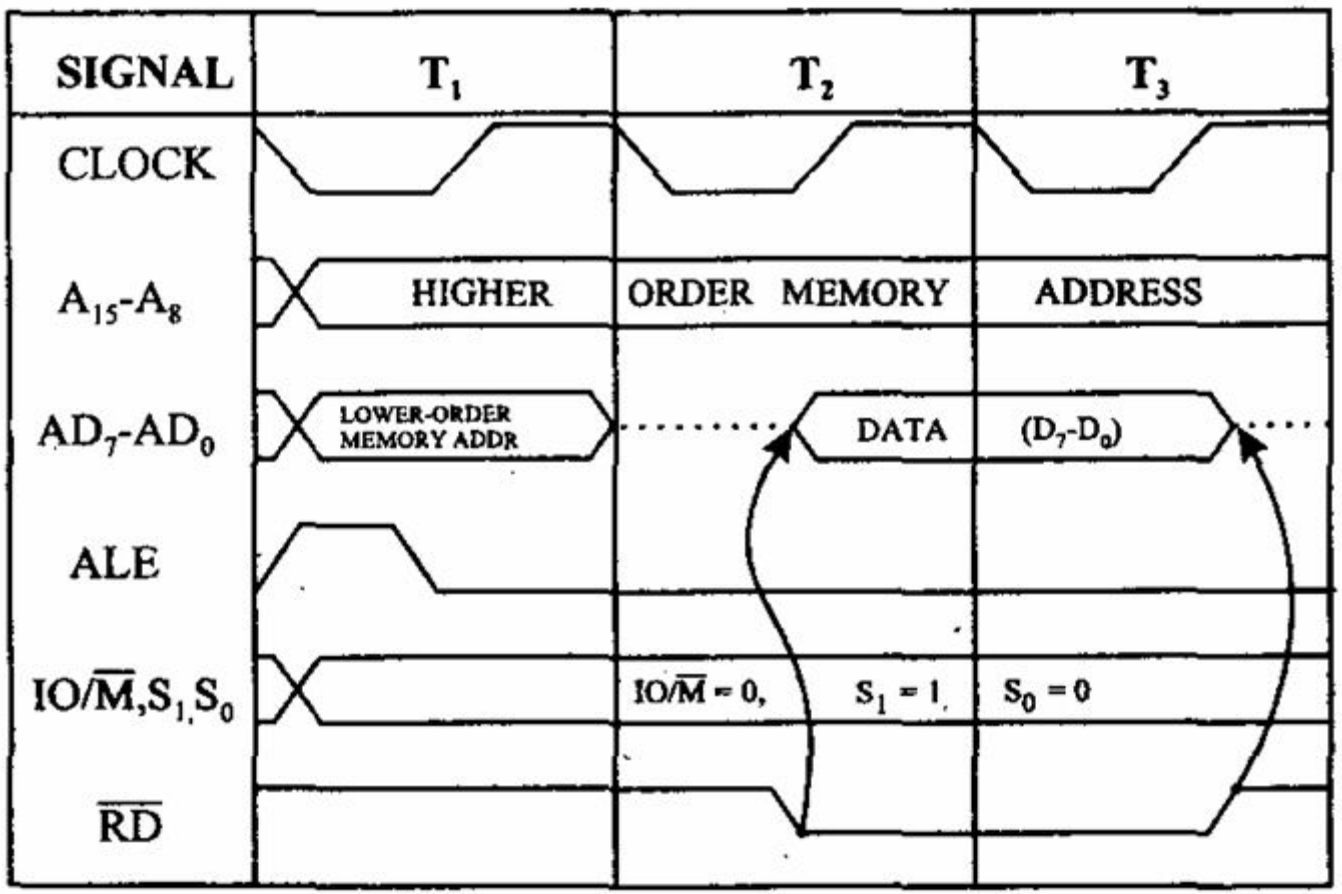
T3 State:

In the previous state the Opcode is placed in D0-D7 of the A/D bus. In this state of the cycle, the Opcode of the A/D bus is transferred to the instruction register of the microprocessor. Now the **RD' goes high** after this action and thus disables the memory from A/D bus.

T4 State:

In this state the Opcode which was fetched from the memory is decoded.

Timing Diagram of memory read cycle



- These machine cycles have 3 T-states.

T1 state:

- The higher order address bus **(A8-A15)** and lower order address and data multiplexed (AD0-AD7) bus. **ALE goes high** so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.

  The mp identifies the memory read machine cycle from the status signals **IO/M'=0, S1=1, S0=0**. This condition indicates the memory read cycle.

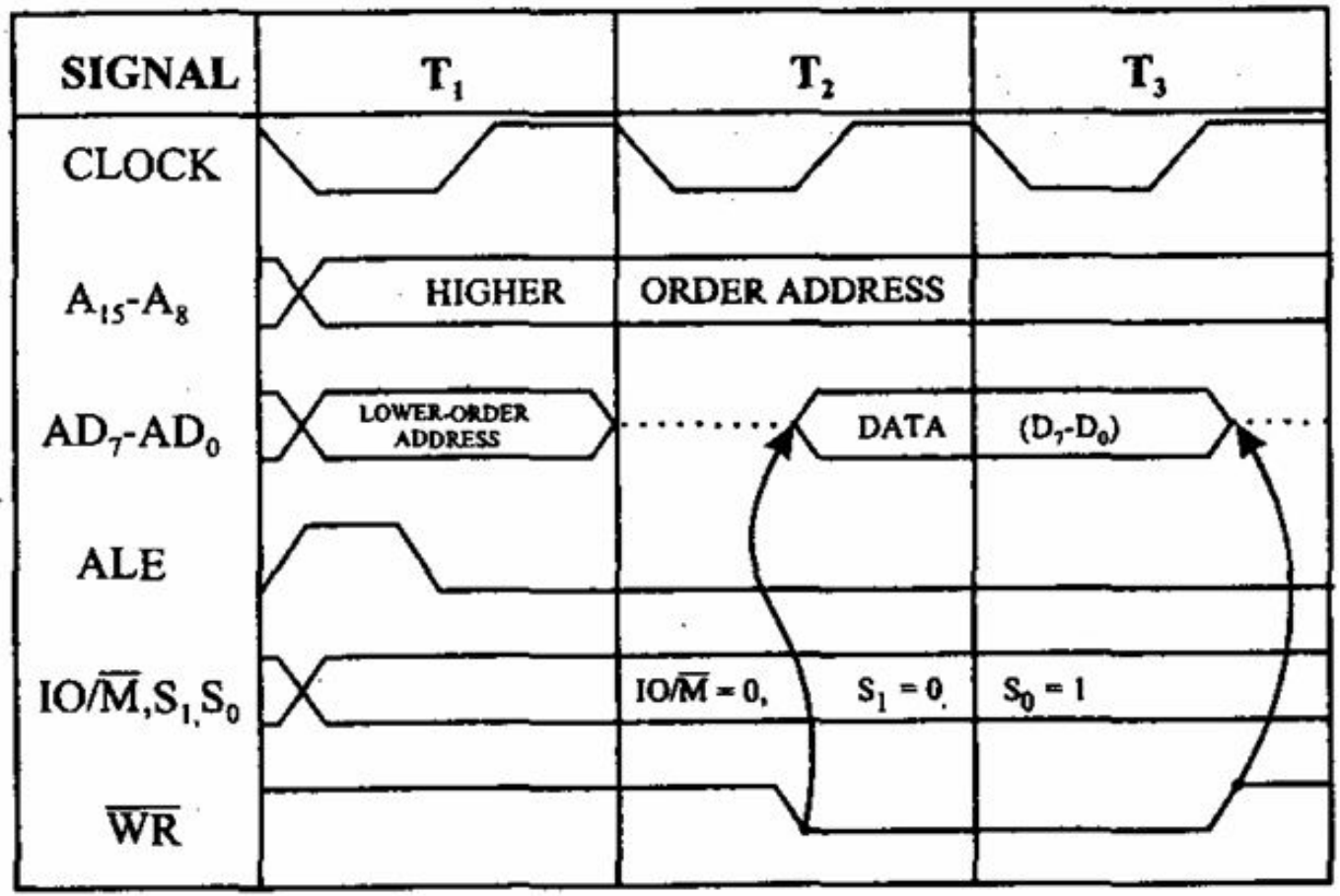
T2 state:

- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. RD' goes **LOW**

T3 State:

- The data which was loaded on the previous state is transferred to the microprocessor. In the middle of the T3 state RD' goes high and disables the memory read operation. The data which was obtained from the memory is then decoded.

Timing Diagram of memory write cycle



- These machine cycles have 3 T-states.

T1 state:

- The higher order address bus **(A8-A15)** and lower order address and data multiplexed (AD0-AD7) bus. **ALE goes high** so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.

  The mp identifies the memory read machine cycle from the status signals **IO/M'=0, S1=0, S0=1**. This condition indicates the memory read cycle.

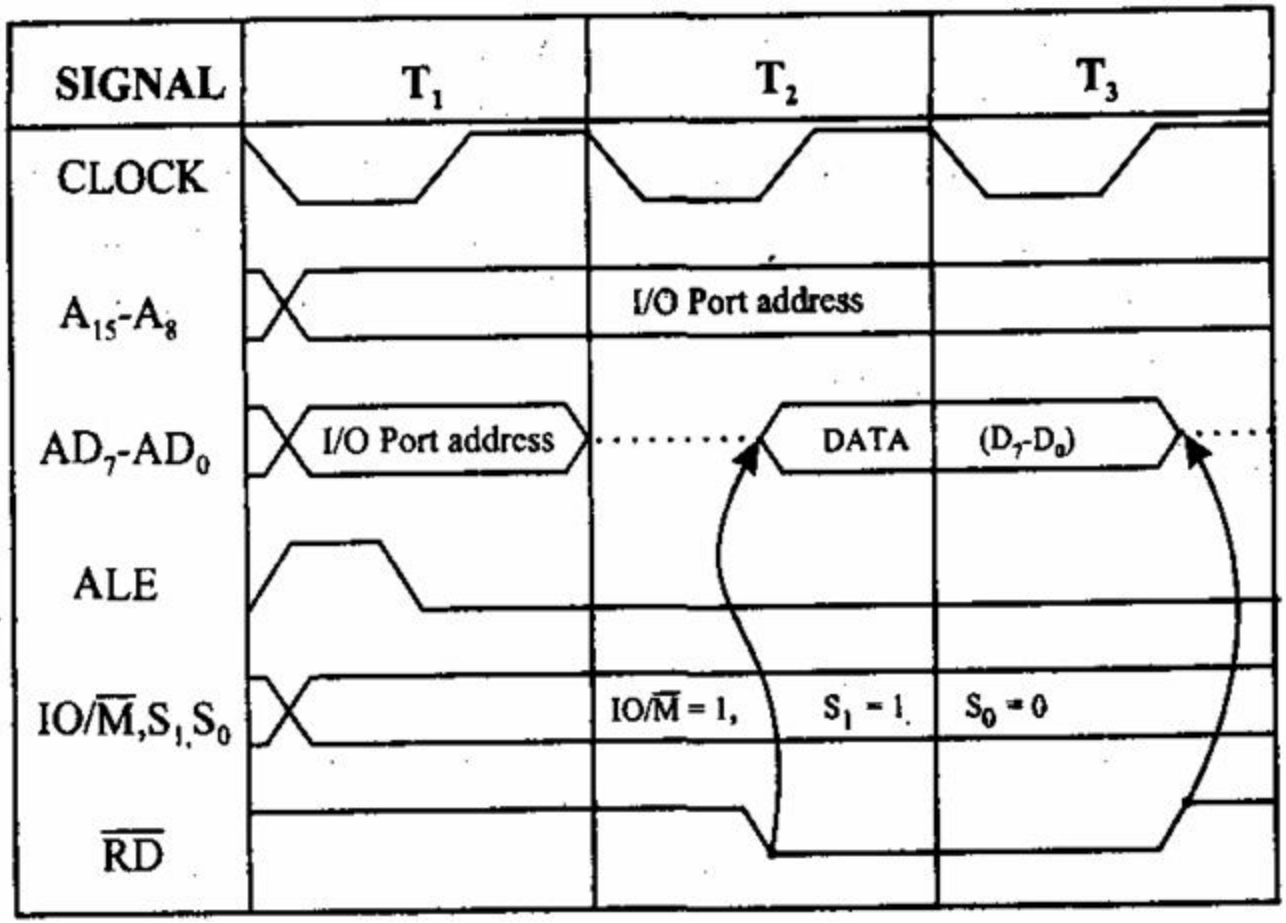
T2 state:

- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. WR' goes **LOW**

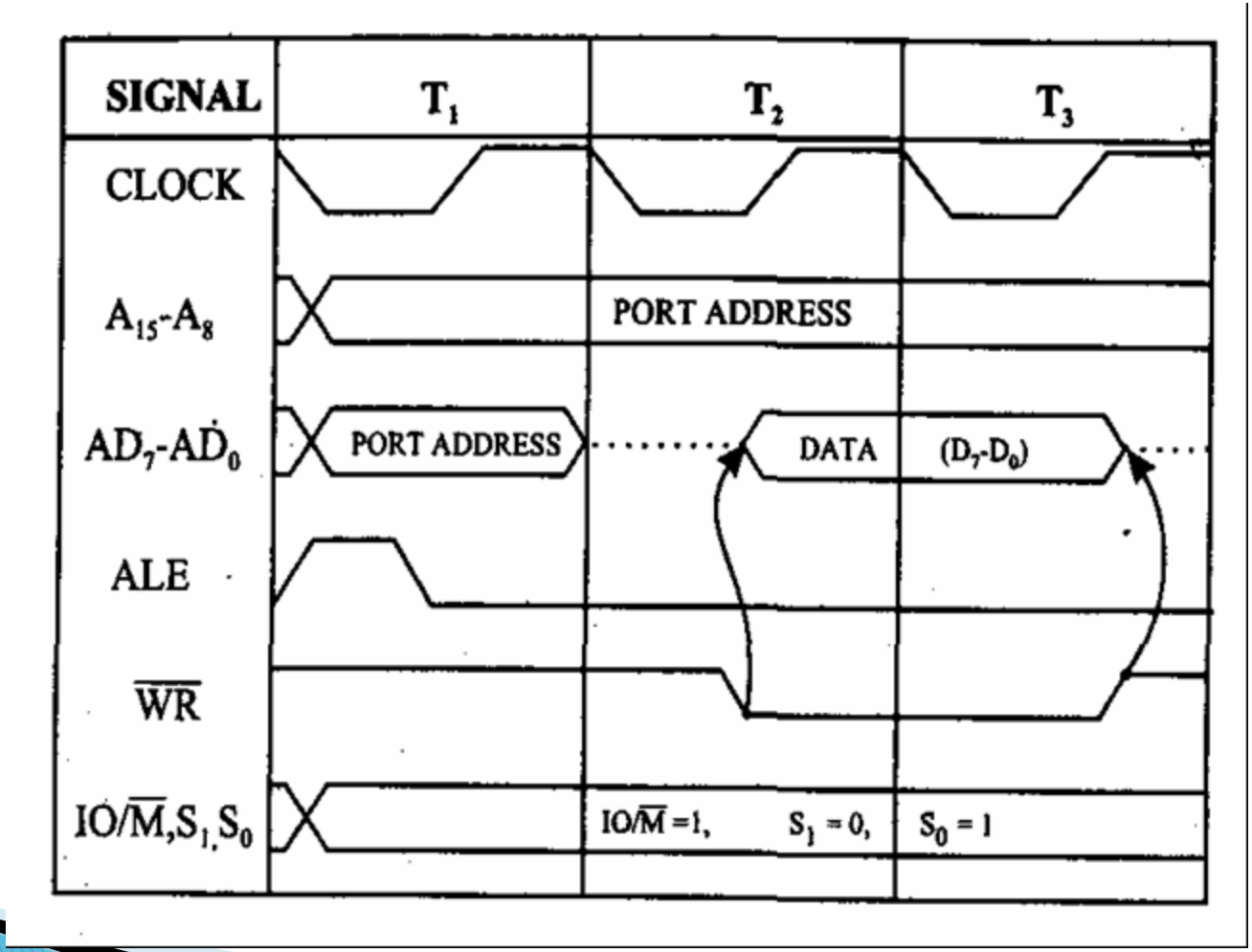
T3 State:

- In the middle of the T3 state WR' goes high and disables the memory write operation. The data which was obtained from the memory is then decoded.

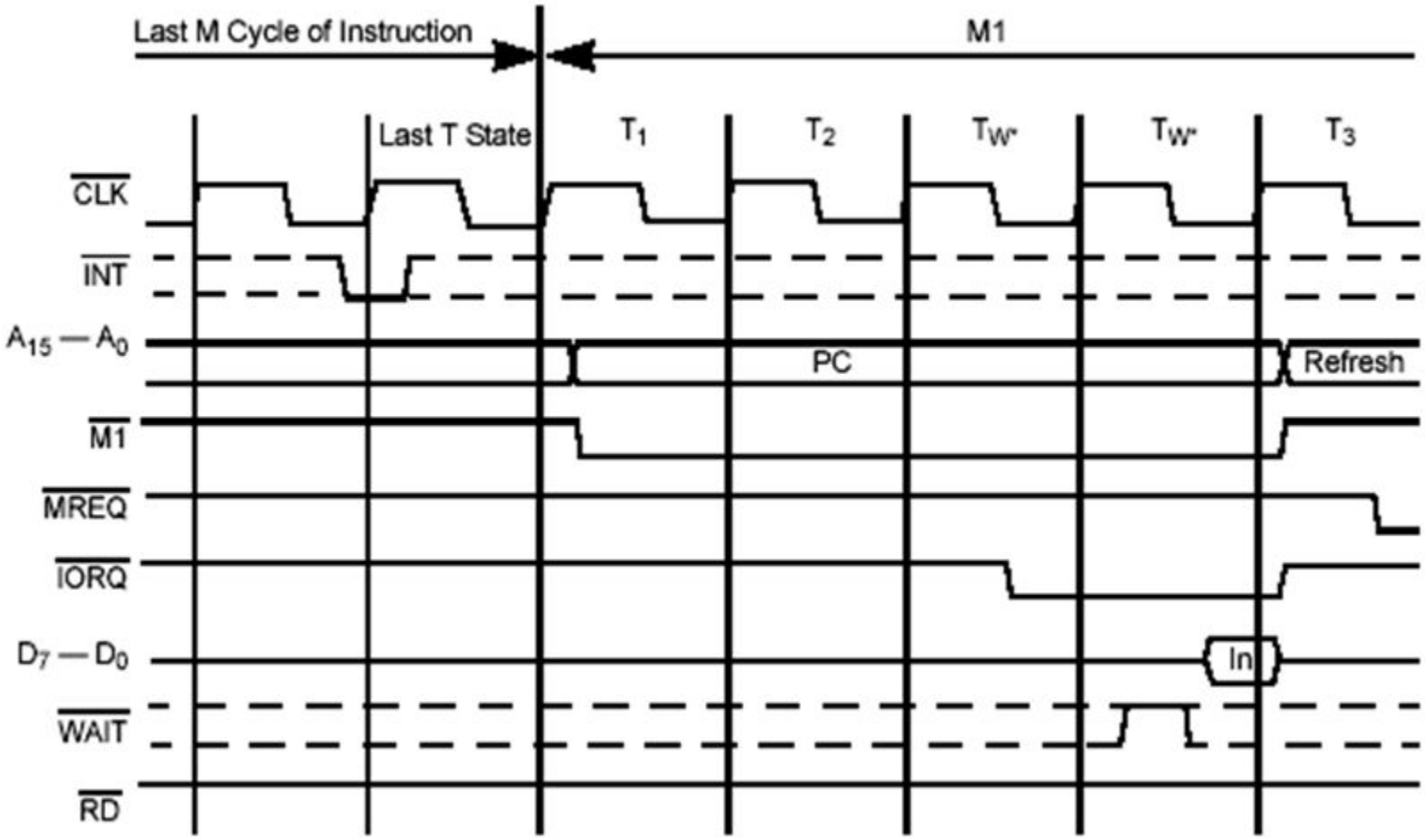
Timing Diagram of I/O read cycle



Timing Diagram of I/O write cycle

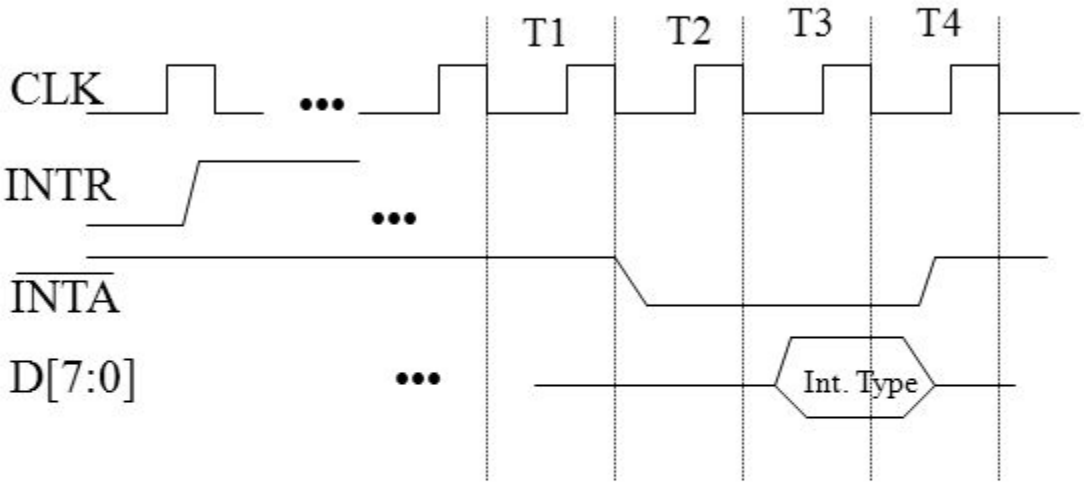


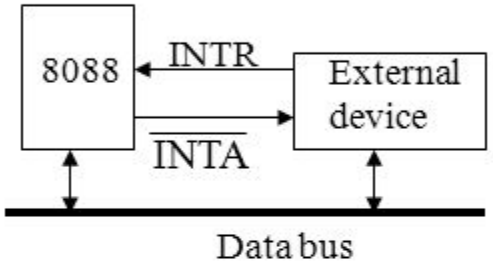
Interrupt Request/Acknowledge Cycle



Two wait states are automatically added to this cycle

Interrupt Acknowledge Timing Diagrams





- It takes one bus cycle to perform an interrupt acknowledge
- During T1, the process tri-states the address bus
- During T2, INTA is pulled low and remains low until it becomes inactive in T4
- The interrupting devices places an 8-bit interrupt type during $\overline{\text{INTA}}$ is active

Timing Diagram of opcode machine cycle

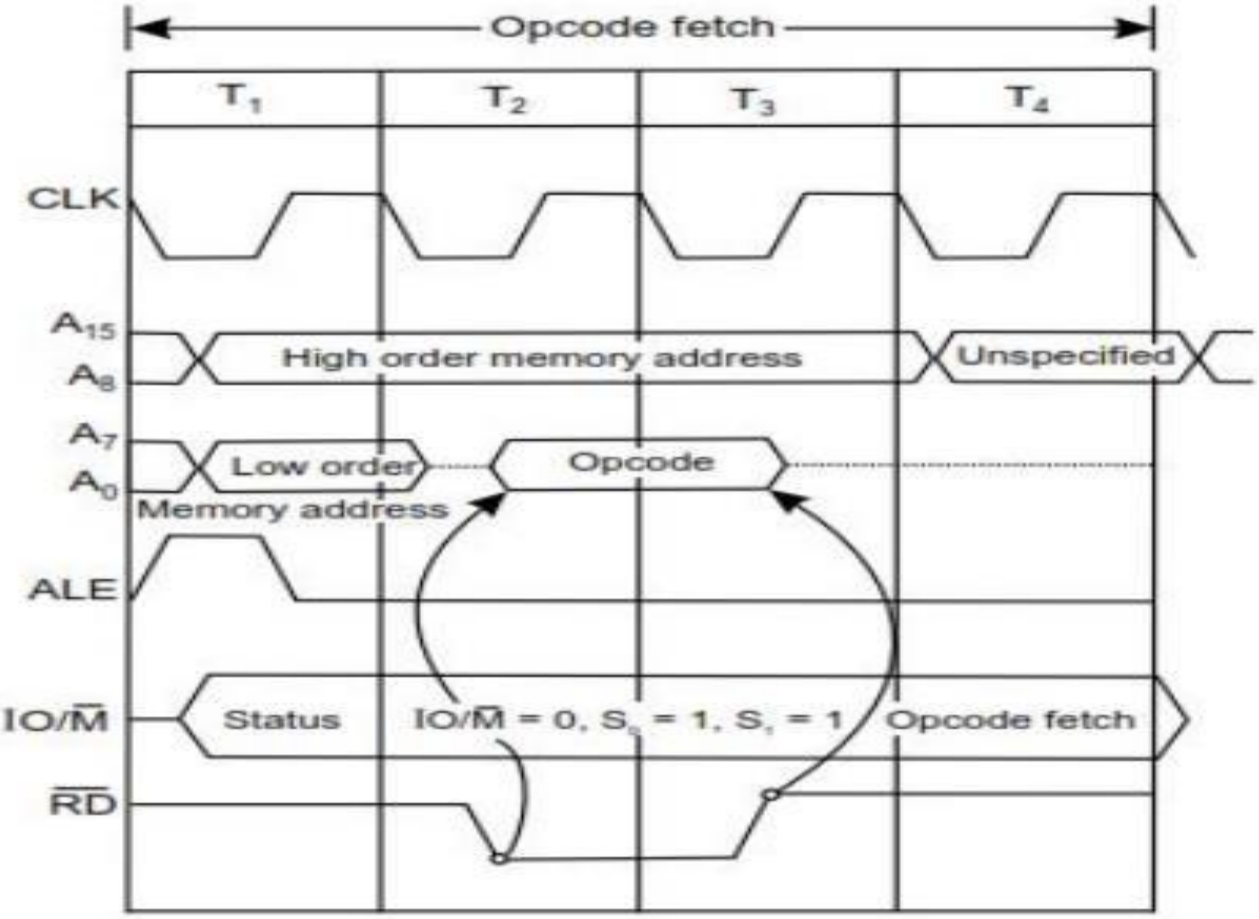
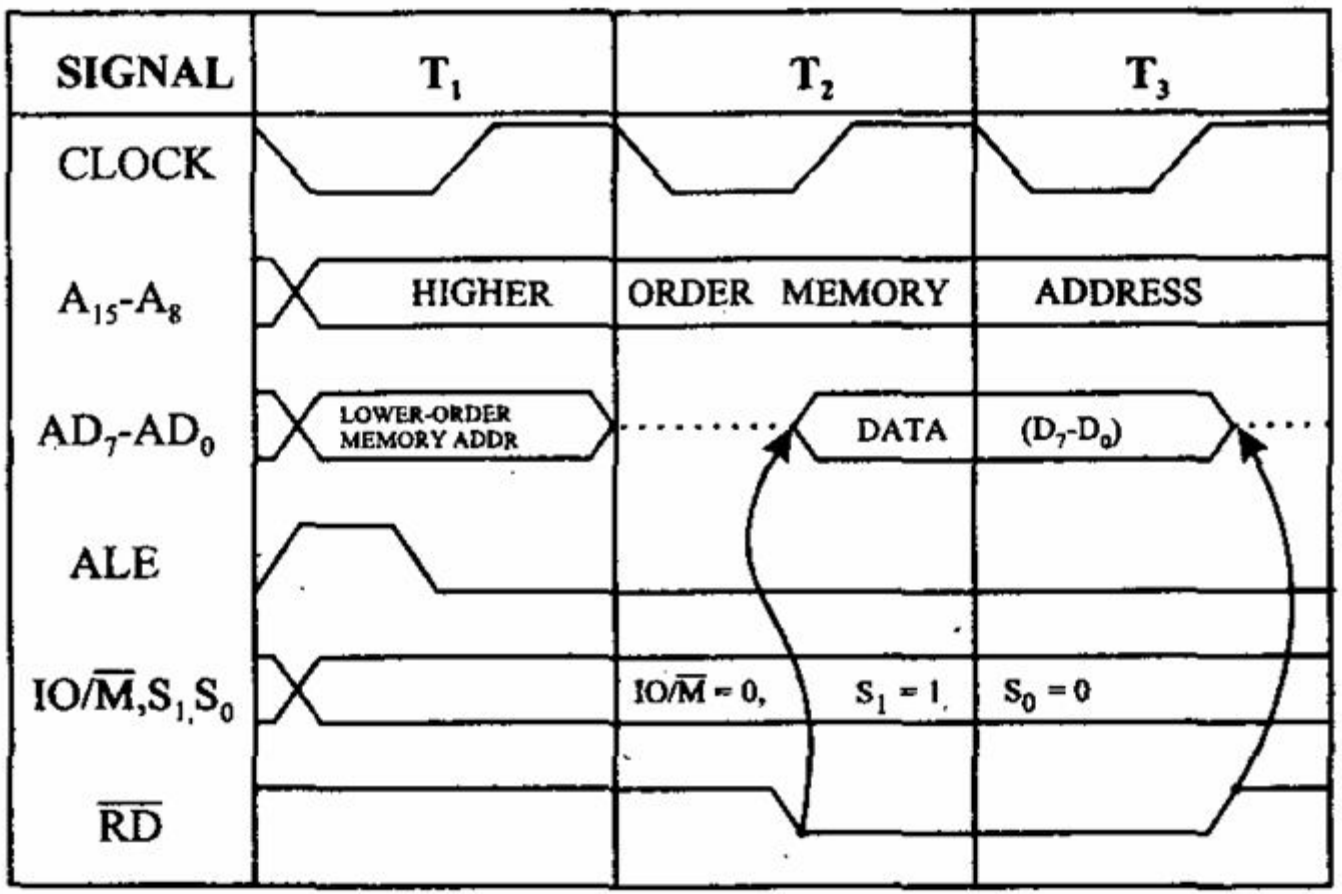


Fig 1.8 Opcode fetch machine cycle

Timing Diagram of memory read cycle



Introduction to Assembly language Programming

Unit 2

Mnemonics

- Mnemonics is a Greek word called mindful(mind aid).
- Manufacturer of microprocessor has devised a symbolic code for each instruction called mnemonics.
- E.g 0011 1100 in 8085 is INR A.

Assembly language and High level languages

- Each device has its own **set of instruction** depending on design of CPU.
- To communicate with computer one must give instructions in **binary form**.
- It is difficult for most people to write programs in sets of 0's and 1's.
- So, microprocessor manufacturer introduce **English like words** to represents the binary instruction of machine.

- Programmer can write programs called **assembly language** programs using **mnemonics.**
- Assembly language program are **non transferrable** from one machine to other.
- Programming languages are of two types-
  - **low level language**
    - Machine language
    - Assembly language
  - **High level language** (machine independent like BASIC,FORTAN)

- High level languages are **machine independent and are transferrable.**
- Machine and assembly languages are microprocessor specific
- Since a program can be written in high level or assembly language but microprocessor only understand binary so it must be **translated to binary code or machine code**
- **The program written in assembly languages are translated into a binary code by using a program called an assembler.**

- Programming with Intel 8085 microprocessor

Introduction to assembly language programming

- Assembly language use two three or four letter mnemonics to represent each instruction type.
- Low level assembly language is designed for a specific family of processor
- The symbolic instruction of assembly language are assembled into machine language
- Assembly language makes writing program easier than machine code.
- Later it is translated into machine language and can be loaded into memory and run.

Advantage of assembly language

- Requires less memory and execution time than high level language.
- Gives programmer the ability to perform highly technical tasks.
- Provides more control over handling particular hardware requirement.
- Generates smaller and compact executable modules.

Example of machine-language

Here's what a program-fragment looks like:

```
10100001 10111100 10010011 00000100
00001000 00000011 00000101 11000000
10010011 00000100 00001000 10100011
11000000 10010100 00000100 00001000
```

It means: $z = x + y;$

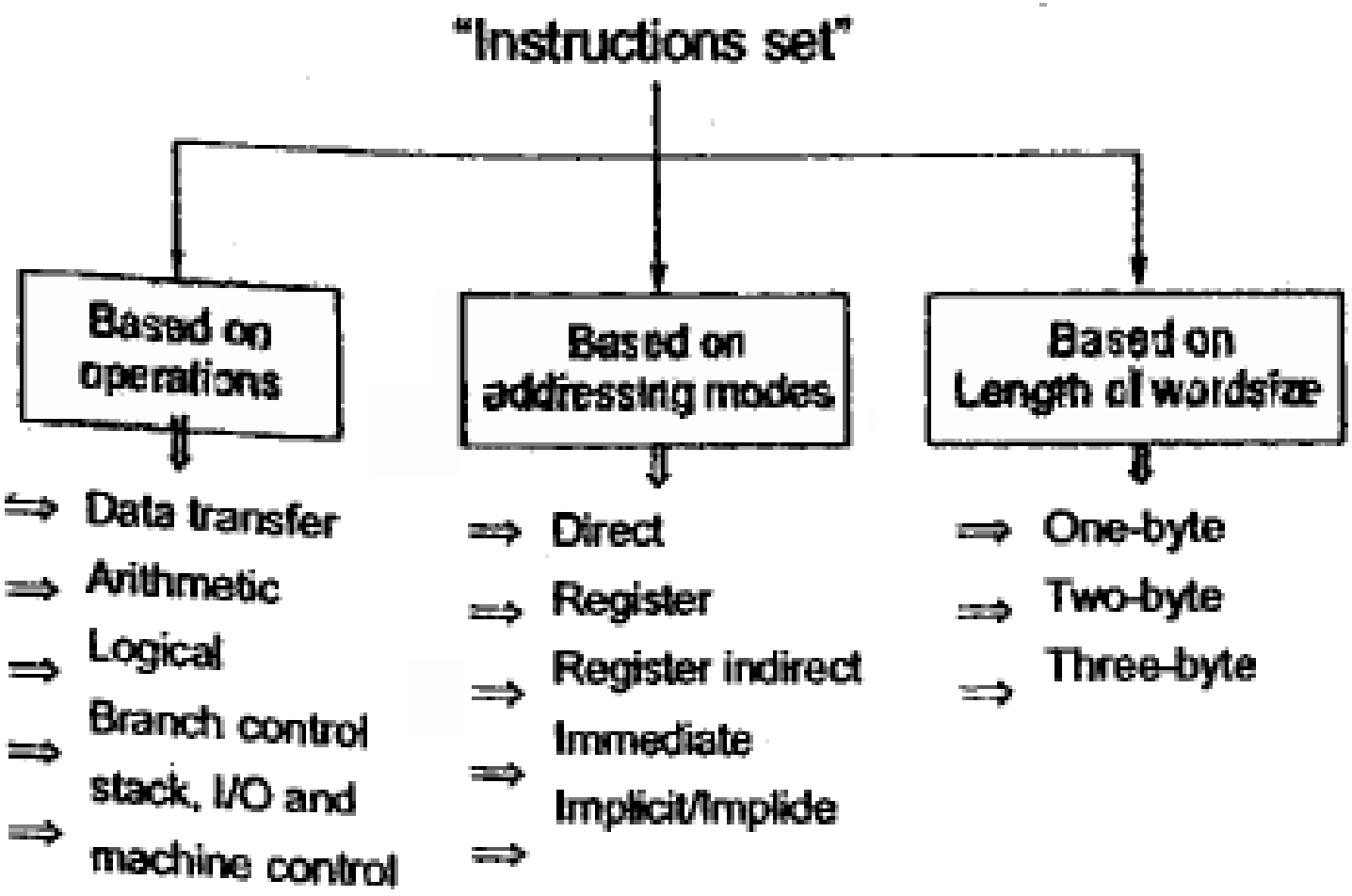
Typical Format of Assembly Language Instruction

Label	Opcode Field	Operand Field	Comments
Next:	ADD	AL,07H	;Add correction factor

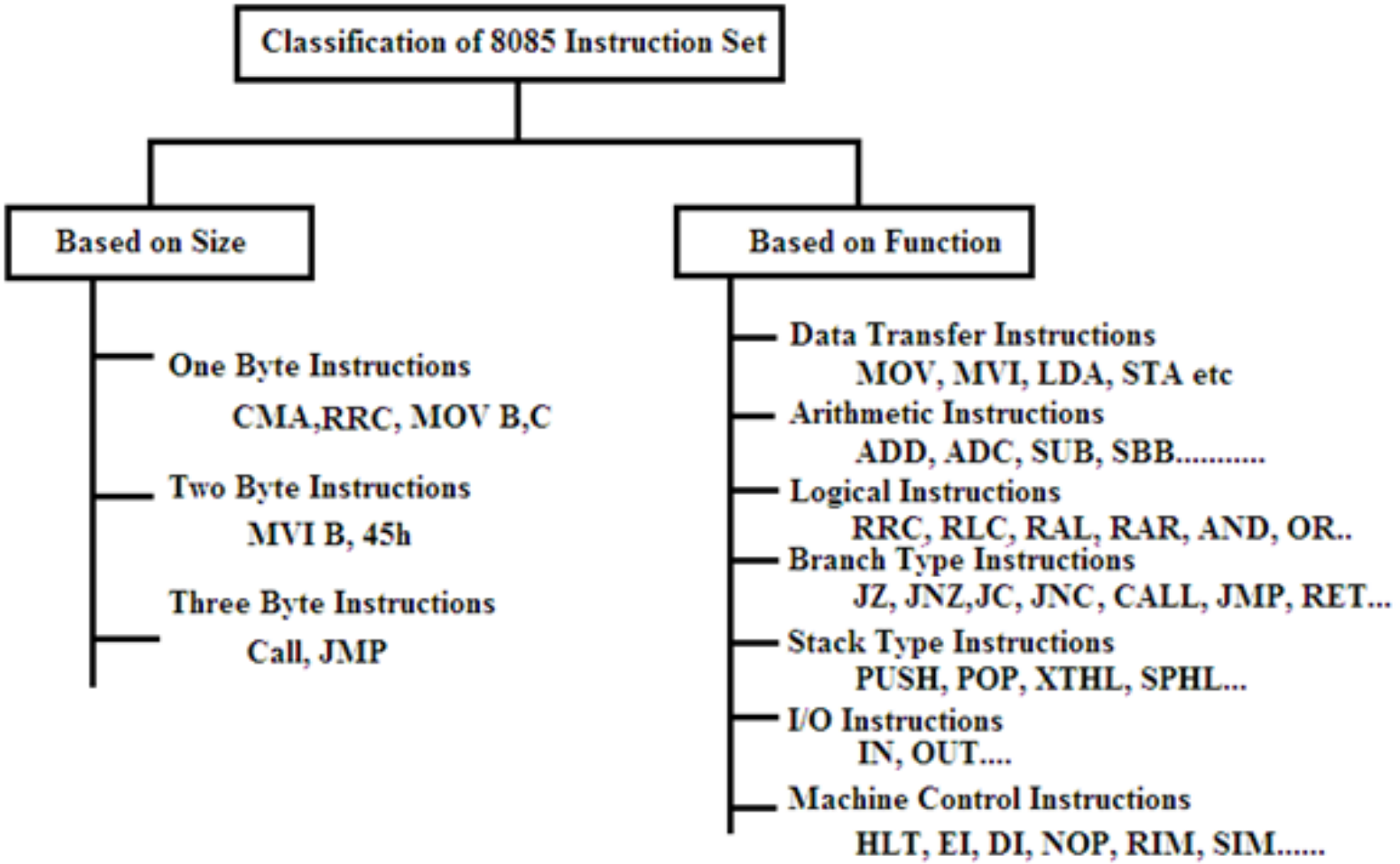
ADD R1, R0

- Here R0 is source Register & R1 is destination register
- Instruction adds contents of R0 with the contents of R1 and stores result in R1 register.
- 8085 Can handle maximum of 256 instructions.

Classification of Instruction sets



Classification of 8085 Instruction sets



Instruction format

- Depending on the number of address specified, the Instruction format can be classified into three categories:
  - One Byte Instruction:
    - Here one byte will be operand.
    - E.g MOV A,B
  - Two Byte Instruction:
    - First byte will be opcode and second byte will be operand/data.
    - E.g MVI A, data
  - Three Byte Instruction:
    - First byte opcode and second , third byte will be operand/data.
    - E.g LXI B,4050H

Instruction Set of 8085

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions that a microprocessor supports is called ***Instruction Set***.
- 8085 has **246** instructions.
- Each instruction is represented by an 8-bit binary value.
- These 8-bits of binary value is called ***Op-Code*** or ***Instruction Byte***.

Intel 8085 uses the following addressing modes:

1. Direct Addressing Mode
2. Register Addressing Mode
3. Register Indirect Addressing Mode
4. Immediate Addressing Mode
5. Implicit Addressing Mode

1. Direct Addressing Mode

In this mode, the address of the operand is given in the instruction itself.

LDA 2500 H	Load the contents of memory location 2500 H in accumulator.

- LDA is the operation.
- 2500 H is the address of source.
- Accumulator is the destination.

2. Register Addressing Mode

In this mode, the operand is in general purpose register.

MOV A, B	Move the contents of register B to A.

- MOV is the operation.
- B is the source of data.
- A is the destination.

3. Register Indirect Addressing Mode

In this mode, the address of operand is specified by a register pair.

MOV A, M	Move data from memory location specified by H-L pair to accumulator.

- MOV is the operation.
- M is the memory location specified by H-L register pair.
- A is the destination.

4. Immediate Addressing Mode

In this mode, the operand is specified within the instruction itself.

MVI A, 05 H	Move 05 H in accumulator.

- MVI is the operation.
- 05 H is the immediate data (source).
- A is the destination.

5. Implicit Addressing Mode

If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction.

CMA	Complement accumulator.

- CMA is the operation.
- A is the source.
- A is the destination.

Writing or Assembling a program

- Step 1: Read problem carefully
- Step 2: Break it down into small steps
- Step 3: Represent these small steps in a possible sequence with a flow chart
- Step 4: Translate the block of flowchart into appropriate mnemonic instructions
- Step 5: Translate mnemonic into machine code
- Step 6: Enter the machine code in memory and execute
- Step 7: Start troubleshooting(debugging a program)

Executing a program

- The machine code can be loaded into read write memory starting with some memory address
- The execution of program can be done in two ways
  - 1. The first is to execute entire code by pressing execute key
  - 2. Second is to use the single step key. It executes one instruction at time. By using examine register key we can observe the contents of registers and flags after each instruction being executed.

Debugging a program

- Debugging binary code is much more difficult and cumbersome.
- Essentially we search carefully for the errors in the program logic, machine codes and execution
- The debugging process can be divided into two parts,
  - Static Debugging: Visual Inspection of a flowchart and machine code
  - Dynamic Debugging: observing output or register contents following a execution of each instruction (Single step technique) or Group of Instructions (Breakpoint Technique)

Developing Counters and Time Delay Routines

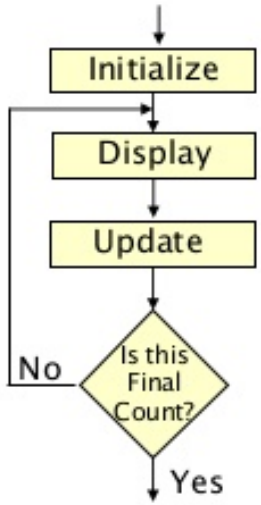
- Designing counter is a frequent programming application.
- Counters are used to keep track of event.
- Time delays are important to set up reasonably accurate timing between two events.
- Process of designing counters and time delay using software instructions is more flexible and less time consuming than design process using hardware.

Counter

- A counter is designed simply by loading an appropriate number into one register and using INR(increment by 1) and DCR (decrement by 1) instructions.
- A loop is established to update a the count and each count is checked whether it has reached final number. If not, loop is repeated.
- Drawback of such counter is counting perform at very high speed that only last count is observed.
- To observe counting time delay between count must be used.

Counter

COUNTERS



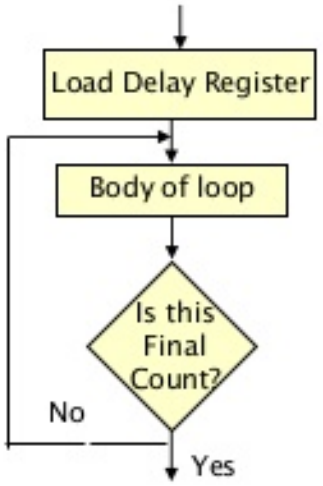
Flowchart of a Counter

A Counter is designed simply by loading an appropriate number into one of the registers and using the INR (Increment by one) or the DCR (Decrement by one) instructions. A loop is established to update the count, and each count is checked to determine whether it has reached the final number; if not, the loop is repeated.

Time delay

- Designing Time delay is similar to that used to set up counter.
- The register is loaded with number depending on time delay required.
- And register is decremented until it reaches zero by setting up loop with conditional jump instruction.
- The loop causes delay depending on clock period of system.

TIME DELAYS



Flowchart of a Time Delay

The procedure used to design a specific delay is similar to that used to set up a counter. A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero by setting up a loop with a conditional jump instruction. The loop causes the delay, depending upon the clock period of the system.

Basic Computer Architecture

Unit 3

- Computer Architecture
  - **Computer Architecture** refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.
- Computer Organization
  - **Computer organization** refers to the operational units and their interconnections that realize the architectural specifications.

History of Computer Architecture

Computer Architecture

- The design ,arrangement, construction or organization of the different parts of a computer system is known as Computer Architecture
- Computer architecture defines the way hardware components are connected together to form a computer system
- Computer architecture deals with high level design issues such as Instruction sets, Addressing modes, Data types, Cache optimization etc.
- It helps us to understand the functionalities of a system.
- It acts as an interface between hardware and software

Brief History of Computer Architecture

- First Generation (1945-1958)

- Vacuum Tubes
- Machine code
- CPU was unique to that machine
- Use of drum or magnetic core memory, programs are loaded using paper tape or punch cards
- 2 Kb memory, 10 KIPS(thousands instruction per second)

Architecture for a Computing Machine

- Based on the storage and signaling of instruction and data.
- **Harvard architecture:** uses physically separate storage and signal pathways for instructions and data. (Harvard Mark I). it stored instructions on punched tape and data in relay latches.
- **Von Neumann architecture:** a single storage structure to hold both set of instructions and data Aka stored program computers.

Second Generation(1958-1964)

- Transistors – small, low-power, low-cost, more reliable than vacuum tube
- Magnetic core memory
- Reduced computational time to microseconds
- High level languages
- First OS handled one program at a time

Third Generation(1964-1974)

- Integrated circuits, combined thousands of transistors on a single chip using LSI
- Semiconductor Memory
- Timesharing, Graphics and Structured programming
- 2 Mb memory, 5 MIPS
- Use of Cache memory

Fourth Generation(1974- Present)

- Microprocessor, combined millions of transistors using VLSI and ULSI technology
- Single Chip processor and the single board computer emerged
- Creation of the Personal computer(PC)
- Object oriented programming and Artificial intelligence concept
- Intel 4004 was the worlds first universal microprocessor.

Self Study !!

- ENIAC
- Whirlwind computer
- UNIVAC
- IBM 7090
- IBM System/360
- Intel 4004

Microoperations

- The operation executed on data stored in registers are called micro operations.
- A micro operation is elementary operation performed on the information stored in one or more registers.
- The result of operation may replace the previous binary information of register or may be transferred to other register.
- Example: shift, count, clear and load.

Instruction Codes

- **Basic computer is smaller in comparison to commercial computer.**
- **Organization of the computer is** defined by its **internal register**, the **timing and control structure** and **set of instructions** and its uses.
- The general purpose digital computer is capable of executing various **sequence of micro operations.** It performs on data stored in registers
- The user of computer can control the process by means of a program which is set of instructions that specify the operations and operand.

Instruction Codes

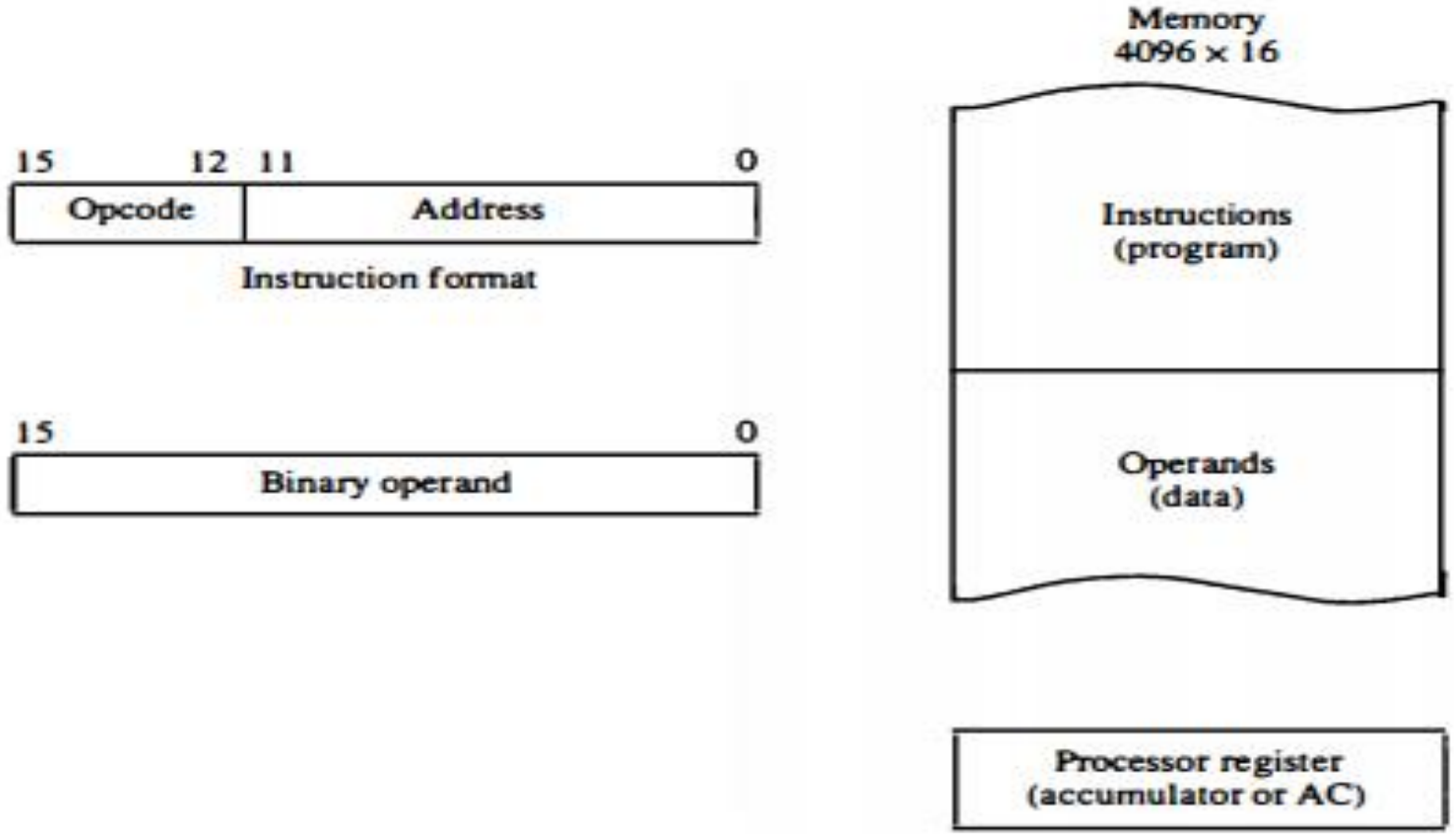
- **Computer instruction**
  - Is a binary code that **specifies a sequence of micro operations for computer**
  - Instruction code and data are **stored in memory**
  - Computer read each instruction from memory and keep in register
  - Control **interprets** the binary code of instructions and proceed to execute it by issuing a sequence of micro operations.
  - Every computer has its own unique instruction set.

Instruction Codes

- Instruction code is group of bits that instruct computer to perform a specific operations.
- It is usually divided into 2 parts:
  - **Operation code**
    - If n bits are in operation code, then there are 2^n distinct operations.
    - Operation can be add, subtract, multiply, shift, complement etc.
    - The control unit receives the instruction from memory and interprets the operation code bits and then issues a control signals to initiate microperations on internal computer registers.
  - **Operand code**

Stored Program Organization

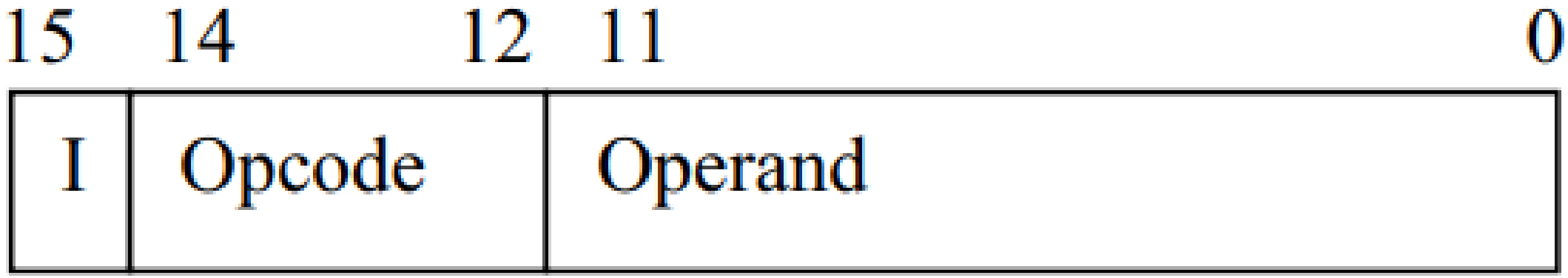
Figure 5-1 Stored program organization.



Addressing Modes of basic computer (Direct and Indirect addressing mode)

- 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 the address of a memory word in which the address of the operand is available, the instruction is said to have **Indirect address.**

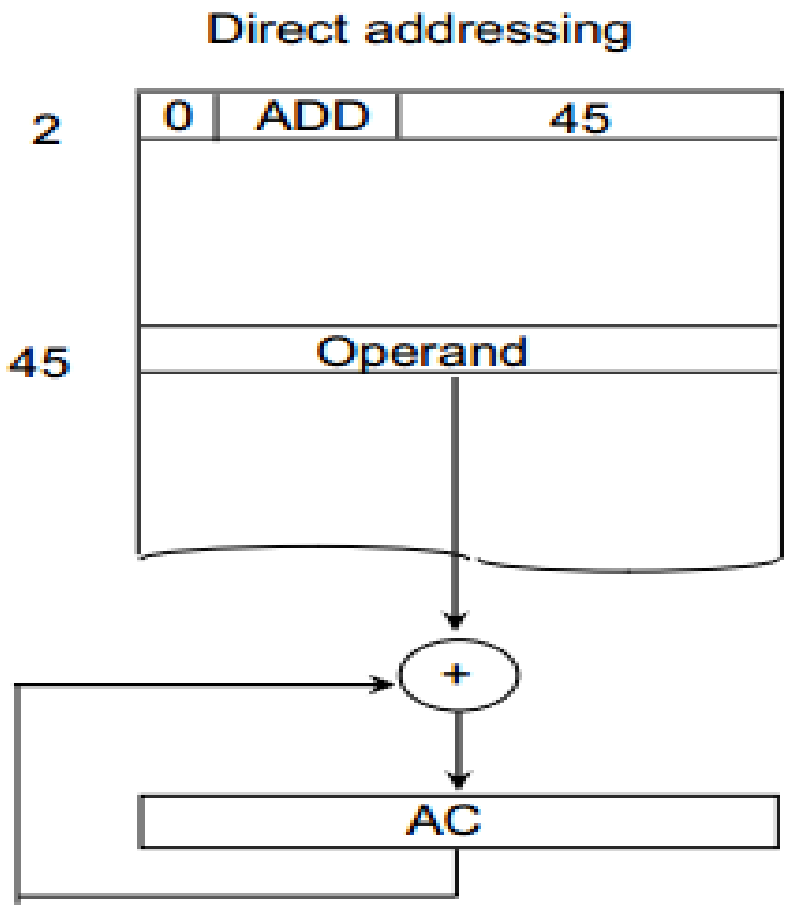
Addressing Modes of Basic computer (Direct and Indirect addressing mode)

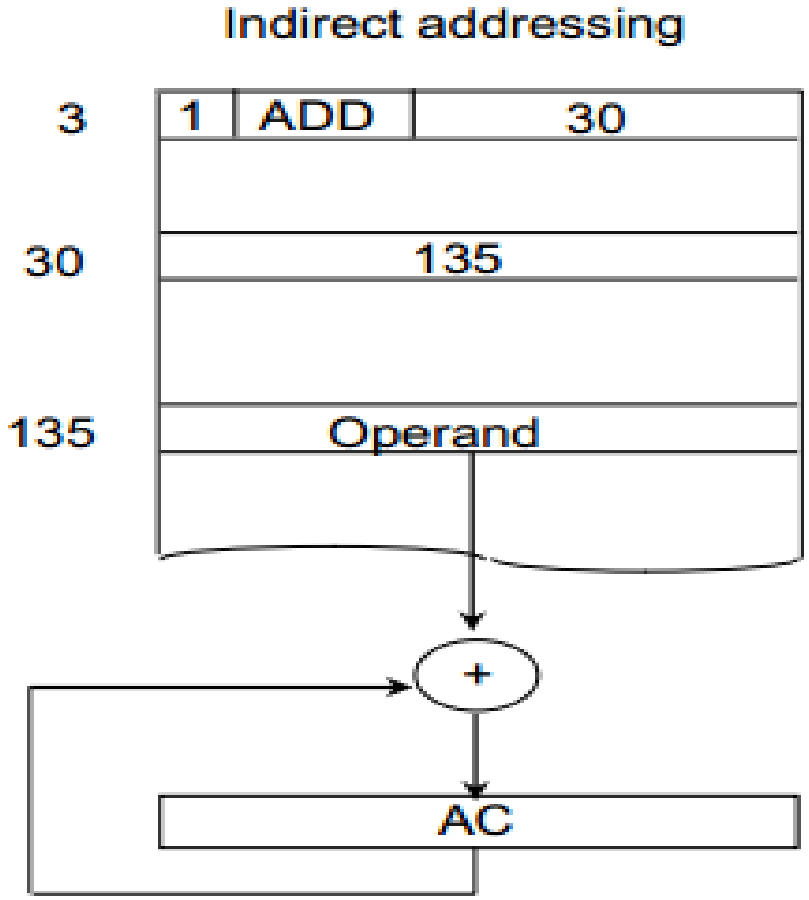


Addressing Modes of Basic computer (Direct and Indirect addressing mode)

- 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.
  - Second reference is for the operand itself.

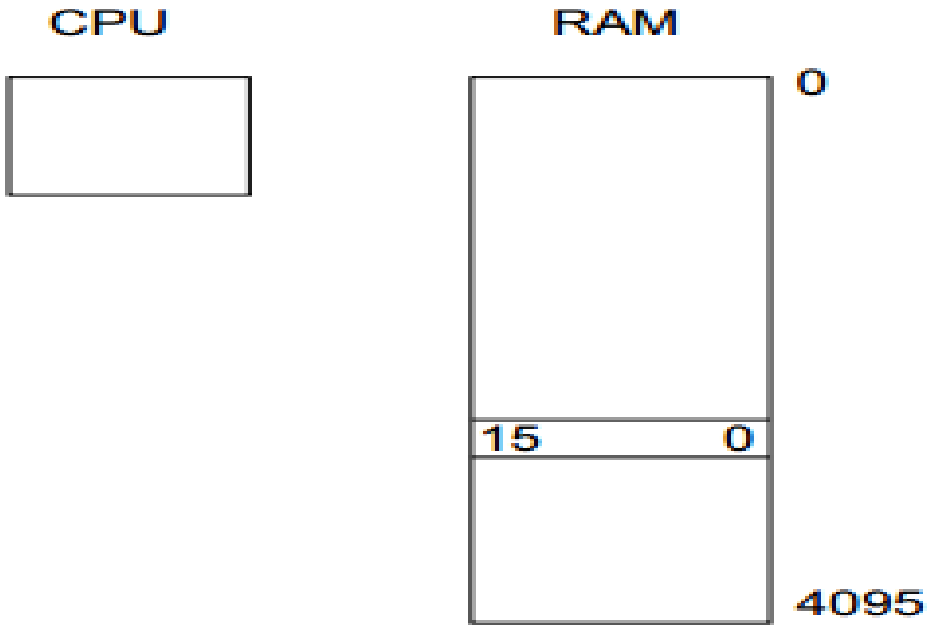
Addressing Modes of basic computer (Direct and Indirect addressing mode)





Computer Registers

- Basic computer have memory and Processors.
- Memory have capacity=4096 words=2^{12}
- 1 word=16 bits



Computers Registers

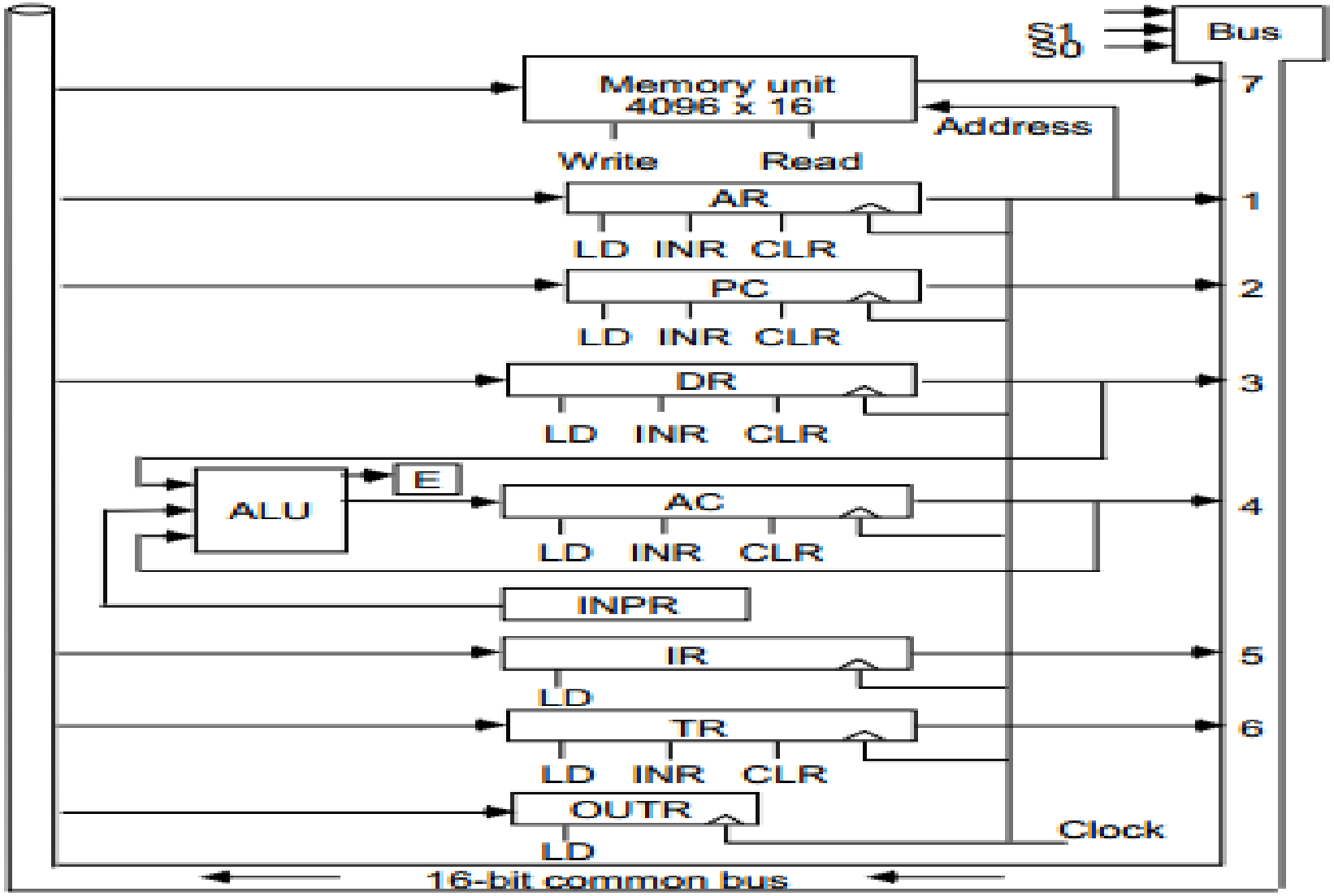
- DR (Data registers)–To hold operand read from memory. **16 bits**
- AC (Accumulator)– General purpose register, used for storing intermediate arithmetic and logic results. **16 bits**
- IR(Instruction Register)– Instruction fetched from memory are placed in IR. **16 bits**
- TR(Temporary Registers)– used for holding temporary data. **16 bits**

- AR(Address registers)- holds address for memory. **12 bits**
- PC(Program counter)- Address of next instruction. **12 bits**
- INPR(Input Register)-receives 8 bit input from input device. **8 bits**
- OUTR(Output Registers)-hold 8 bit character for output. **8 bits**

Why common bus system is used?

- Basic computer has eight registers, a memory unit and a control unit.
- Path must be provided to transfer information from one register to another and between memory and register.
- The number of wires will be excessive if connections are between the outputs of each register and input of other registers.
- So efficient way is to use common bus system.

Common Bus system



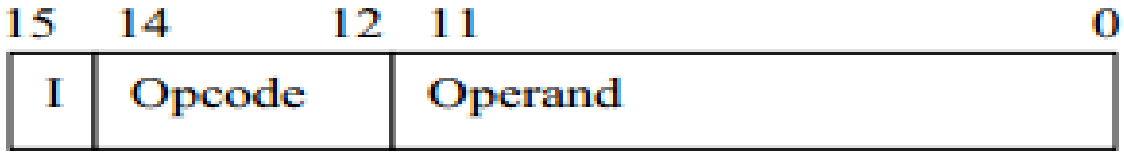
Computer Instructions

- **Memory reference instructions**–instruction that has operand addresses referring to a location in memory
- **Register Reference Instructions**–specifies a operation on AC or a test of the AC
- **Input Output instructions**–used to input data / output data

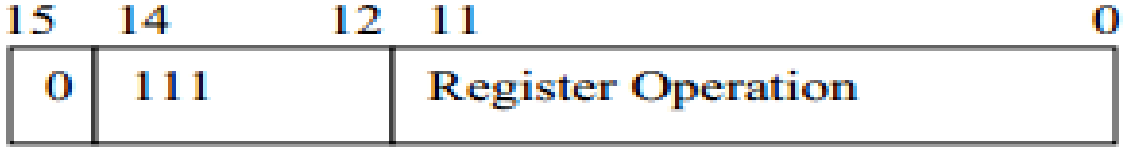
Computer Instruction Format with its types

Instruction Formats of Basic Computer

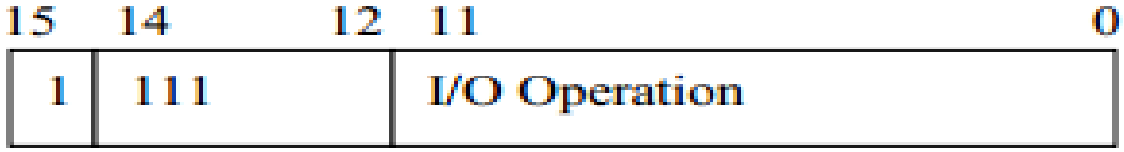
Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)



Input-Output Instructions (OP-code =111, I = 1)



Basic Computer Instructions

TABLE 5-2 Basic Computer Instructions

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

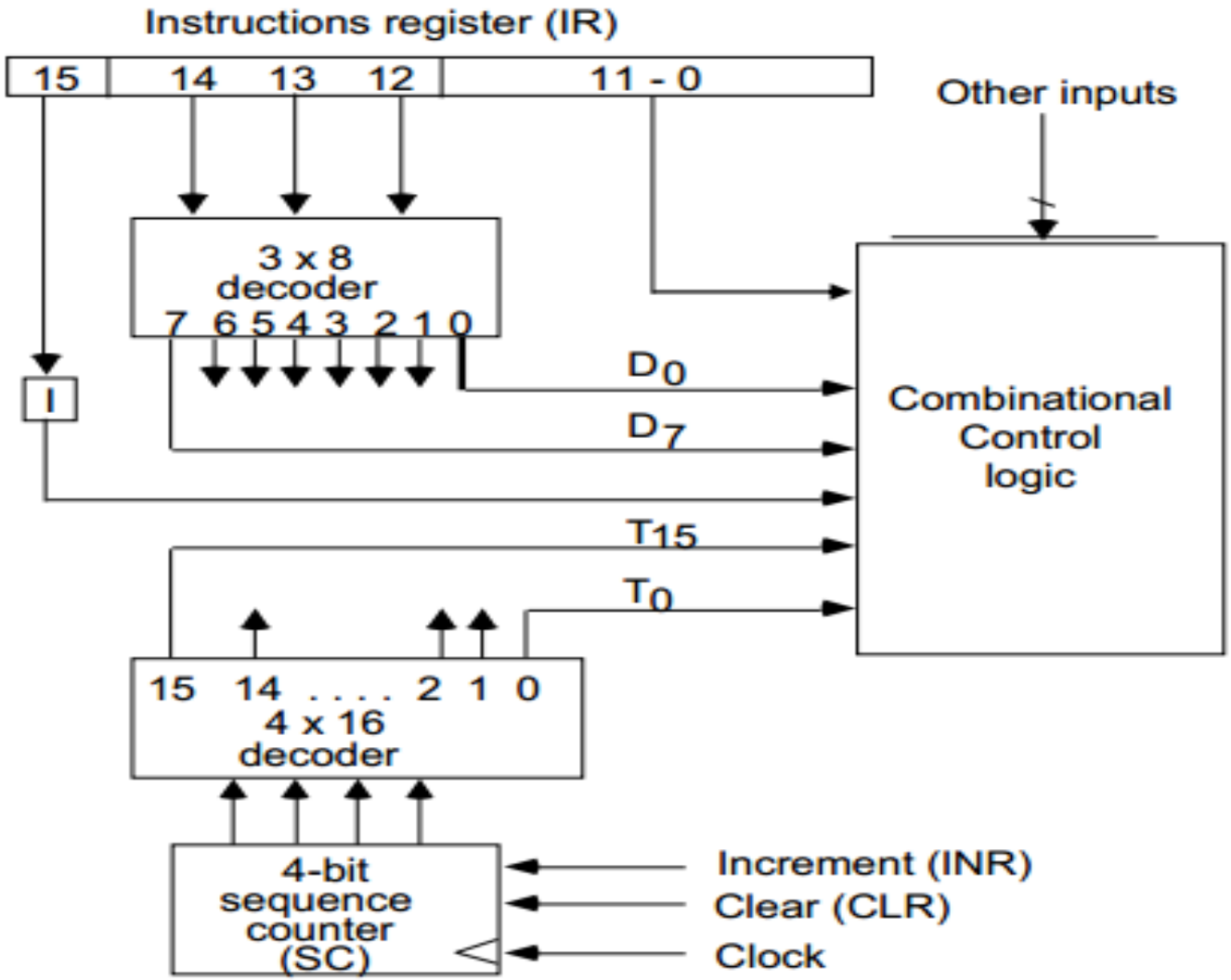
Instruction Set Completeness

- Instruction set is complete if it contains sufficient set of instructions in each of following categories-
  - Arithmetic, logical and shift instruction
  - Instruction for moving information to and from memory and processor registers.
  - Program control instructions together that check status conditions(E.g Branch)
  - Input and Output Instructions

Control Unit

- Control Unit is implemented In 2 ways:-
- ***Hardwired* Control**
  - CU is made up of sequential and combinational circuits to generate the control signals.
  - If logic is changed we need to change the whole circuitry
  - Expensive
  - Fast
- ***Micro programmed* Control**
  - A control memory on the processor contains micro-programs that activate the necessary control signals.
  - If logic is changed we only need to change the micro-program.
  - Cheap
  - Slow

Control Unit of Basic Computer



Instructions cycle

- Program is executed in computer by going through a cycle for each instruction. Each instruction cycle is subdivided into different 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.

Fetch and decode

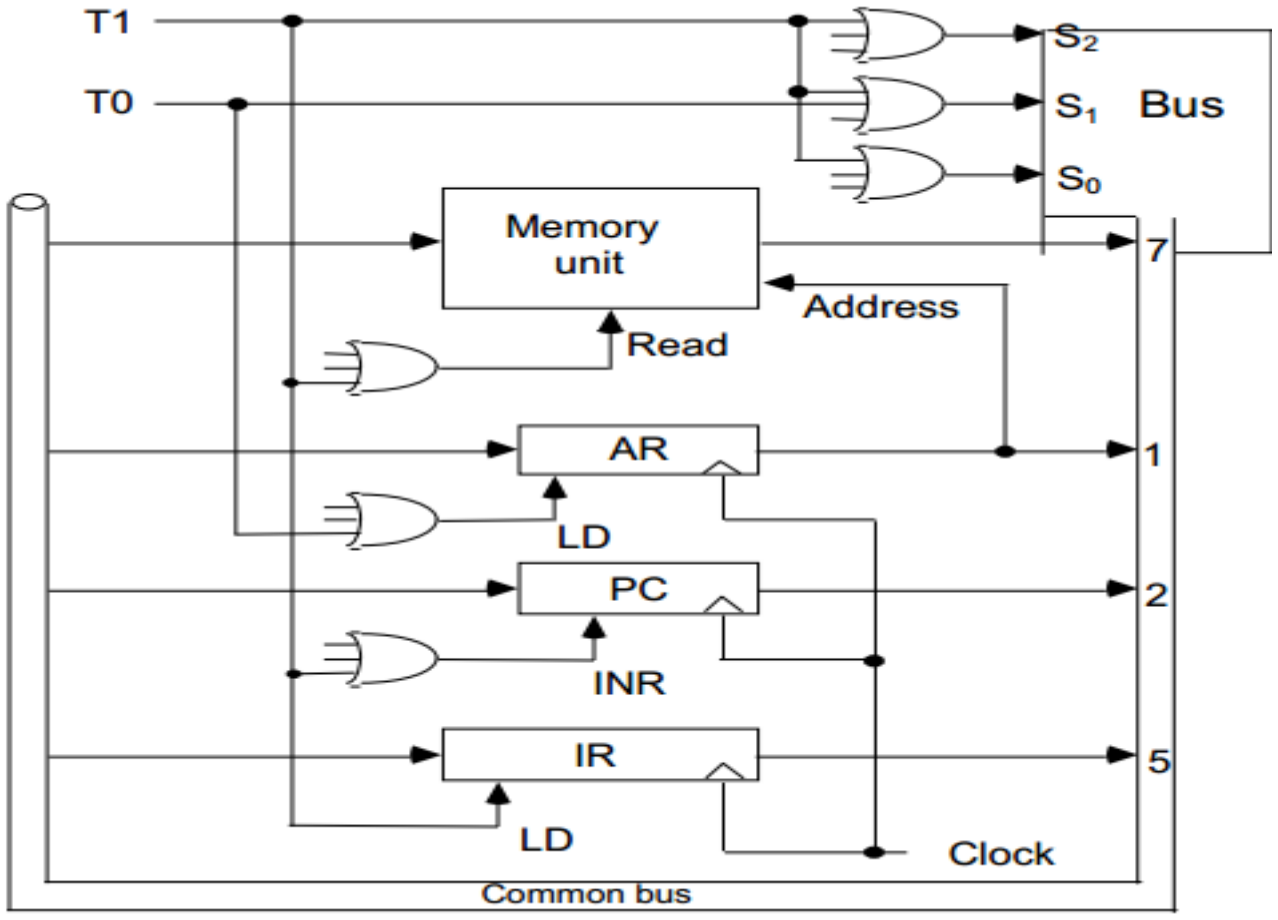
- Initially PC is loaded with address of first instruction .
- Sequence counter SC is cleared to 0, providing T_0.After each clock pulse SC is Incremented and timing sequence go through T_0, T_1, T_2 and so on.

Fetch and Decode

- The micro operations for the fetch and decode phases can be specified by the following register transfer statements:

T0: AR ← PC (S0S1S2=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

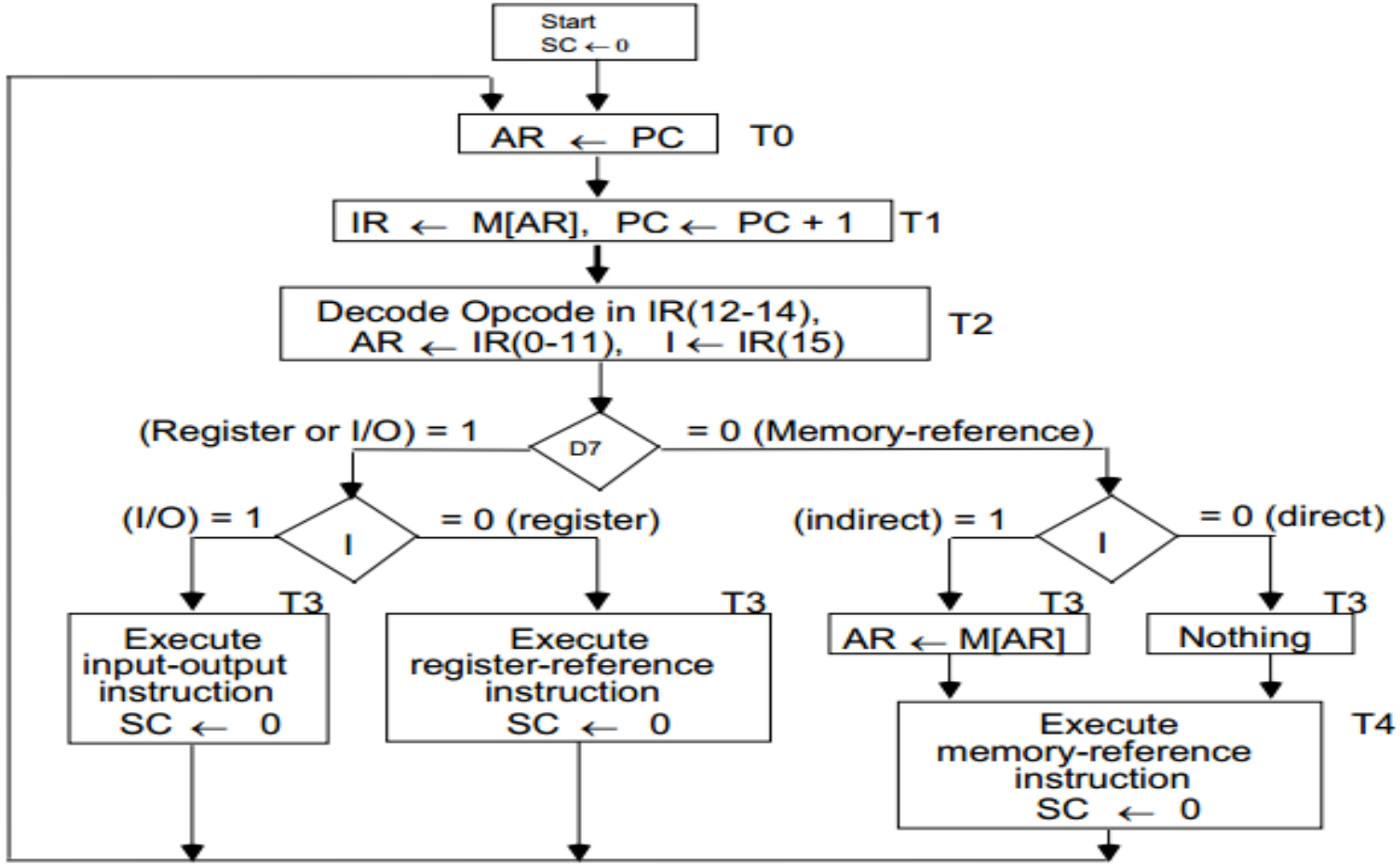
Fetch Phase


T1
T0
S_2
S_1
S_0
Bus
Memory unit
Address
Read
AR
LD
PC
INR
IR
LD
Clock
7
1
2
5
Common bus

Determine the type of instruction

- After decoding the timing signal is T_3
- Control unit determine the type of instruction just read from memory.
- There are 3 type of instruction.
  - D_7 =1, opcode=111
    - Instruction can be register reference or input/output type.
    - I=0 register reference
    - i=1 I/O reference
  - D_7 =0, opcode=000 to 110
    - Instruction is memory referenced.
    - I=0 direct addressing
    - I=1 indirect Addressing

Flowchart of instruction cycle



Register –Reference Instructions

- Register Reference Instructions are identified when
  - D7 = 1, I = 0
  - Register Ref. Instr. is specified in b0 ~ b11 of IR
  - Execution starts with timing signal T3

Register –Reference Instructions

CLA	rB11:	$AC \leftarrow 0, SC \leftarrow 0$
CLE	rB10:	$E \leftarrow 0, SC \leftarrow 0$
CMA	rB9:	$AC \leftarrow AC', SC \leftarrow 0$
CME	rB8:	$E \leftarrow E', SC \leftarrow 0$
CIR	rB7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0), SC \leftarrow 0$
CIL	rB6:	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB5:	$AC \leftarrow AC + 1, SC \leftarrow 0$
SPA	rB4:	$if\ (AC(15) = 0)\ then\ (PC \leftarrow PC+1), SC \leftarrow 0$
SNA	rB3:	$if\ (AC(15) = 1)\ then\ (PC \leftarrow PC+1), SC \leftarrow 0$
SZA	rB2:	$if\ (AC = 0)\ then\ (PC \leftarrow PC+1), SC \leftarrow 0$
SZE	rB1:	$if\ (E = 0)\ then\ (PC \leftarrow PC+1), SC \leftarrow 0$
HLT	rB0:	$S \leftarrow 0, SC \leftarrow 0$ (S is a start-stop flip-flop)

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 complete in a CPU cycle
- The execution of memory reference instruction starts with T4

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 $PC \leftarrow PC+1$

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 effective address of the instruction is in AR and was placed there during timing signal T_2 when $I = 0$, or during timing signal T_3 when $I = 1$
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

AND to AC

D_0T_4:	$DR \leftarrow M[AR]$	Read operand
D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$	AND with AC

ADD to AC

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

MEMORY REFERENCE INSTRUCTIONS

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC

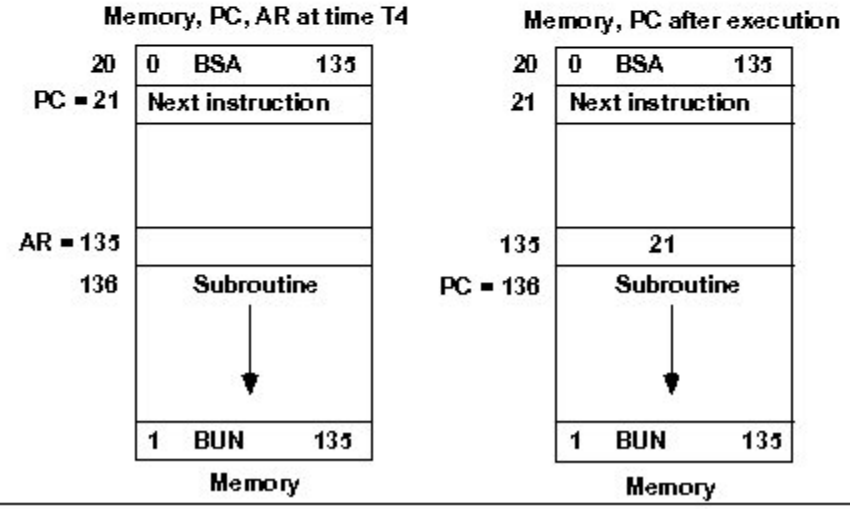
D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

M[AR] ← PC, PC ← AR + 1



MEMORY REFERENCE INSTRUCTIONS

BSA:

D_5T_4: M[AR] ← PC, AR ← AR + 1
D_5T_5: PC ← AR, SC ← 0

ISZ: Increment and Skip-if-Zero

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

Microprogrammed Control

Unit 4

Design of Accumulator logic

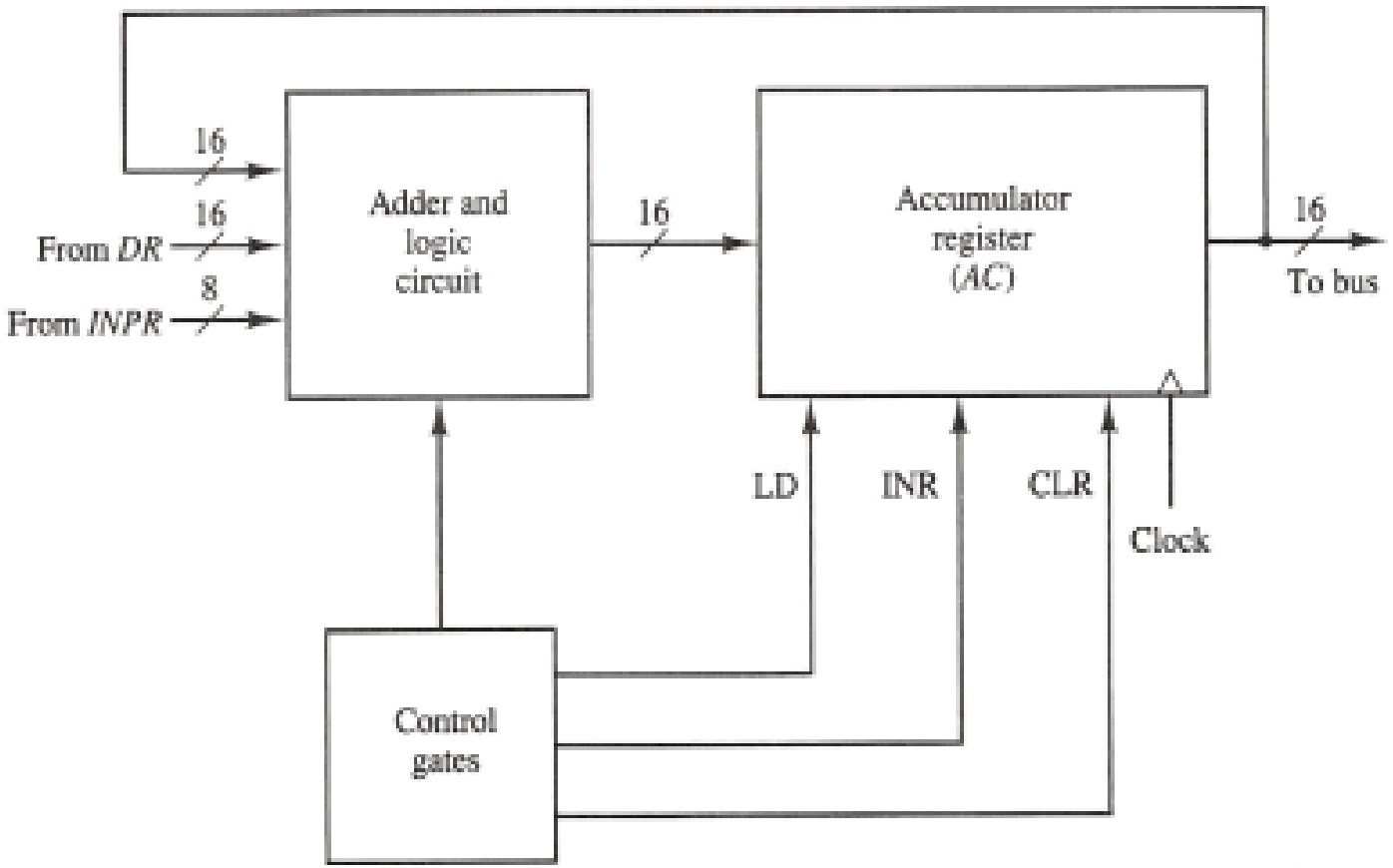


Fig: circuits associated with AC

Change AC

D_0T_5:	$AC \leftarrow AC \wedge DR$		AND with *DR*
D_1T_5:	$AC \leftarrow AC + DR$		Add with *DR*
D_2T_5:	$AC \leftarrow DR$		Transfer from *DR*
pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$		Transfer from *INPR*
rB_9:	$AC \leftarrow \overline{AC}$		Complement
rB_7:	$AC \leftarrow \text{shr } AC,$	$AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shl } AC,$	$AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$		Clear
rB_5:	$AC \leftarrow AC + 1$		Increment

Gate Structure For Controlling AC

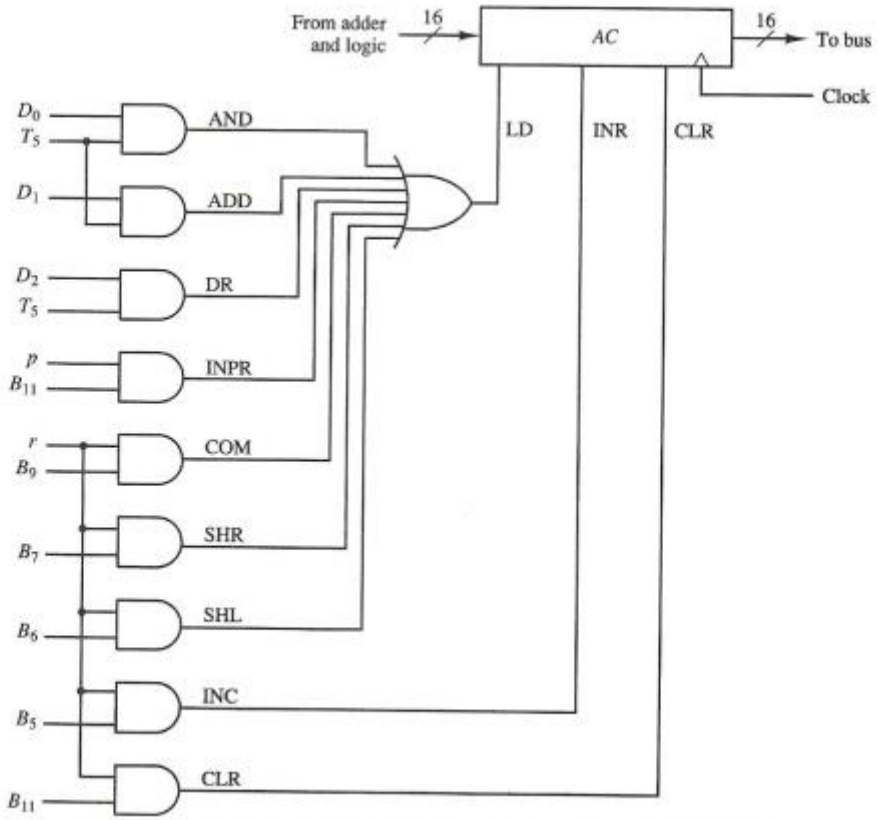


Fig: Gate structure for controlling LD, INR and CLR of AC

Control unit

- Initiate sequence of microoperations
- Number of microoperations are finite

Control Unit

- Control Unit is implemented In 2 ways:-
- ***Hardwired* Control**

  – fixed instructions, fixed logic block, encoder, decoder.

  -CU is made up of sequential and combinational circuits to generate the control signals.

  – If logic is changed we need to change the whole circuitry

  – Expensive

  – high speed operations.

  -complex

  -no flexibility of adding instructions

  -Intel 8085,motorola 6802,RISC.

- ***Micro programmed*** **Control**
  - A control memory on the processor contains micro-programs that activate the necessary control signals.
  - If logic is changed we only need to change the micro-program.
  - Cheap
  - Slow

-Intel 8080,Motorola 68000, CISC.

Hardwired Control

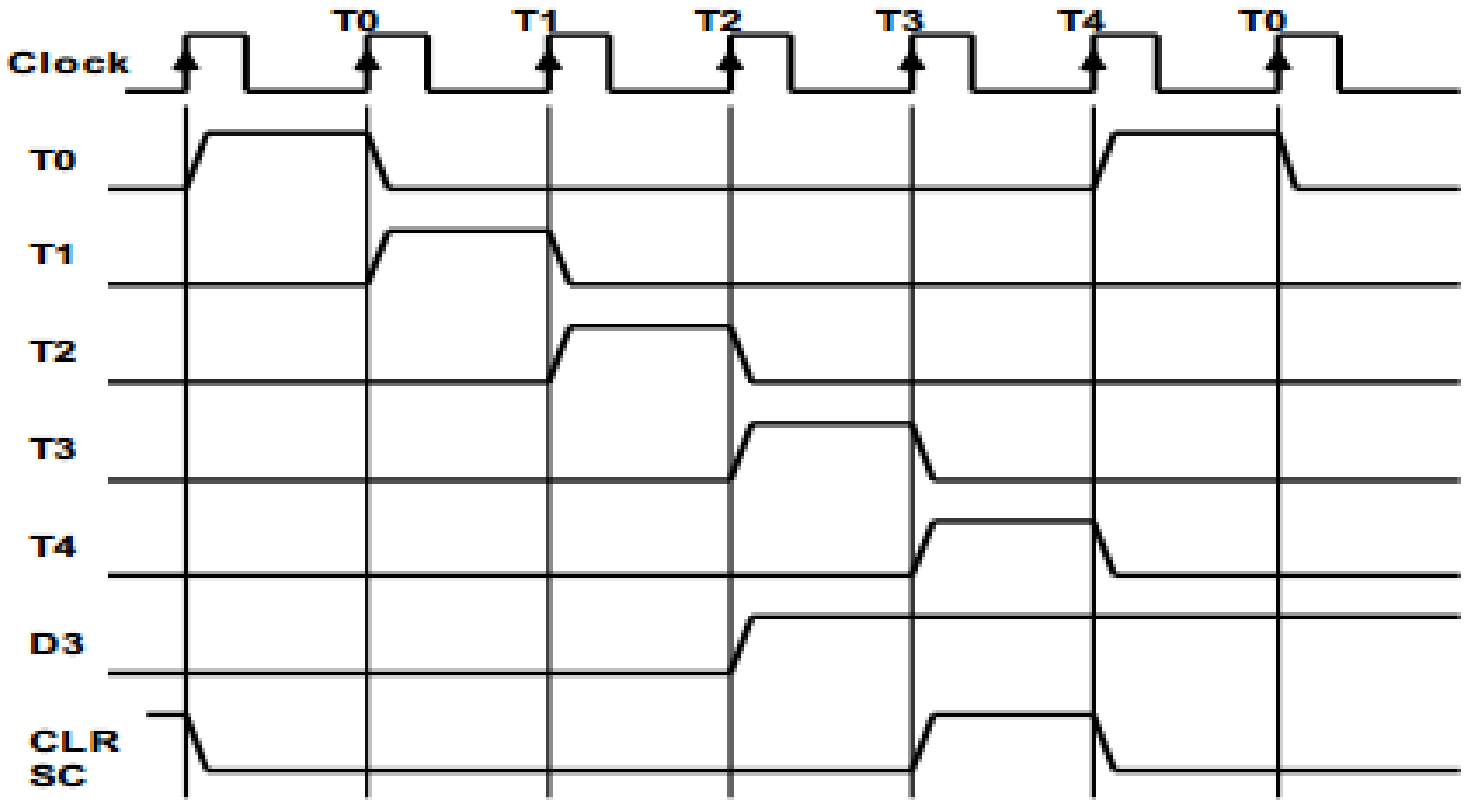
Introduction

- Control logic is implemented with gates, flip flops, decoder and other digital circuits.
- If logic is changed we need to change the whole circuitry
- Expensive
- Fast

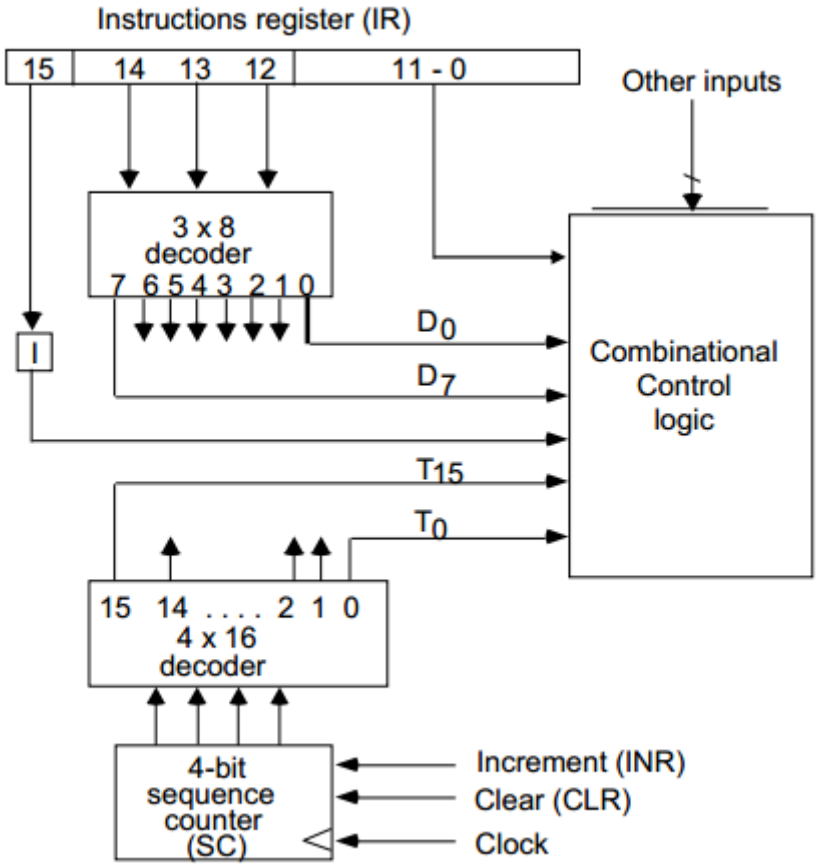
Timing And Control

- Generated by 4-bit sequence counter and 4x16 decoder.
  - The SC can be incremented or cleared.
  - Example: T0, T1, T2, T3, T4, T0, T1 . . .

Timing Signal



Control Unit Of Basic Computer



Control Memory

Control Word

- Control function specifies a microoperation is a binary variable.
- When it is in one binary state microoperation is executed.(other doesn't change)
- Active state can be 1 or 0 depending on application.
- Control variables at any given time can be represented by string of 0's and 1's called control word.

Control Memory

- Contains 2 memory
  - Main Memory
    I. Used to store program
    II. Content can be altered
  - Control Memory
    I. Located in control unit
    II. Contains fixed microprogrammed that can't be altered by occasional user.
    III. Consists of microinstructions
    IV. Microinstructions specify various register microoperations.

Microinstruction

- Each word in control memory is ***microinstruction.***
- Microinstruction specifies one or more microoperations.
- Sequence of microinstructions are ***Microprogram.***
- Advance technology-Dynamic programming
  - Microprogram loaded from auxiliary device.
  - Writable control memory.
  - But control memory are mostly used for reading.

Micro programmed Control

- **Control word**
  - It is a string of control variables (0's and 1's) occupying a word in control memory.
- **Microprogram**
  - Program stored in control memory that generates all the control signals required to execute the instruction set correctly
  - Consists of microinstructions

- **Microinstruction**
  - Contains a control word and a sequencing word
  - *Control Word* – contains one or more microoperations
  - *Sequencing Word* – Contains information needed to decide the next microinstruction address .

Micro Programmed Control Organization

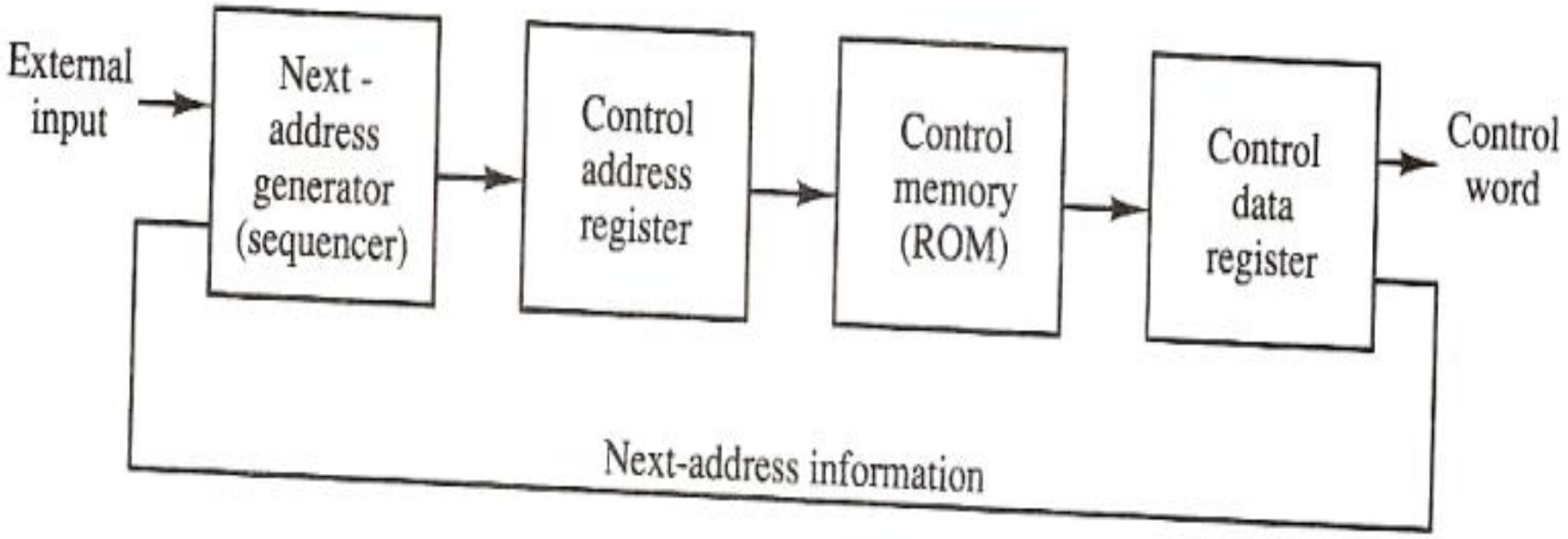


Fig: Microprogrammed control organization

Microprogrammed control Organization

- **Control memory** is ROM, within which all control information is permanently stored.
- **Control memory Address Register** specifies the address of microinstruction.
- **Control Data Register** holds the microinstruction read from memory.

Microprogrammed control Organization

- Microinstructions contains control word that specifies **one or more microoperations.**
- After execution **the location of next microinstruction must be determine.**
  - *Address can be **in sequence or out of sequence***.
  - So, some bits of present microinstruction are used to determine the next address of microinstruction.
- Next address is calculated in next address generator.
  - Also know as sequencer.(Determine next address in sequence)

Address Sequencing (Introduction)

- Microinstructions are stored in control memory in groups.
  - **Called routine.**
  - Every **computer instruction has its own microprogram routine in control memory**, which generate sets of microoperations to execute that instruction.
  - Hardware that control address sequencing of control memory must be **capable of sequencing microinstructions within routine or may able to branch from one routine to other.**

Steps for execution of single computer instruction

- Control address register is loaded with initial address when computer is on- may be fetch routine.
- Fetch routine is sequenced by incrementing CAR.
- At END of routine instruction is in IR.

Calculating Effective Address of operand.

- Can be reached through branch microinstruction.
- At END operand address is available in memory Address Register.

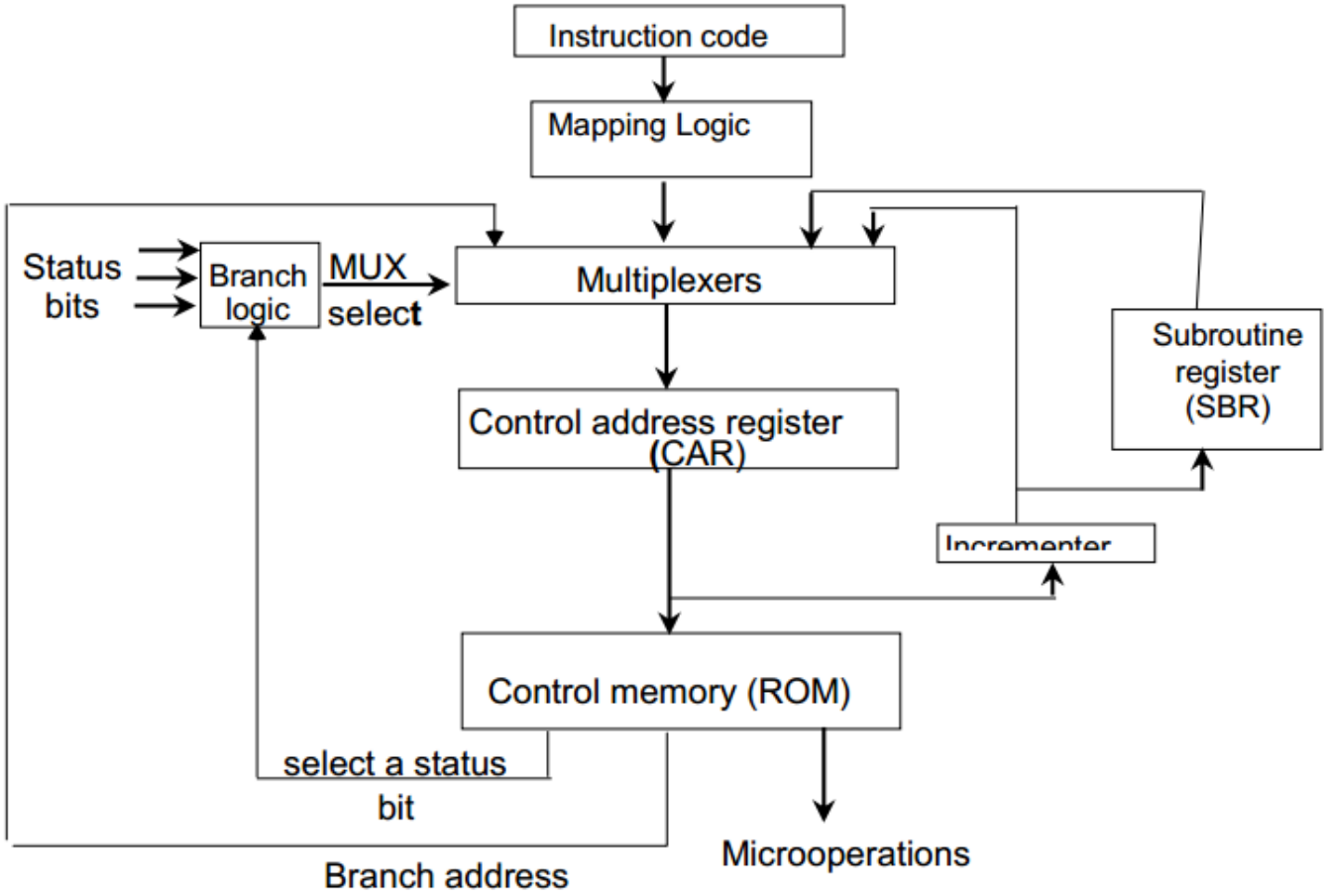
Execute Instruction

- Opcode determine which routine should be run for execution of particular operation.
- Each instruction has its own microprogram routine.
- Mapping process
  - Process of transformation from opcode bit to an address in control memory where routine is located.

Address sequencing capabilities required in control memory

- Incrementing of Control Address register.
- Unconditional branch or conditional branch depending on status bit.
- A mapping process from bit of instruction to control memory address.
- A facility of subroutine call and return.

Block Diagram of Address Sequencer



Conditional Branching

- If Condition is true, set the appropriate field of status register to 1. Conditions are tested for
  - O (overflow),
  - N (negative),
  - Z (zero) and C(carry)
  - Then test the value of that field if the value is 1 take **branch address from the next address field** of the current microinstruction Otherwise **simple increment the address.**

Unconditional branching

- Fix the value of one status bit at the input of the multiplexer to 1. So that always branching is done.

Subroutine

- **Subroutine are program that are used by other routine** to accomplished a particular task.
- Subroutine can be called at any point in main program.
- Some **subroutine are frequently used** in many routine.
- E.g calculation of EA is common to all memory reference instruction.

- When subroutine is called the **incremented address is save in Subroutine register.**
- Register are arranged as stack- used in LIFO manner

Microprogram

- After Microprogrammed Control unit is established
  - Designer generate microcode for control memory(Microprogramming)

Computer Configuration

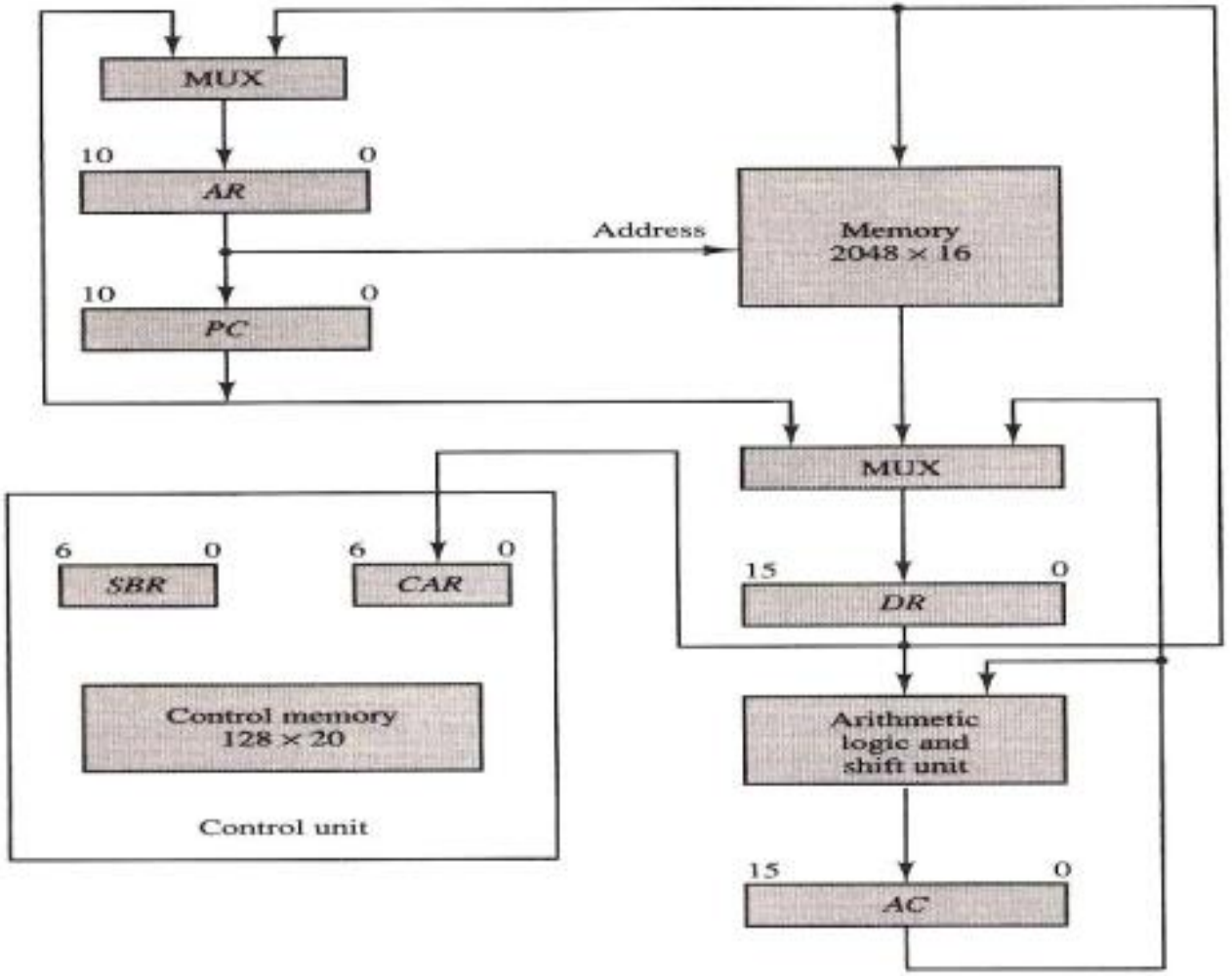


Fig: Computer hardware configuration

Microinstructions

F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Description of CD

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Description of BR

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR (2-5) ← DR (11-14), CAR (0, 1, 6) ← 0

Microinstruction fields (F1, F2, F3)

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{-}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Symbolic Microinstruction

- Symbols like ADD,CLRAC,READ,COM etc are used to specify microinstructions.
- Assembler is used to translate symbolic microinstruction to its binary equivalent.
- Contains five fields:-
  - **Label**:-This field may be empty or it may specify a symbolic address. A label is terminated with colon(:).
  - **Microoperations field**:- contains one, two or three symbols separated by commas.
  - **CD**:- one of U, I,S,Z.
  - **BR**- can contains 4 symbols. JMP,CALL,RET,MAP.

- **AD**–With symbolic address
  - With Symbol NEXT to designate the next address in sequence.
  - When BR field contains RET or MAP Symbol, the AD field is left empty and is converted to 7 zeroes by assemblers.
- **ORG**- origin address

FETCH ROUTINE Example

FETCH Routine: During FETCH Read an instruction from memory and decode the instruction and update PC

Sequence of microoperations in the *fetch cycle*:

```
AR ← PC

DR ← M[AR], PC ← PC + 1

AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

Symbolic microprogram for the fetch cycle:

```
            ORG 64
FETCH:      PCTAR          U     JMP     NEXT
            READ, INCPC    U     JMP     NEXT
            DRTAR          U     MAP
```

Binary Microprogram

- Program not Stored in memory in Symbolic form.
- Translated by assembler.

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Design of Control Unit

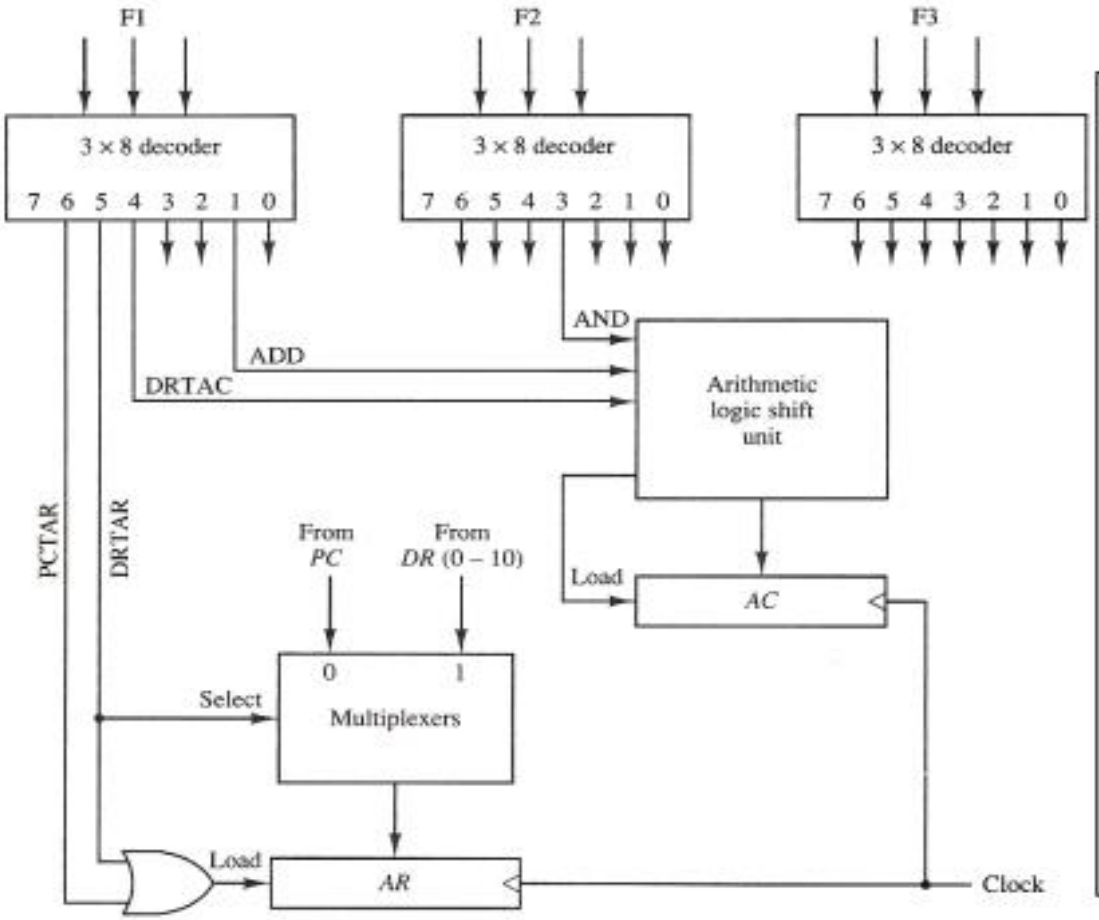


Fig: Decoding of microoperation fields

E.g. when F1=101 (binary 5), next clock pulse transition transfers the content of DR(0-10) to AR (DRTAR). Similarly when F1=110(6), there is a transfer from PC to AR (PCTAR). Outputs 5 & 6 of decoder F1 are connected to the load inputs of AR so that when either is active information from multiplexers is transferred to AR.

Arithmetic logic shift unit instead of using gates to generate control signals, is provided inputs with outputs of decoders (AND, ADD and ARTAC).

Microprogram Sequencer

- Basic component of microprogrammed are
  - Control memory and next address generator
- Address selection part is microprogram sequencer.
- It load CAR with address of microinstruction to fetch and execute next microprogram.
- Control unit doesn't contain one register but stack of SBR.

Microprogram Sequencer

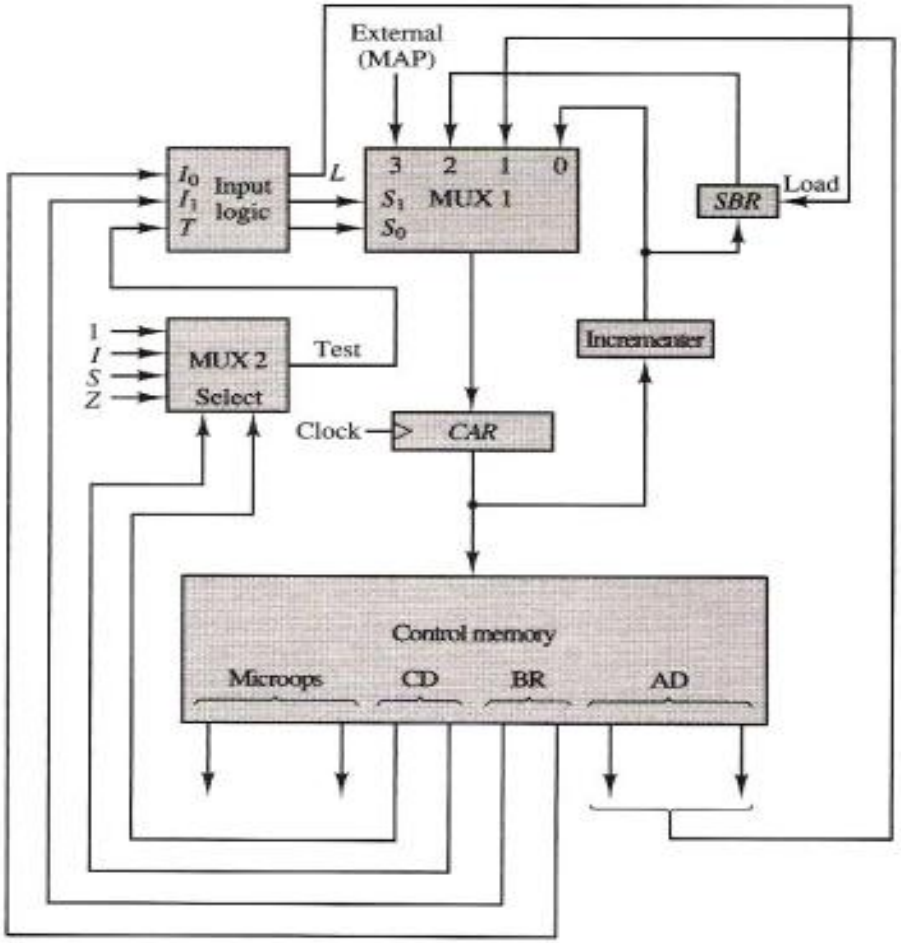


Fig: Microprogram sequencer for a control memory

-MUX1 selects an address from one of four sources of and routes it into CAR.

-MUX2 tests the value of selected status bit and result is applied to input logic circuit.

-Output of CAR provides address for the control memory

-Input logic circuit has 3 inputs I_0, I_1 and T and 3 outputs S_0, S_1 and L. variables S_0 and S_1 select one of the source addresses for CAR. L enables load input of SBR.

-e.g. when S_1S_0=10, MUX input number 2 is selected and establishes a transfer path from SBR to CAR.

BR Field		Input			MUX 1		Load *SBR*
		I_1	I_0	T	S_1	S_0	L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	×	1	0	0
1	1	1	1	×	1	1	0

Fig: Input Logic Truth for Microprogram Sequencer

Central Processing Unit

Unit 5

CPU

- Process Bulk Data
- Three parts
  - CU-Supervises the transfer of information among registers and instruct the ALU to perform instruction.
  - ALU-Perform Arithmetical and logical operation.
  - Register set- Stores Intermediate Data

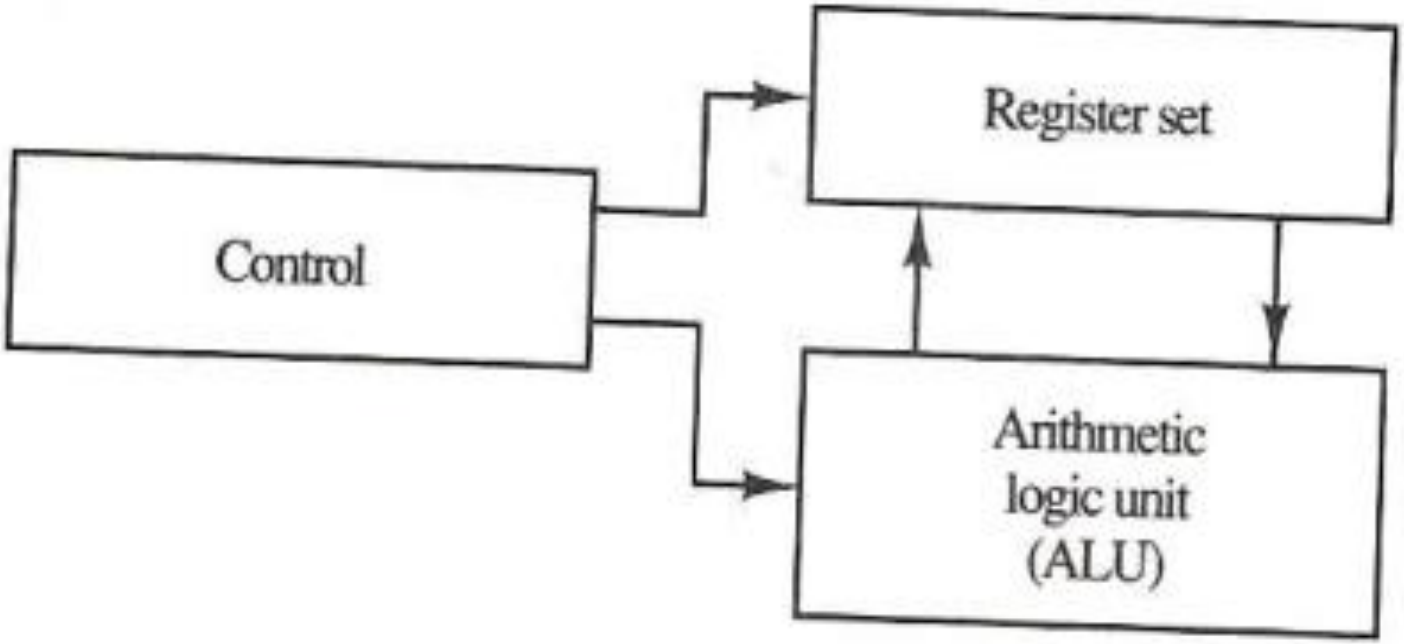
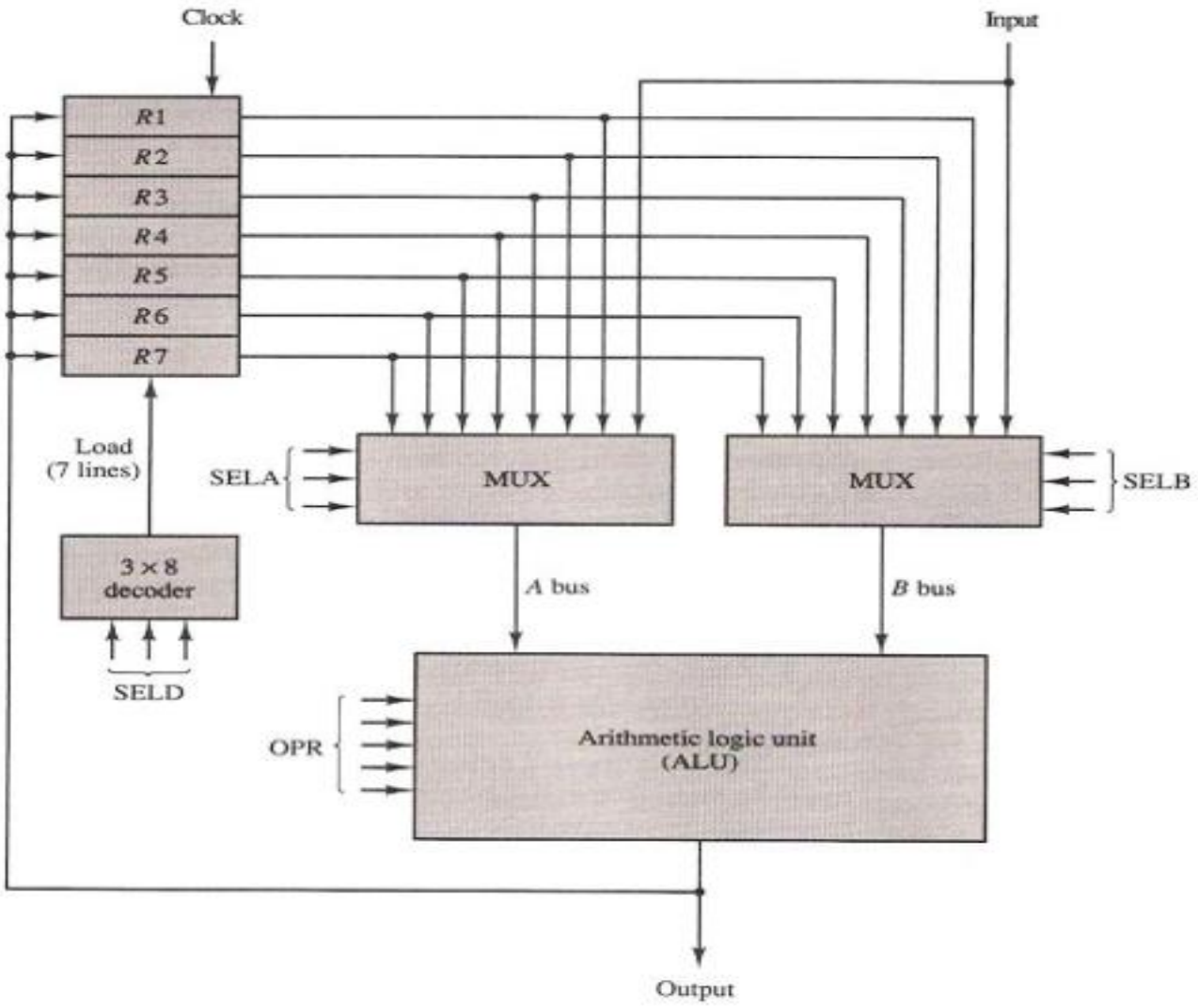


Fig: Major components of CPU

General Register Organization

- Memory are needed
  - To stores intermediate data
  - Partial product of multiplication
  - Temporary results and many more
- Accessing memory is time consuming operations
- So registers are used and are at CPU
- CPU contains large set of registers connected through common bus.

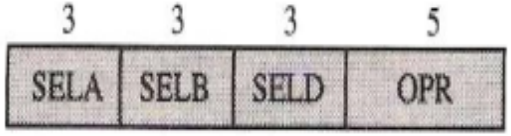

Clock
Input
R1
R2
R3
R4
R5
R6
R7
Load
(7 lines)
SELA
MUX
MUX
SELB
3 × 8
decoder
A bus
B bus
SELD
OPR
Arithmetic logic unit
(ALU)
Output

Example: R1 ← R2 + R3

- MUX selector (SELA): BUS A ← R2
- MUX selector (SELB): BUS B ← R3
- ALU operation selector (OPR): ALU to ADD
- Decoder destination selector (SELD): R1 ← Out Bus

Control word

Combination of all selection bits of a processing unit is called control word. Control Word for above CPU is as below:



The 14 bit control word when applied to the selection inputs specify a particular microoperation.

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R1*	*R1*	*R1*
010	*R2*	*R2*	*R2*
011	*R3*	*R3*	*R3*
100	*R4*	*R4*	*R4*
101	*R5*	*R5*	*R5*
110	*R6*	*R6*	*R6*
111	*R7*	*R7*	*R7*

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

Stack Organization

- CPU included Storage devices that are implemented in LIFO manner i.e Stack
- 2 operation of Stack are PUSH(Insert) and POP(Deletion).
- Register that store Stack Address is called Stack Pointer, it points top of Stack.
- In computer the PUSH And POP operation are simulated by just incrementing and decrementing Stack pointer.

Register Stack

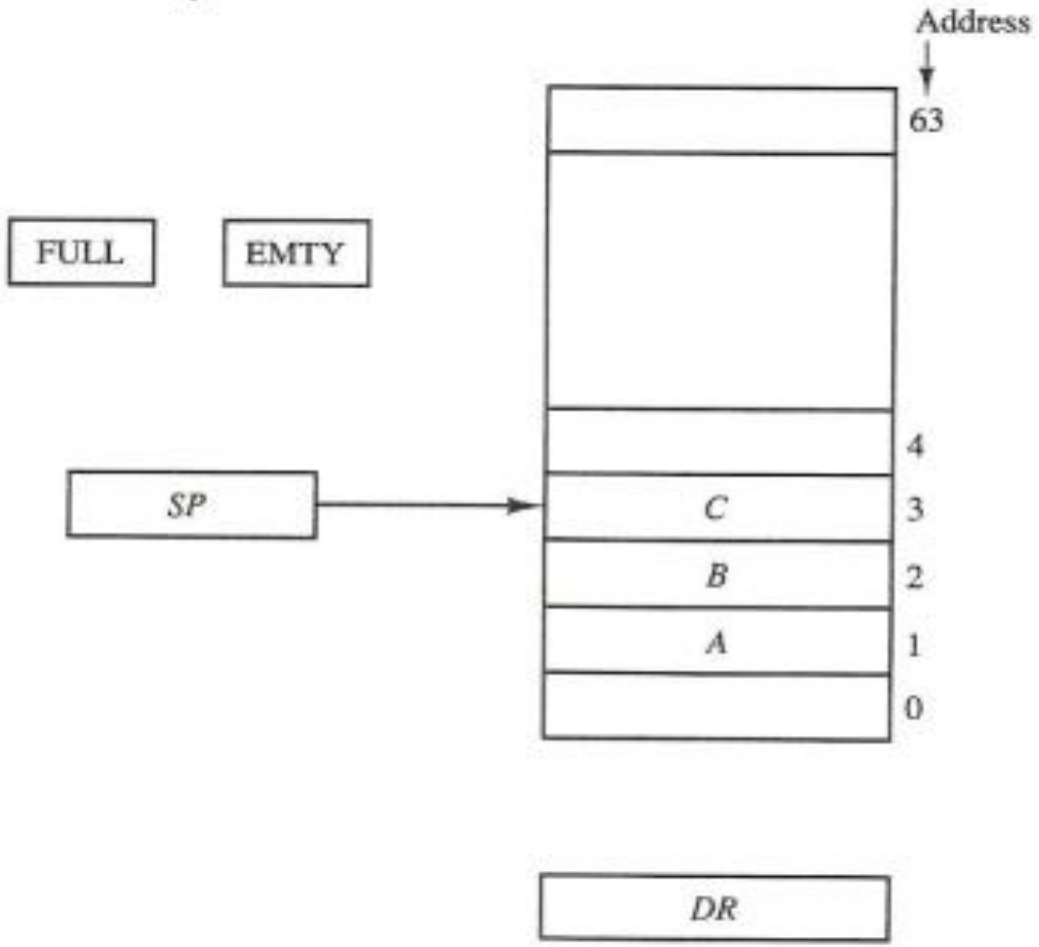


Fig: Block diagram of a 64-word stack

/* Initially, SP = 0, EMPTY = 1(true), FULL = 0(false) */

Push operation

$SP \leftarrow SP + 1$

$M[SP] \leftarrow DR$

If $(SP = 0)$ then $(FULL \leftarrow 1)$

$EMPTY \leftarrow 0$

Pop operation

$DR \leftarrow M[SP]$

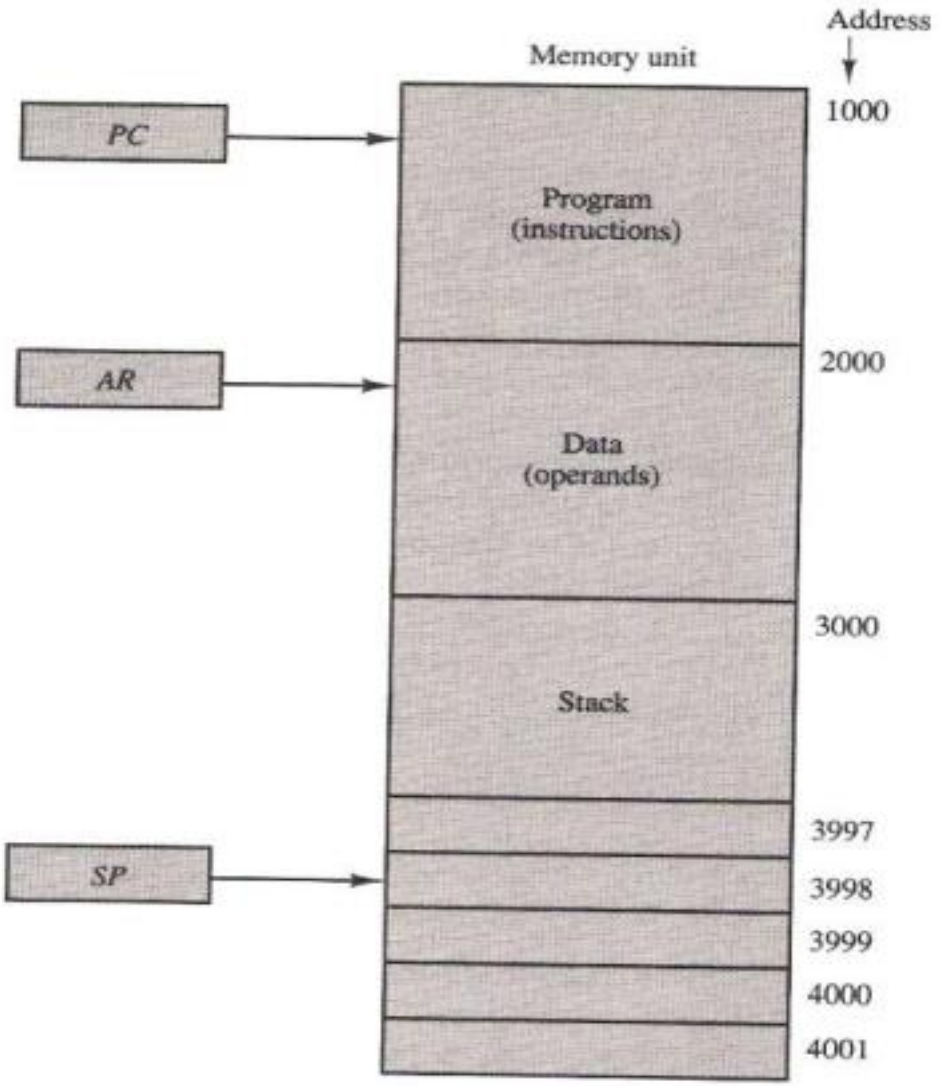
$SP \leftarrow SP - 1$

If $(SP = 0)$ then $(EMPTY \leftarrow 1)$

$FULL \leftarrow 0$

Memory Stack

- A Stack is standalone unit
- Or can be implemented in a Random Access Memory Attached to a CPU
- A portion of memory is assigned to Stack operation using processor register as stack pointer


Address
Memory unit
PC
AR
SP
Program
(instructions)
Data
(operands)
Stack
1000
2000
3000
3997
3998
3999
4000
4001

PC: used during fetch phase to read an instruction.
AR: used during execute phase to read an operand.
SP: used to push or pop items into or from the stack.

Here, initial value of SP is 4001 and stack grows with *decreasing addresses*. First item is stored at 4000, second at 3999 and last address that can be used is 3000. No provisions are available for stack limit checks.

PUSH:

$SP \leftarrow SP - 1$

$M[SP] \leftarrow DR$

POP:

$DR \leftarrow M[SP]$

$SP \leftarrow SP + 1$

- Stack Limits
  - For Stack overflow and stack underflow
  - Stack limit can be checked by using two processor register
  - One to hold the upper limit (3000)
  - One to hold the lower limit (4001)
  - After push, SP is compared with upper limit and after POP it is compared with lower limit.

Instruction Format

Computer instruction have specific format and usually contains three different fields.

1. Operation field that specifies the operation to be performed.

2. An address field that designates a memory address or a processor register.

3. A mode field that specifies the way the operand or the EA is determined.

Processor Organization

1. **Single register (Accumulator) organization**
   - Basic Computer is a good example
   - Accumulator is the only general purpose register
   - Uses implied accumulator register for all operations

Example:

```
ADD X        // AC ← AC + M[X]
LDA Y        // AC ← M[Y]
```

2. **General register organization**
   - Used by most modern processors
   - Any of the registers can be used as the source or destination for computer operations.

Example:

```
ADD R1, R2, R3    // R1 ← R2 + R3
ADD R1, R2        // R1 ← R1 + R2
MOV R1, R2        // R1 ← R2
ADD R1, X         // R1 ← R1 + M[X]
```

3. **Stack organization**
   - All operations are done with the stack
   - For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack.

Example:

```
PUSH X       // TOS ← M[X]
ADD          // TOS = TOP(S) + TOP(S)
```

- **Three-Address Instructions**

  Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

  Assembly language program to evaluate X = (A + B) * (C + D):

```
ADD   R1, A, B      // R1 ← M [A] + M [B]
ADD   R2, C, D      // R2 ← M[C] + M [D]
MUL   X, R1, R2     // M[X] ← R1 * R2
```

  - Results in short programs
  - Instruction becomes long (many bits)

- **Two-Address Instructions**
  These instructions are most common in commercial computers.
  Program to evaluate X = (A + B) * (C + D):

```
MOV   R1, A      // R1 ← M [A]
ADD   R1, B      // R1 ← R1 + M [A]
MOV   R2, C      // R2 ← M[C]
ADD   R2, D      // R2 ← R2 + M [D]
MUL   R1, R2     // R1 ← R1 * R2
MOV   X, R1      // M[X] ← R1
```

- Tries to minimize the size of instruction
- Size of program is relatively larger.

- **One-Address Instructions**

  One-address instruction uses an implied accumulator (AC) register for all data manipulation. All operations are done between AC and memory operand.

  Program to evaluate X = (A + B) * (C + D):

```
LOAD   A        // AC ← M [A]
ADD    B        // AC ← AC + M [B]
STORE  T        // M [T] ← AC
LOAD   C        // AC ← M[C]
ADD    D        // AC ← AC + M [D]
MUL    T        // AC ← AC * M [T]
STORE  X        // M[X] ← AC
```

  - Memory access is only limited to load and store
  - Large program size

- **Zero-Address Instructions**
  A stack-organized computer uses this type of instructions.

  Program to evaluate X = (A + B) * (C + D):

```
PUSH  A      // TOS ← A
PUSH  B      // TOS ← B
ADD          // TOS ← (A + B)
PUSH  C      // TOS ← C
PUSH  D      // TOS ← D
ADD          // TOS ← (C + D)
MUL          // TOS ← (C + D) * (A + B)
POP X        // M[X] ← TOS
```

The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

Addressing modes

- **Implied Mode**

  Address of the operands is specified implicitly in the definition of the instruction.

  - No need to specify address in the instruction
  - Examples from Basic Computer CLA, CME, INP

  ADD X;

  PUSH Y;

- **Immediate Mode**

  Instead of specifying the address of the operand, operand itself is specified in the instruction.

  - No need to specify address in the instruction
  - However, operand itself needs to be specified
  - Sometimes, require more bits than the address
  - Fast to acquire an operand

- **Immediate Mode**

  Instead of specifying the address of the operand, operand itself is specified in the instruction.

  - No need to specify address in the instruction
  - However, operand itself needs to be specified
  - Sometimes, require more bits than the address
  - Fast to acquire an operand

- **Register Mode**

  Address specified in the instruction is the address of a register

  - Designated operand need to be in a register
  - Shorter address than the memory address
  - A k-bit address field can specify one of 2^k registers.
  - Faster to acquire an operand than the memory addressing

- **Register Indirect Mode**
  - Instruction specifies a register in CPU whose content give the address of the operand in memory.
  - Uses fewer bit in address field of instruction because register address uses fewer bit than memory address.

- **Autoincrement or Autodecrement Mode**

  It is similar to register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.

- **Direct Addressing Mode**

  Instruction specifies the memory address which can be used directly to access the memory

  - Faster than the other memory addressing modes
  - Too many bits are needed to specify the address for a large physical memory Space
  - EA= IR(address)

- **Indirect Addressing Mode**
  - The address field of an instruction specifies the address of a memory location that contains the address of the operand
  - When the abbreviated address is used large physical memory can be addressed with a relatively small number of bits
  - Slow to acquire an operand because of an additional memory access
  - EA= M[IR (address)]

- **Relative Addressing Modes**

  The Address field of an instruction specifies the part of the address which can be used along with a designated register (e.g. PC) to calculate the address of the operand.

  - Address field of the instruction is short
  - Large physical memory can be accessed with a small number of address bits

3 different Relative Addressing Modes:

* ***PC Relative Addressing Mode***:
  - EA = PC + IR(address)
* ***Indexed Addressing Mode***
  - EA = IX + IR(address) { IX is index register }
* ***Base Register Addressing Mode***
  - EA = BAR + IR(address)

Indexed Addressing mode

- Content of index register is added to the address part of the instruction to obtain effective address.{EA=IR(Address)+IX)}
- Index register contain index value.
- Address field define beginning address of data array.
- The distance between the beginning of address and the address of the operand is index value stored in the index register.
- The index register can be increment to access consecutive data.

Base register indexing

- Base register contains base address.
- Base register is added to address part of instruction to find effective address.
- Used in program relocation.
- In program relocation- a base register the displacement value of instruction is not changed, only base register value is updated.

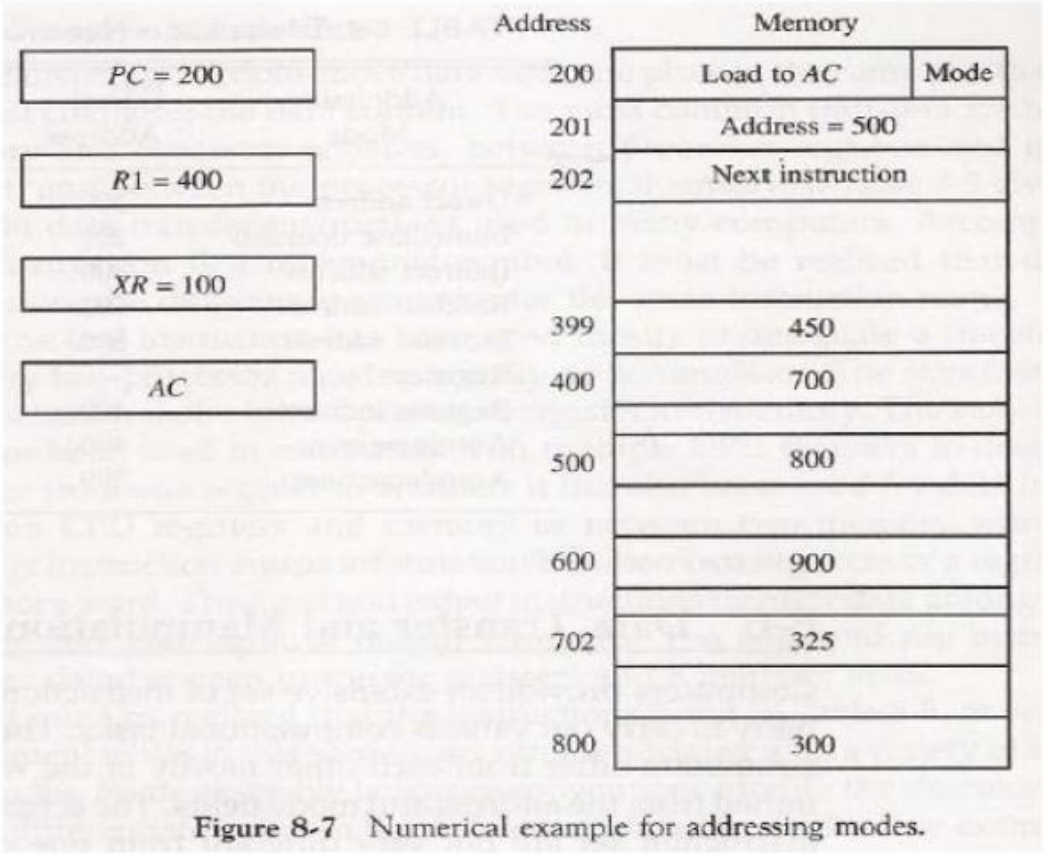


Figure 8-7 Numerical example for addressing modes.

Basic Instructions Set

- Computer provide **various instruction set** to carry out various computational tasks.
- **Instruction set differ from computer to computer**---with operand determination by using mode bits and address.
- Every computers instruction sets are similar instead they may have **different binary coding or symbolic representation.**
- There are sets of computer instructions which are as follows:

Basic Instruction sets

- **Data transfer instructions**
  - Use for transferring data from one location to other without changing the content.
- **Data manipulation Instructions**
  - This instructions implements arithmetic, logic, and shift operations.
- **Program control instructions**
  - This instruction provide decision making capabilities and change path taken by the program when executed in the computer.

Data Transfer Instructions

- between memory and processor registers
- between processor registers and I/O
- between processor register themselves

Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Load: denotes transfer from memory to registers (usually AC)
Store: denotes transfer from a processor registers into memory
Move: denotes transfer between registers, between memory words or memory & registers.
Exchange: swaps information between two registers or register and a memory word.
Input & Output: transfer data among registers and I/O terminals.
Push & Pop: transfer data among registers and memory stack.

Data Manipulation

- Data manipulation instructions provide computational capabilities for the computer. These are divided into 3 parts:
  1. Arithmetic instructions
  2. Logical and bit manipulation instructions
  3. Shift instructions

Arithmetic Operations

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

- Increment (decrement) instr. adds 1 to (subtracts 1 from) the register or memory word value.
- Add, subtract, multiply and divide instructions may operate on different data types (fixed-point or floating-point, binary or decimal).

Logical and Bit Manipulation

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

- Clear instr. causes specified operand to be replaced by 0's.
- Complement instr. produces the 1's complement.
- AND, OR and XOR instructions produce the corresponding logical operations on individual bits of the operands.

Shift Microoperations

- Rotate-type operations

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- Table lists 4 types of shift instructions.
- Logical shift inserts 0 at the end position
- Arithmetic shift left inserts 0 at the end (identical to logical left shift) and arithmetic shift right leave the sign bit unchanged (should preserve the sign).
- Rotate instructions produce a circular shift.
- Rotate left through carry instruction transfers carry bit to right and so is for rotate shift right.

Program Control Instruction

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

- Branch (usually one address instruction) and jump instructions can be changed interchangeably.
- Skip is zero address instruction and may be conditional & unconditional.
- Call and return instructions are used in conjunction with subroutine calls.

Program control

- Program may execute in sequence
  - PC value is incremented every time and after one instruction is execute next in sequence instruction is fetched , decode and execute.
  - After again next instruction is fetched and so on.
- Program may execute out of sequence
  - A program control type of instruction, when executed may change the value of program counter and cause the flow of control to be altered.
  - This cause break in sequence of execution.
  - Program control is Important feature as it provides control over the flow of program execution and a capability for branching to different segments.
  -

Program Control Instruction

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

- Branch (usually one address instruction) and jump instructions can be changed interchangeably.
- Skip is zero address instruction and may be conditional & unconditional.
- Call and return instructions are used in conjunction with subroutine calls.

- **Branch and jump** instructions are used for conditional and unconditional.
- An **unconditional branch** causes instruction to branch to address specify in instructions without any condition.
- The **conditional branch** instruction specify a condition to be true or false.
  - If condition is met , the PC is loaded with address specify in instruction or branch address.
  - If condition fails the next instruction is executed.

- The **skip** instruction doesn't need an address field therefore it is zero address instruction
- A **conditional skip** instruction will skip the next instruction if condition is met.
- It is accomplished by incrementing the PC value.

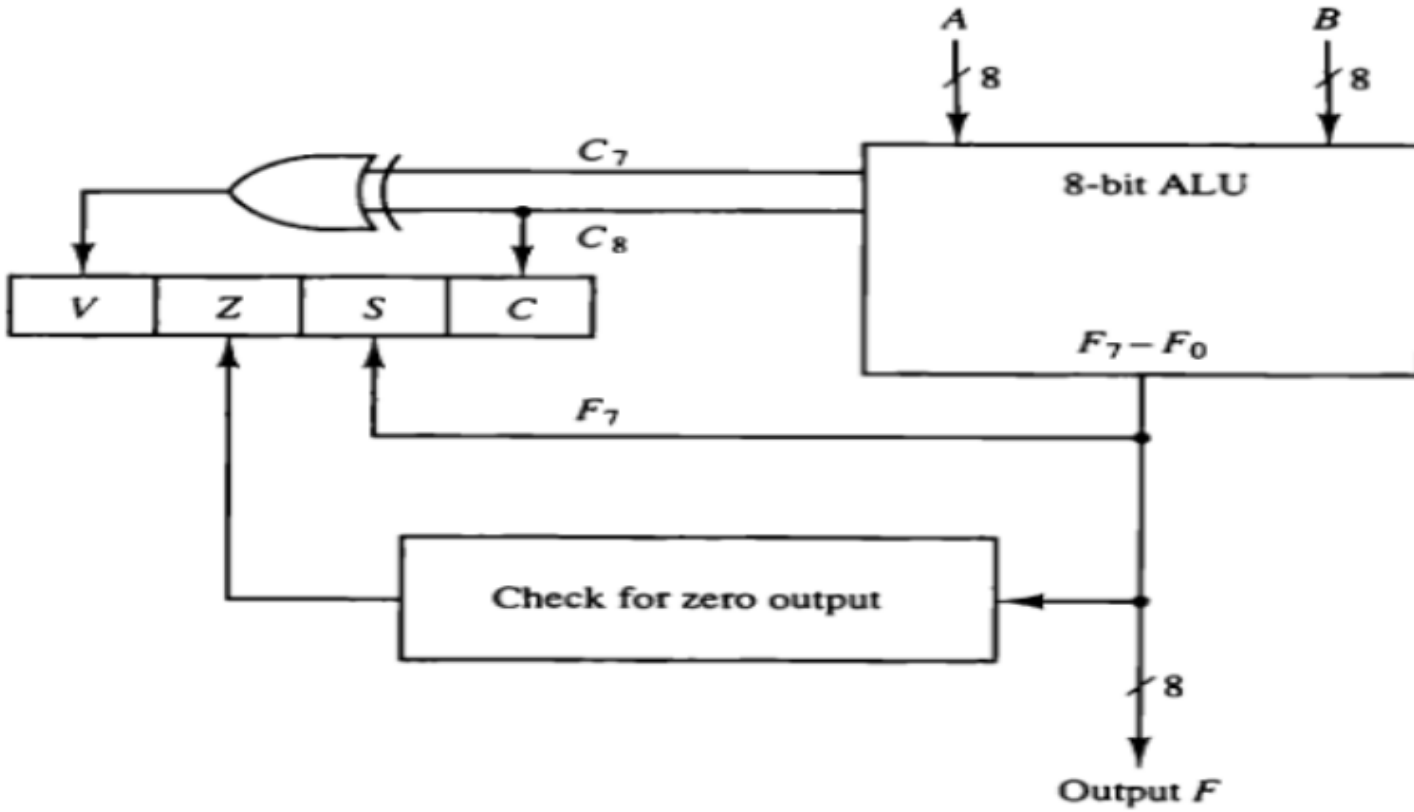
Status bit condition

- It is sometime convenient to supplement ALU with status register where status condition are stored.
- Status bit are also called flag bit or condition code.
- The four status bit are V,Z,S and C.
- The bit are set or cleared according to operation performed by ALU.

- **Bit C (carry)** is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is O.
- **Bit S (sign)** is set to 1 if the highest-order bit F7 is 1. It is set to 0 if the bit is 0.
- **Bit Z (zero)** is set to 1 if the output of the ALU contains all 0's.In other words, Z = 1 if the output is zero and Z = 0 if the output is not.
- **Bit V (overflow)** is set to 1 if the exclusive-OR of the last two carries is equal to 1, and Cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement.

In 2's complement.

For the 8-bit ALU, V = 1 if the output is greater than +127 or less than -128.



Status register bits.

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
	Unsigned compare conditions ($A - B$)	
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
	Signed compare conditions ($A - B$)	
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Subroutine Call and Return

- A subroutine is a self-contained sequence of instructions that performs a given computational task.
- The instruction that transfers program control to a subroutine is known by different names. The most common names used are call ***subroutine, jump to subroutine, branch to subroutine, or branch and save address***.
- The instruction is executed by performing two operations:

- The address of the next instruction available in the program counter (the return address) is Stored in a temporary location so the subroutine knows where to return.
- Control is transferred to the beginning of the subroutine.
  Different computers use a different temporary location for storing the return address. Some store the return address in the ***first memory location of the subroutine, some store it in a fixed location in memory, some store it in a processor register, and some store it in a memory stack.***

- The most efficient way is to store the return address in a memory stack. The advantage of using a stack for the return address is that when a succession of subroutines is called, the sequential return addresses can be pushed into the stack. The return from subroutine instruction causes the stack to pop and the contents of the top of the stack are transferred to the program counter.

- **A subroutine call is implemented with the following micro operations:**
  SP ←SP – 1 Decrement stack pointer
  M [SP] ←PC Push content of PC onto the stack
  PC ← effective address Transfer control to the subroutine
  – If another subroutine is called by the current subroutine, the new return address is pushed into The stack and so on.
- **The instruction that returns from the last subroutine is implemented by the Micro operations:**
  PC←M [SP] Pop stack and transfer to PC
  SP ←SP + 1 Increment stack pointer

Program Interrupt

- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed.
- The interrupt procedure is, in principle, quite similar to a subroutine call except for three Variations:

- (1) The interrupt is usually initiated by an internal or external signal rather than from the Execution of an instruction (except for software interrupt).
- (2) The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.
- (3) An interrupt procedure usually stores all the information

- The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined
  From:
  1. The content of the program counter
  2. The content of all processor registers
  3. The content of certain status conditions
  - **Program status word the** collection of all status bit conditions in the CPU is sometimes called a program status word or PSW. The PSW is stored in a separate hardware register and contains the status information that characterizes the state of the CPU.

- **Types of Interrupts**
  - There are three major types of interrupts that cause a break in the normal execution of a Program. They can be classified as:
  1. External interrupts 2. Internal interrupts
  3. Software interrupts
  - External interrupts come from input-output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.

- **Internal interrupts** arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called
traps. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.
- A **software interrupt** is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.