Microprocessor and Computer Architecture

Unit -1 Introduction of Microprocessor

1.1 Evolution of microprocessor and it's types

Evolution of Microprocessors

The Intel 4004 is a 4-bit central processing unit (CPU) released by Intel Corporation in 1971. It was the first commercially available microprocessor. It began as the "Busicom Project", a joint development by America's Intel and Japan's Busicom with initial design concepts by Busicom's Masatoshi Shima and Sharp's Tadashi Sasaki in 1968, before being designed by Intel's Marcian Ted Hoff and Federico Faggin and Busicom'sShima from 1969 to April 1970. it was completed under Faggin's leadership, with Shima's assistance, in January 1971. It was integrated with 2300 transistors with a max CPU clock rate 740KHZ. We can see details of more microprocessors in following table:

Name	Year	Transistors	Data Width	Clock Speed
Intel 4004	1971	2300	4 bits	740 KHZ
Intel 4040	1974	3000	4	500-740 KHZ
Intel 8008	1972	3300	8	0.2-0.8 MHZ
Intel 8080	1974	6000	8	2 MHZ
Intel 8085	1976	6500	8	5 MHZ
Intel 8086	1978	29000	16	5 MHZ
Intel 8088	1979	29000	8	5 MHZ
Intel 80286	1982	134,000	16	6 MHZ
80386	1985	275,000	32	16 MHZ

80486	1989	1,200,000	32	25 MHZ
Pentium	1993	3,100,000	32/64 bits	60 MHZ
Pentium II	1997	7,500,000	64	233 MHZ
Pentium III	1999	9,500,000	64	450 MHZ
Pentium IV	2000	42,000,000	64	1.5 GHZ
Intel Itanium	2001	25,000,000	64	2 GHZ

Table: 1.1

Fig1.1: 16-pin Intel 4004 first microprocessor

Microprocessors were categorized into five generations: first, second, third, fourth, and fifth generations. Their characteristics are described below: **1) First-generation**

The microprocessors that were introduced in 1971 to 1972 were referred to as the first-generation system. First generation microprocessors processed their instructions serially—they fetched the instruction, decoded it, and then executed it. When an instruction was completed, the microprocessor updated the instruction pointer and fetched the next instruction, performing this sequential drill for each instruction in turn.

2) Second generation

By the late 1970s, enough transistors were available on the IC to usher in the second generation of microprocessor sophistication:16-bit arithmetic and pipelined instruction processing.

Motorola's MC68000 microprocessor, introduced in 1979, is an example. Another example is Intel's 8080. This generation is defined by overlapped fetch, decode, and execute steps (Computer 1996). As the first instruction is processed in the execution unit, the second instruction is decoded and the third instruction is fetched. The distinction between the first and second-generation devices was primarily the use of newer semiconductor technology to fabricate the chips. This new technology resulted in a five-fold increase in instruction, execution, speed, and higher chip densities.

3) Third generation

The third generation, introduced in 1978, was represented by Intel's 8086 and the Zilog-Z8000, which were 16-bit

processors with minicomputer-like performance. The third generation came about as IC transistor counts approached 250,000. Motorola's MC68020, for example, incorporated an on-chip cache for the first time and the depth of the pipeline increased to five or more stages. This generation of microprocessors was different from the previous ones in that all major workstation manufacturers began developing their own RISC-based microprocessor architectures (Computer, 1996).

4) Fourth generation

As the workstation companies converted from commercial microprocessors to inhouse designs, microprocessors entered their fourth generation with designs surpassing a million transistors. Leading-edge microprocessors such as Intel's 80960CA and Motorola's 88100 could issue and retire more than one instruction per clock cycle.

5) Fifth generation

Microprocessors in their fifth generation, employed decoupled super scalar processing, and their design soon

surpassed 10 million transistors. In this generation, PCs are a low-margin, high-volume-business dominated by a single microprocessor.

Von Neumann Architecture or IAS (Immediate Access Store)

A Von Neumann Architecture machine, designed by physicist and mathematician John von Neumann (1903–1957) is a theoretical design for a stored program computer that serves as the basis for almost all modern computers. A Von-Neumann Machine consists of a central processor with an arithmetic/logic unit and a control unit, a memory, mass storage, and input and output.

The von Neumann machine was created by its namesake, John Von Neumann, a physicist and mathematician, in 1945, building on the work of Alan Turing. The design was published in a document called "First Draft of a Report on the EDVAC."

The report described the first stored-program computer. Earlier computers, such as the ENIAC, were hard-wired to do one task. If the computer had to perform a different task, it had to be rewired, which was a tedious process. With a stored-program computer, a general purpose

computer could be built to run different programs. The theoretical design consists of:

- A central processor consisting of a control unit and an arithmetic/logic unit
- A memory unit
- Input and output unit

The ***Von Neumann*** Design thus forms the basis of modern computing. A similar model, the Harvard architecture, had dedicated data address and buses for both reading and writing to memory. The von Neumann Architecture won out because it was simpler to implement in real hardware

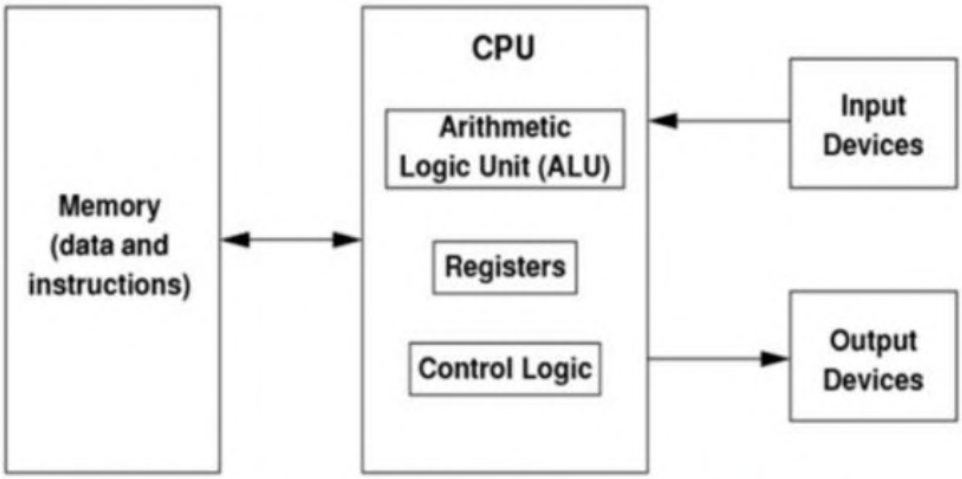


Figure1.2: General Von Neumann Architecture

Types of Microprocessor

There are basically 5 kinds of microprocessors :

Complex Instruction Set Microprocessors: They are also called as CISM in short and they categorize a microprocessor in which orders can be executed together along with other low-level activities. It mainly performs the task of uploading, downloading and recalling data into and from the memory card. Apart from that it also does complex mathematical calculations within a single command.

Reduced Instruction Set Microprocessor : This processor is also called as RISC in short. These kinds of chips are made according to the function in which the microprocessor can carry out small things within a particular command. In this way it completes more commands at a faster rate.

Superscalar Processors : This is a processor that copies the hardware on the microprocessor for performing numerous tasks at a time. They can be used for arithmetic and as multipliers. They have several operational units and thus carry out more than a one command by constantly transmitting various instructions to the superfluous operational units inside the processor.

The Application Specific Integrated Circuit : This processor is also known as ASIC. They are used for specific purposes that comprises of automotive emissions control or personal digital assistant computer. This kind of processor is made with

proper specification but apart from that it can also be made using the off the shelf gears.

Digital Signal Multiprocessors : Also called as DSP's, these are used for encoding and decoding videos or to convert the digital and video to analog and analog to digital. They need a microprocessor that is excellent in mathematical calculations. The chips of this processor are employed in SONAR, RADAR, home theaters audio gears, Mobile phones and TV set top boxes

1.2 Microprocessor Bus organization: Data Bus, Address Bus and Control Bus

Bus Structure of 8085

A bus in a microprocessor-based system is defined as a group of separate wires which work together to perform a particular task. A microprocessor-based system, or microcomputer, has three buses which combine to transfer information between the microprocessor and other parts of the system, such as memory or input/output devices. Typical tasks performed by these buses include selecting the source or destination location address for a data transfer, actually moving the data from one part of the system to another, and finally, controlling

and synchronizing the electronic devices involved in the data transfer process ***There are three buses in Microprocessor:***

1) Address Bus
2) Data Bus
3) Control Bus

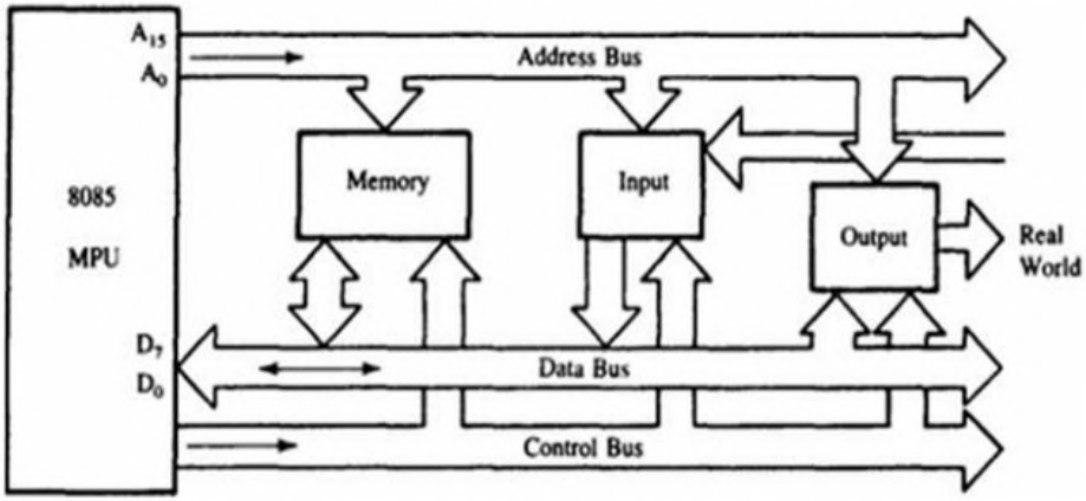


Fig2.1: 8085 Bus Structure

Address Bus: Generally, microprocessor has 16-bit address bus(A0-A15). 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 the memory location to be written or to be read from. The address bus is unidirectional. It means bit flowing occurs only in one direction, only from the microprocessor to the peripheral devices.We can

find how much memory location is using by microprocessor by the formula 2N where N is the number of bits used for address lines. Here 2 16 =

65536 bits or 64KB. So, we can say 8085 can access 64KB of memory locations

Data Bus: 8085 microprocessor has 8-bit data buses. So, it can be used to carry 8 bits of data starting from 00000000(00H) to 11111111(FFH). Here H tells hexadecimal number. The data buses are bidirectional. These lines are used for data flowing in both direction means data can be transferred or received through these lines. The data bus also connects I/O ports and the CPU. The largest number that can be appeared in the bus is 11111111. It has 8 parallel lines of data bus so it can access up to 28 =256 data bus lines.

Control Bus: The control bus is used for sending control signals to the memory and I/O devices. The CPU sends control signal on the control bus to enable the outputs of addressed memory devices or I/O port devices. Microprocessors are the central processing units (CPUs) of computers and other electronic devices. They perform a wide range of operations, including internal data manipulation, microprocessor-initiated operations, and peripheral or external-

initiated operations. Here's an overview of these three categories:

1.3 Operations of microprocessor: internal data manipulation, microprocessor initiated and peripheral or external initiated **Operations of microprocessor:-**

- Internal data manipulation:
  1. **Arithmetic Operations**: Microprocessors can perform basic arithmetic operations like addition, subtraction, multiplication, and division on data stored in their internal registers
  2. **Logic Operations**: They can execute logical operations such as AND, OR, XOR, and NOT on binary data
  3. **Data Movement**: Microprocessors can move data between registers, memory, and other internal components. This includes instructions to load data into registers, store data in memory, or transfer data between registers.
- Microprocessor-Initiated Operations:
  1. **Program Execution**: Microprocessors fetch and execute instructions stored in memory. These

instructions can be arithmetic and logic operations, control flow instructions, or data manipulation instructions.

2. **Control Flow**: The microprocessor can change the sequence of execution by branching (jumping) to different parts of the program based on conditions or executing loops.

3. **Interrupt Handling**: Microprocessors can handle interrupts, which are external signals that temporarily halt normal program execution to process specific tasks or events.

4. **Exception Handling:** They can also handle exceptions, which are unexpected events or errors that require special processing.

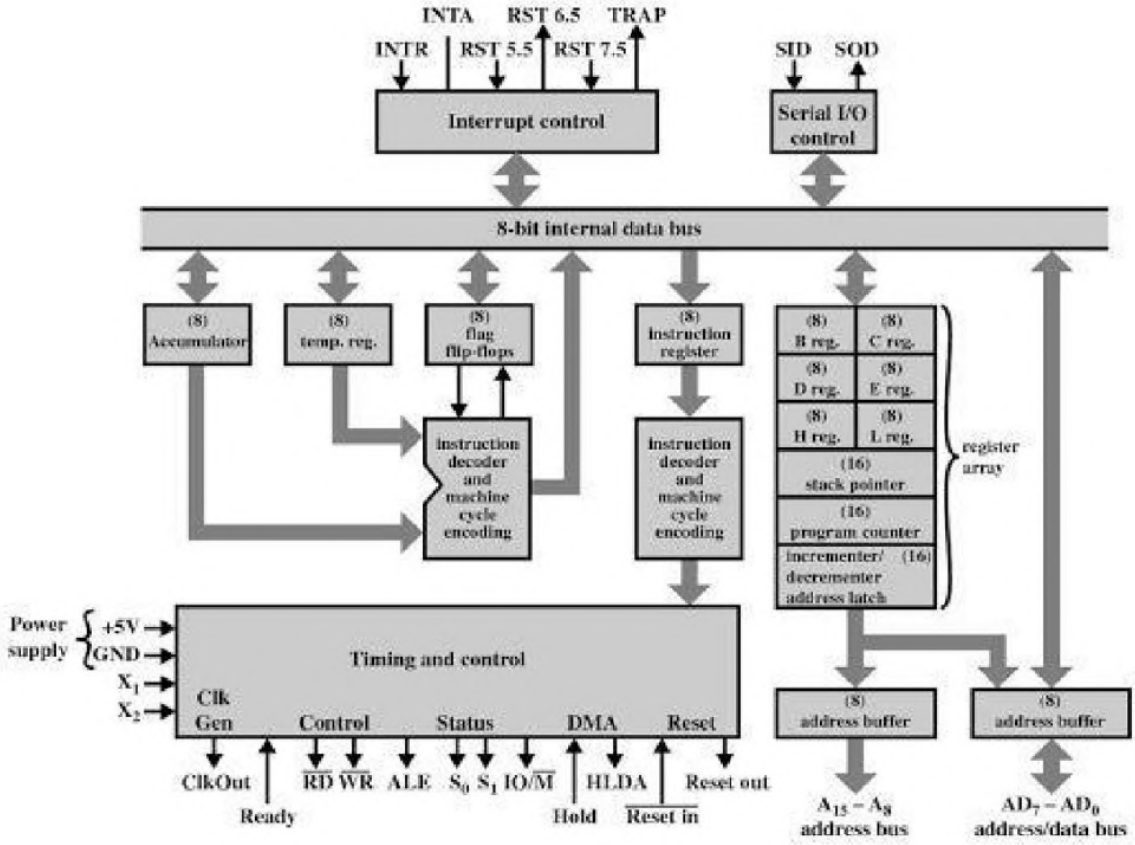
- Peripheral or External-Initiated Operations:
  1. **Input/Output (I/O) Operations**: Microprocessors communicate with external devices and peripherals, such as keyboards, displays, storage

devices, and sensors, through input and output instructions.

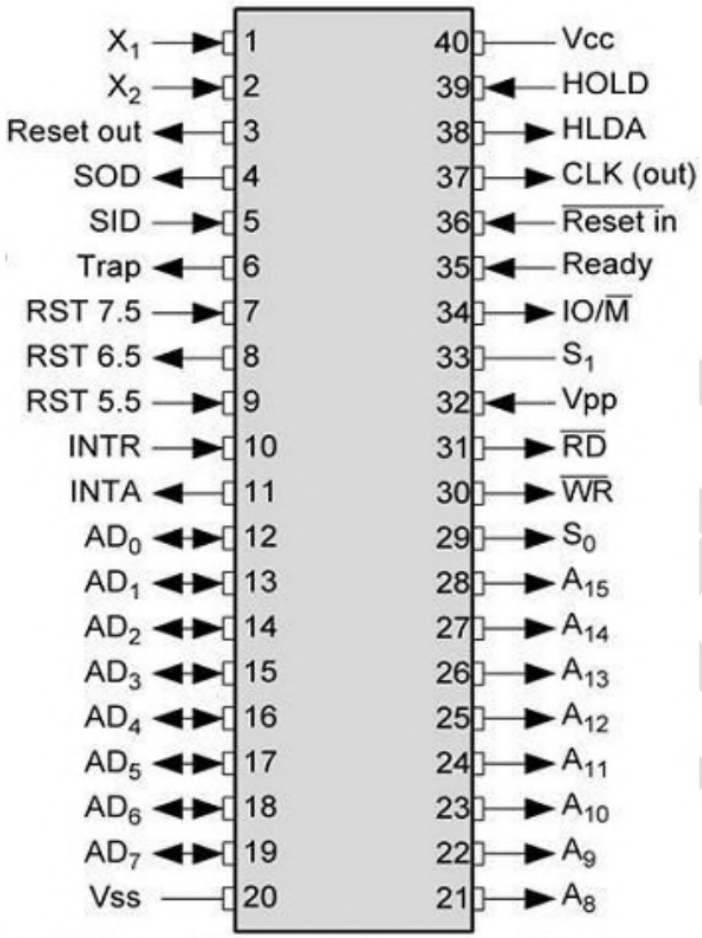
2. **DMA (Direct Memory Access):** Some microprocessors support DMA controllers, which allow peripherals to transfer data directly to or from memory without CPU intervention.

3. **Communication**: Microprocessors can communicate with other devices or microcontrollers through various communication protocols like UART, SPI, I2C, and Ethernet.

1.4 Pin diagram and internal Architecture of 8085


INTA
RST 6.5
TRAP
INTR
RST 5.5
RST 7.5
Interrupt control
SID
SOD
Serial I/O control
8-bit internal data bus
(8) Accumulator
(8) temp. reg.
(8) flag flip-flops
(8) instruction register
(8) B reg.
(8) C reg.
(8) D reg.
(8) E reg.
(8) H reg.
(8) L reg.
(16) stack pointer
(16) program counter
incrementer/ decrementer address latch
(16)
register array
instruction decoder and machine cycle encoding
instruction decoder and machine cycle encoding
Power supply
+5V
GND
X1
X2
Timing and control
Clk Gen
Control
Status
DMA
Reset
ClkOut
Ready
RD WR
ALE
S0 S1 IO/M
HLDA
Hold
Reset in
Reset out
(8) address buffer
(8) address buffer
A15 – A8 address bus
AD7 – AD0 address/data bus

Pin Name	Description	Type
AD_0 - AD_7	Address / Data Bus	Bidirectional, Tristate
A_8 - A_{15}	Address Bus	Output, Tristate
ALE	Address Latch Enable	Output, Tristate
$\overline{RD}$	Read Control	Output, Tristate
$\overline{WR}$	Write Control	Output, Tristate
IO/$\overline{M}$	I/O or memory Indicator	Output, Tristate
S_0, S_1	Bus State Indicator	Output
READY	Wait state request	Input
SID	Serial Input Data	Input
SOD	Serial Output Data	Output
HOLD	HOLD request	Input
HLDA	HOLD acknowledge	Output
INTR	Interrupt request	Input
TRAP	Nonmaskable interrupt request	Input
RST 5.5	Hardware vectored Interrupt request	Input
RST 6.5	Hardware vectored Interrupt request	Input
RST 7.5	Hardware vectored Interrupt request	Input
$\overline{INTA}$	Interrupt acknowledge	Output
$\overline{RESET\ IN}$	System reset	Input
RESET OUT	Peripherals reset	Output
X_1, X_2	Crystal or RC Connection	Input
CLK (OUT)	Clock Signal	Output
V_{cc}, V_{ss}	Power, ground	.....



Description of each blocks: Registers, Flag, Data and Address Bus, Timing and Control Unit, Interrupts

- **Accumulator**: It is an 8-bit register which is used to perform arithmetical and logical operation. It stores the output of any operation. It also works as registers for I/O accesses.

- **Temporary Register**:-It is an 8-bit register which is used to hold the data on which the accumulator is computing operation. It is also called as operand register because it provides operands to ALU
- **Registers**: These are general purposes registers. Microprocessor consists 6 general purpose registers of 8-bit each named as B, C, D, E, H and L. Generally, theses registers are not used for storing the data permanently. It carries the 8-bits data. These are used only during the execution of the instructions. These registers can also be used to carry the 16 bits data by making the pair of 2 registers. The valid register pairs available are BC, DE and HL. We cannot use other pairs except BC, DE and HL. These registers are programmed by the user.

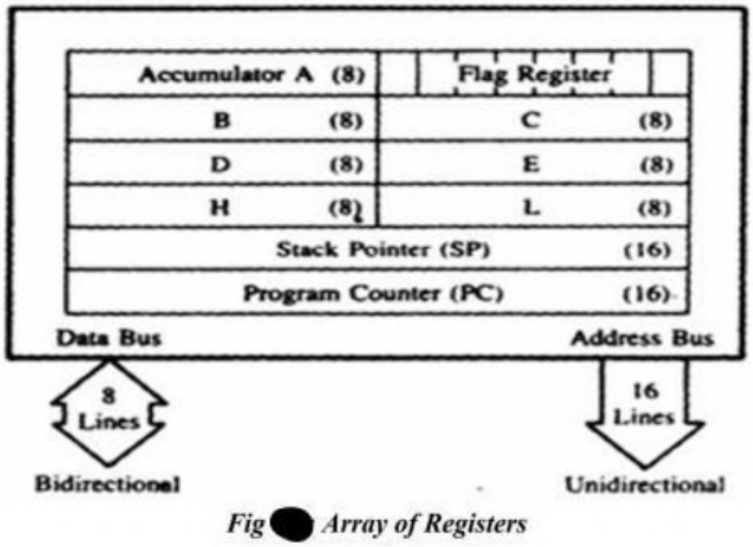


Fig Array of Registers

- **ALU**: Algorithm Logical Units performs the arithmetic operations and logical operation.
- **Flag Registers**: It consists of 5 flip flop which changes its status according to the result stored in an accumulator. It is also known as status registers. It is connected to the ALU.

There are five flip-flops in the flag register are as follows:

1. Sign(S) 2. Zero(z) 3. Auxiliary carry(AC) 4. Parity(P) 5. Carry(C)

The bit position of the flip flop in flag register is:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	***Z***		***AC***		***P***		***CY***

All of the threeflip flop set and reset according to the stored result in the accumulator.

1. **Sign**(S)- If D7 of the result is 1 then sign flag is set otherwise reset. As we know that a number on the D7 always decides the sign of the number. if D7 is 1: the number is negative. if D7 is 0: the number is positive.

2. **Zeros(Z)-**If the result stored in an accumulator is zero then this flipflop is set otherwise it is reset.

3. **Auxiliary carry(AC)-**If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.

4. **Parity**(**P**)-If the no of 1's is even in the output stored in the accumulator then it is set otherwise it is reset for the odd.

5. **Carry(C)-**If the result stored in an accumulator generates a carry in its final output then it is set otherwise it is reset.

- **Instruction registers(IR):** It is a 8-bit register. When an instruction is fetched from memory then it is stored in this register.
- **Instruction Decoder**: Instruction decoder identifies the instructions. It takes the information from instruction register and decodes the instruction to be performed.

1.5 Internal registers organization of 8085

- **Program Counter**: It is a 16-bit register used as memory pointer. It stores the memory address of the next instruction to be executed. So, we can say that this

register is used to sequencing the program. Generally the memory have 16-bit addresses so that it has 16 bit memory. The program counter is set to 0000H.

- **Stack Pointer**: It is also a 16-bit register used as memory pointer. It points to the memory location called stack. Generally, stack is a reserved portion of memory where information can be stores or taken back together.
- **Timing and Control Unit:-** It provides timing and control signal to the microprocessor to perform the various operation. It has three control signals. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers. There are three control signals:

1. ALE-Arithmetic Latch Enable, itprovides control signal to synchronize the components of microprocessor.
2. RD-This is active low used for reading operation.
3. WR-This is active low used for writing operation.

There are three status signals used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.

IO/M(Active Low)	S1	S2	Data Bus Status(Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

Fig [redacted] Timing and control signals

Serial Input Output Control- There are two pins in this unit. This unit is used for serial data communication.

Interrupt Unit- An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a highpriority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service

routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. There are two types of interrupts: hardware interrupts and software interrupts. There are 6 interrupt pins in this unit. Generally, an external hardware is connected to these pins. These pins provide interrupt signal sent by external hardware to microprocessor and microprocessor sends acknowledgement for receiving the interrupt signal. Generally, INTA is used for acknowledgement.

Register Section: Many registers have been used in microprocessor

Trick :- https://www.youtube.com/watch?v=I_tqbZ7UKk

1.6 Limitations of 8085

- Multiplexed address and data bus, so extra hardware is required to separate address and the data signals.
- Limited flags and interrupts.
- Limited memory and processing power.
- Operating frequency is less, so the speed of execution is slow.
- Overall product design requires more time.
- The processor has a limitation on the size of data.

- Limited bandwidth and latency

Unit 2 : Instruction Cycle and Timing Diagram

The time needed for completing one operation of accessing memory, I/O or acknowledging an external request is termed as Machine cycle. It is comprised of T-states. One subdivision of the operation completed in one clock period is termed as T-state. The following are the various machine cycles of 8085 microprocessor.

1. **Opcode Fetch (OF)**
2. **Memory Read (MR)**
3. **Memory Write (MW)**
4. **I/O Read (IOR)**
5. **I/O Write (IOW)**
6. **Interrupt Acknowledge (IA)**
7. **Bus Idle (BI)**

All instructions have at least one Opcode Fetch machine cycle. Depending on the type of instruction one or more other machine cycles are required to complete the execution of the instruction. The number and type of machine cycles for different instructions are shown in table.

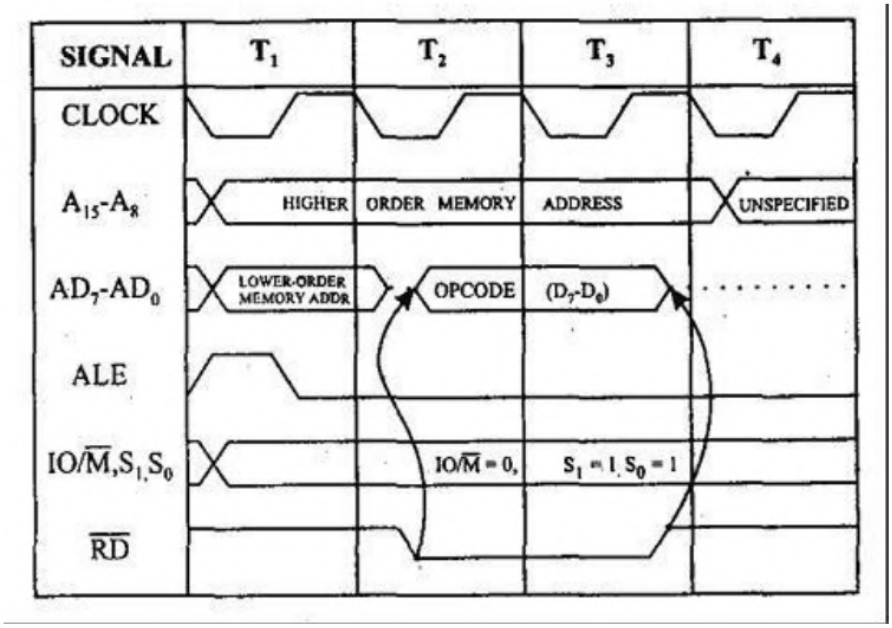
S No	Instruction	No: of machine cycles	Machine cycle - 1	Machine cycle - 2	Machine cycle - 3	Machine cycle - 4
1	MOV A,B	1	OF	-	-	-

2	MVI A, 50H	2	OF	MR	-	-
3	LDA 5000H	4	OF	MR	MR	MR
4	STA 5000H	4	OF	MR	MR	MW
5	IN 80H	3	OF	MR	IOR	-
6	OUT 80H	3	OF	MR	IOW	-

Opcode Fetch (OF) machine cycle of 8085:

Each instruction of the microprocessor has one byte Opcode. The Opcode is stored in memory. In order to fetch the
Opcode from memory, processor executes the Opcode Fetch machine cycle. So, every instruction starts with Opcode Fetch machine (OFM) cycle. The time taken by the microprocessor to execute the Opcode Fetch cycle is 4T (T- states). Inorder to fetch the Opcode from memory, the first 3 T-states are used. The remaining T-state is used for internal operations by the microprocessor.

The timing diagram for Opcode Fetch machine cycle is shown in figure.



The steps in Opcode Fetch machine cycle are given in table.

S. No	T state	Operation
1	T_1	The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus.
2		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3		The status signals are changed as IO/M ' = 0, S1 =1 and S0 = 1. These status signals do not change throughout the OF machine cycle.
4	T_2	The microprocessor makes the RD' line LOW to enable memory read and increments the Program Counter.
5		The contents on D7 – D0 (i.e. the Opcode) are placed on the address / data bus.

6		The microprocessor transfers the Opcode on the address / data bus to Instruction Register (IR).
7	T_3	The microprocessor makes the RD' line HIGH to disable memory read.
8	T_4	The microprocessor decodes the instruction.

Memory Read Machine Cycle of 8085:

Single byte instructions require only Opcode Fetch machine cycles. But, 2-byte and 3-byte instructions require additional machine cycles to read the operands from memory. The additional machine cycle is called Memory Read machine cycle. For example, the instruction MVI A, 50H requires one OF machine cycle to fetch the operand from memory and one MR machine cycle to read the operand (50H) from memory. The MR machine cycle takes 3 T-states. The timing diagram for Memory Read machine cycle is shown in figure.

Timing Diagram for Memory Read Machine Cycle

The steps in Memory Read machine cycle are given in table.

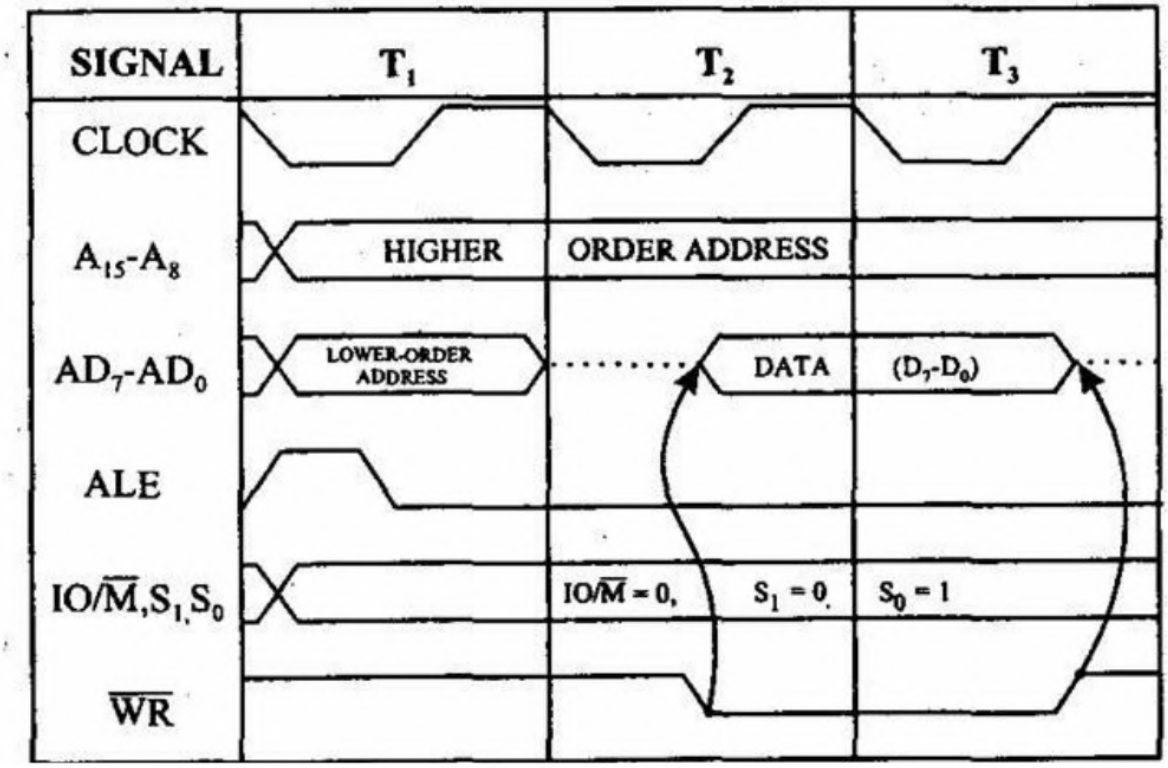
S. No	T state	Operation
1	T_1	The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus.

2		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3		The status signals are changed as IO/$\overline{M}$ ' = 0, S1 =1 and S0 = 0. These status signals do not change throughout the memory read machine cycle.
4	T_2	The microprocessor makes the RD' line LOW to enable memory read and increments the Program Counter.
5		The contents on D7 – D0 (i.e. the data) are placed on the address / data bus.
6	T_3	The data loaded on the address / data bus is moved to the microprocessor.
7		The microprocessor makes the RD' line HIGH to disable the memory read operation.

Memory Write Machine Cycle of 8085:

Microprocessor uses the Memory Write machine cycle for sending the data in one of the registers to memory. For example, the instruction STA 5000H writes the data in accumulator to the memory location 5000H. The MW machine cycle takes 3 T-states.

The timing diagram for Memory Write machine cycle is shown in figure.



Timing Diagram for Memory Write Machine Cycle

The steps to disable the memory write machine cycle are given in table.

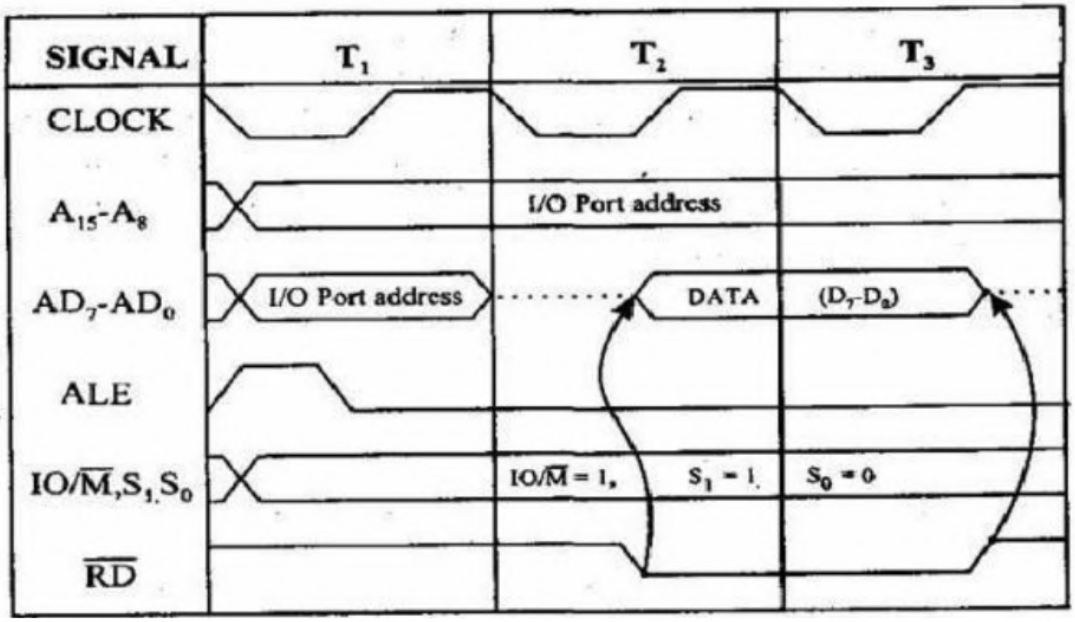
S. No	T state	Operation
1		The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus.
2		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3	T_1	The status signals are changed as IO/*M*' = 0, S1 =0 and S0 = 1. These status signals do not change throughout the memory write machine cycle.

4	T_2	The microprocessor makes the *WR*' line LOW to enable memory write.
5		The contents of the specified register are placed on the address / data bus.
6	T_3	The data placed on the address / data bus is transferred to the specified memory location.
7		The microprocessor makes the *WR*' line HIGH to disable the memory write operation.

I/O Read Machine Cycle of 8085

Microprocessor uses the I/O Read machine cycle for receiving a data byte from the I/O port or from the peripheral in I/O mapped I/O systems. The IN instruction uses this machine cycle during execution. The IOR machine cycle takes 3 T-states.

The timing diagram for I/O Read machine cycle is shown in figure.



Timing Diagram for I/O Read Machine Cycle

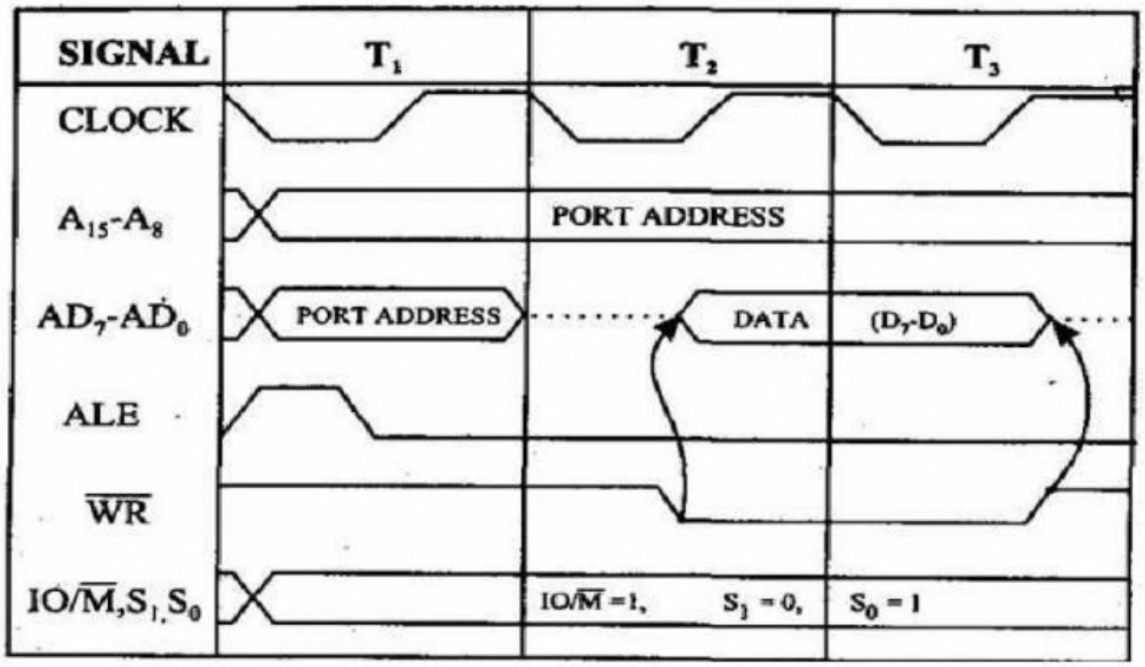
The steps in I/O Read machine cycle are given in table.

S. No	T state	Operation
1	T_1	The microprocessor places the address of the I/O port specified in the instruction on A15 – A8 address bus and also on AD7 – AD0 address / data bus.
2		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3		The status signals are changed as IO/*M*' = 0, S1 =1 and S0 = 0. These status signals do not change throughout the I/O read machine cycle.
4	T_2	The microprocessor makes the *RD*' line LOW to enable I/O read.
5		The contents on D7 – D0 (i.e. the data) are placed on the address / data bus.
6	T_3	The data loaded on the address / data bus is moved to the microprocessor ie., to the accumulator.
7		The microprocessor makes the *RD*' line HIGH to disable the I/O read operation.

I/O Write Machine Cycle of 8085

Microprocessor uses the I/O Write machine cycle for sending a data byte to the I/O port or to the peripheral in I/O mapped I/O systems. The OUT instruction

uses this machine cycle during execution. The IOR machine cycle takes 3 T-states. The timing diagram for I/O Write machine cycle is shown in figure.



Timing Diagram for I/O Write Machine Cycle

The steps in I/O Read machine cycle are given in table.

S. No	T state	Operation
1	T_1	The microprocessor places the address of the I/O port specified in the instruction on A15 – A8 address bus and also on AD7 – AD0 address / data bus.
2		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3		The status signals are changed as IO/*M*' = 0, S1 =0 and S0 = 1. These status signals do not change throughout the I/O write machine cycle.

4		The microprocessor makes the *WR*' line LOW to enable I/O write.
5	T_2	The contents of the Accumulator are placed on the address / data bus.
6		The data placed on the address / data bus is transferred to the specified I/O port.
7	T_3	The microprocessor makes the *WR*' line HIGH to disable the I/O write operation

2.2 Bus timings to fetch, decode, execute instruction from memory

The concept of "bus timings" in the context of fetching, decoding, and executing instructions from memory is related to the operation of a computer's central processing unit (CPU). In a computer system, buses are communication channels that transfer data between different components, such as the CPU, memory, and input/output devices.

The process of executing instructions involves several steps, and buses play a crucial role in facilitating the flow of information between these steps.

Fetch:

- Imagine your computer's brain (CPU) needs to know what to do next.
- It looks at a special note called the Program Counter (PC), which tells it where to find the next instruction in its memory.
- The CPU asks the memory for that instruction.

- The memory responds by handing over the instruction on a virtual conveyor belt (address bus), and the CPU catches it (data bus).
- The CPU puts this instruction in a special holding area called the Instruction Register (IR).

Decode:

- Now, the CPU looks at the instruction in the Instruction Register.
- It's like reading a recipe. The CPU figures out what needs to be done based on the instruction.
- This step is decoding – understanding the instruction's meaning.
- For example, if the instruction says "add two numbers," the CPU knows it needs to add numbers in the next step.

Execute:

- The CPU starts doing what the instruction says. If it's an "add two numbers" instruction, it gets the numbers from the right places (like registers or memory).
- It performs the addition or whatever operation the instruction requires.
- The result is then either stored or used for the next instruction.
- This step is executing – carrying out the actual task specified by the instruction.

Unit – 3 8085 Instruction set

5.1 Instruction format and data format

An 8085program instructions format may be one, two or three length.

1. 1-byte instruction (8 bit)

1-byte instructionincludesoperand and opcode in the same byte. It occupies 1-byte space in memory.

E.g.

Opcode operand

MOV A, B

ADD B

CMA

2. Byte instruction (16 bit)

In a 2-byte instruction, the first byte specifies the operation and second byte specifies opcode. It occupies 2-byte space in memory

Opcode

DATA OR ADDRESS

E.g.

Opcode	Operand
MVI	A, 45H
MVI	B, F2H

3. 3-byte instruction

In 3-byte instruction the first byte specifies the opcode and following two byte

specifies the 16-bit address. It occupies 3-byte space in memory.

E.g.

Opcode	Operand	Hex code
LDA	2050H	3A
		50
		20
JMP	2085H	C3
		85
		20

Instruction with a Memory Address

Operation: go to address 2085.

Instruction: JMP 2085

Opcode: JMP

Operand: 2085

Binary code

1100 0011	C3	1st byte
1000 0101	85	2nd byte
0010 0000		203rd byte

Data Format

In 8085, data stores in the form of 8-bit binary integer. In the Intel 8085, bit 0 is referred as the least signification bit (LSB) and bit 7 is referred as most signification bit (MBS).

Data can be represented in four formats

1. ASCII (American Standard Code for Information Interchange) e.g. - A-Z, a-z, 0-9 etc.

2. BCD (Binary Code Decimal) e.g. - 0-F
3. Signed Integer e.g. +2, +3,
4. Unsigned Integer e.g. -2, -3,

Various Addressing Modes:-

Types of addressing modes –

1) Immediate Addressing Mode –

In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Examples:

- MVI B 45 (move the data 45H immediately to register B)
- LXI H 3050 (load the H-L pair with the operand 3050H immediately)
- JMP address (jump to the operand address immediately)

2) Register Addressing Mode –

In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore the operation is performed within various registers of the microprocessor.

Examples:

- MOV A, B (move the contents of register B to register A)

- ADD B (add contents of registers A and B and store the result in register A)
- INR A (increment the contents of register A by one)

3) **Addressing Mode –**

In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.

Examples:

- LDA 2050 (load the contents of memory location into accumulator A)
- LHLD address (load contents of 16-bit memory location into H-L register pair)
- IN 35 (read the data from port whose address is 35)

4) **Register Indirect Addressing Mode –**

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

Examples:

- MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)
- LDAX B (move contents of B-C register to the accumulator)

- STAX B (store accumulator contents in memory pointed by register pair B-C)

Instruction

An Instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described here are of Intel 8085.The *instruction set* of the 8085 microprocessor consists of 74 instructions with 246 different bit patterns.These instructions are of Intel Corporation. They cannot be used byother microprocessor manufactures. The programmer can write a program in assembly language using these instructions. These instructions have been classified as below:

1. Data transfer instruction

This instruction copies data from one location to another location. There are various types of data transfers.

Between register

E.g. MOV A, B; MOV C, B etc.

Specify the data byte to register or memory location.

E.g. MVI B, 23H; LDA 2500H etc.

Between memory location and register

E.g. LXI 2050H

Between input and output

E.g. IN 02H; OUT PORT1 etc.

2. Arithmetic instruction

This instruction performs arithmetic operation such as addition, subtraction, increment, and decrement. E.g. ADD B, ADI 25H, SUB C, SUI 23H, SUB M, INR B, DCR A etc.

Addition (ADD, ADI)

– Any 8-bit number.

– The contents of a register.

– The contents of a memory location.

- Can be added to the contents of the accumulator and the result is stored in the accumulator.

Subtraction (SUB, SUI)

– Any 8-bit number

– The contents of a register

– The contents of a memory location

- Can be subtracted from the contents of the accumulator. The result is stored in the accumulator.

Increment (INR) and Decrement (DCR)

- The 8-bit contents of any memory location or any register can be directly incremented or decremented by 1.
- No need to disturb the contents ofthe accumulator

3. Logical instruction

This instruction performs logical operation like AND, OR, COMPARE, ROTATED.

CMP- compare

RAL- rotation left

RAR- rotation right

ANA- AND

ORA- OR

These instructions perform logic operations on the contents of the accumulator – ANA, ANI, ORA, ORI, XRA and XRI

- **Source: Accumulator and**
  - An 8-bit number
  - The contents of a register
  - The contents of a memory location

- **Destination: Accumulator**

ANA	R/M	AND Accumulator withReg/Mem
ANI	#	AND Accumulator With an 8-bit number
ORA	R/M	OR Accumulator withReg/Mem
ORI	#	OR Accumulator With an 8-bit number
XRA	R/M	XOR Accumulator withReg/Mem
XRI	#	XOR Accumulator With an 8-bit number

- **Complement**
- 1's complement of the contents of the accumulator.
  CMA No operand
- Rotate – Rotate the contents of the accumulator one position to the left or right.

– RLC Rotate the accumulator left. Bit 7 goes to bit 0AND the Carry flag.

– RAL Rotate the accumulator left through the carry. Bit 7 goes to the carry and carry goes to bit 0.

– RRC Rotate the accumulator right. Bit 0 goes to bit 7 AND the Carry flag.

– RAR Rotate the accumulator right through the carry. Bit 0 goes to the carry and carry goes to bit 7.

- **Compare**
- Compare the contents of a register or memory location with the contents of the accumulator.

– CMP R/M compares the contents of the register or memory location to the contents of the accumulator.

– CPI # Compare the 8-bit number to the contents of the accumulator.

- The compare instruction sets the flags (Z, Cy, and S).
- The compare is done using an internal subtraction that does not change the contents of the accumulator.

  A –(R / M / #)

4. Branch instruction

Two types

Unconditional branch

Go to a new location no matter what.

Conditional branch

Go to a new location if the condition is true.

Unconditional Branch

– JMP Address Jump to the address specified (Go to).

– CALL Address Jump to the address specified but treat it as a subroutine.

– RET Return from a subroutine.

– The addresses supplied to all branch operations must be 16-bits.

Conditional Branch

– Go to new location if a specified condition is met.

JZ Address (Jump on Zero) – Go to address specified if the Zero flag is set.

JNZ Address (Jump on NOT Zero) – Go to address specified if the Zero flag is not set.

JC Address (Jump on Carry) – Go to the address specified if the Carry flag is set.

JNC Address (Jump on No Carry) – Go to the address specified if the Carry flag is not set.

JP Address (Jump on Plus) – Go to the address specified if the Sign flag is not set

JM Address (Jump on Minus) – Go to the address specified if the Sign flag is set.

5. Machine Control

– HLT

Stop executing the program. – NOP

No operation

Exactly as it says, do nothing.

Usually used for delay or to replace instructions during debugging.

These instruction control machines function such as HLT, NOP, and EI etc.

Microprocessor program

#) Program-01

Write a program to store data type 32H into memory location.

```
MVI A, 32H      : store 32H in the accumulator.
STA 2000H       : copy accumulator content to memory location.
HLT             : stop
```

#) Program-02

Write a program to store data of register B into memory location 4000H.

```
MOV A, B
STA 4000H
HLT
```

#) Program-03

Write a program to exchange the content of memory location 2000H and 4000H.

```
LDA 2000H    : get the content of memory location 2000H into accumulator
MOV B, A     : save the content into B.
LDA 4000H    : get the content of memory location 4000H into accumulator
STA 2000H    : store the content of accumulator at 2000H.
MOV A, B     : save the content back to accumulator.
STA 4000H    : store the content of accumulator into memory location 4000H.
```

HLT : stop

#) **Program-04**

Write a program to add two numbers.

2000H	14H
2001H	89H
2002H	14+89=9D (Result)
LXI H, 2000H :	HL point 2000H.
MOV A, M :	get first operand.
INX H :	HL points 2001H.
ADD M :	Add
INX H :	HL points 2002H.
MOV M, A :	store the result 2003H.
HLT :	stop.

#) **Program-05**

Write a program to subtract two numbers.

4000H	51H
4001H	19H
4002H	51-19=38H (Result)
LXI H, 4000H	: HL point 4000H.
MOV A, M	: get first operand.
INX H	: HL points 4001H.
SUB M	: Add
INX H	: HL points 4002H.
MOV M, A	: store the result 4003H.
HLT	: stop.

#) **Program-06**

Write a program to find the 1's complements of the number stored at memory location 4400H and stored the complements number at memory location 4300H.

```
4400H          55H
43OOH          AAH (Result)
LDA 4400H      : get the number
CMA            : complement
STA 4300H      : stored the result.
HLT            : stop
```

#) **Program-07**

Write a program to find the 1's complements of the number stored at memory location 4400H and stored the complements number at memory location 4300H.

```
4400H          55H
43OOH          AAH+1= ABH (Result)
LDA 4400H      : get the number
CMA            : complement
ADI, 01H       : add one in the number.


STA 4300H      : stored the result.
HLT            : stop
```

#) **Program-08**

Write a program to shift an 8-bit data four bits right. Assume that the data is in the register C.

```
MOV A, C
RAR
RAR
RAR
```

RAR

MOVC, A

HLT

Stack Operations:

1. **Push:**
   - Adds an item to the top of the stack.
   - Commonly used to store data or addresses during subroutine calls.
   - Example: `` `PUSH Register` `` in x86 assembly language.
2. **Pop:**
   - Removes the item from the top of the stack.
   - Typically used to retrieve data or addresses from the stack.
   - Example: `` `POP Register` `` in x86 assembly language.

Stack Instructions:

1. **CALL (Call Subroutine):**
   - Pushes the return address onto the stack and jumps to the specified subroutine.
   - Upon reaching the subroutine's return statement, the return address is popped from the stack.
   - Example: `` `CALL subroutine_label` `` in x86 assembly language.
2. **RET (Return from Subroutine):**
   - Pops the return address from the stack and jumps to that address, returning from a subroutine.
   - Example: `` `RET` `` in x86 assembly language.

3. **PUSH (Push Data onto Stack):**
   - Pushes a value or register content onto the stack.
   - Example: `` `PUSH AX` `` in x86 assembly language.
4. **POP (Pop Data from Stack):**
   - Pops a value from the stack and stores it in a register.
   - Example: `` `POP BX` `` in x86 assembly language.
5. **INT (Interrupt):**
   - Invokes a specific interrupt service routine (ISR), which typically involves pushing necessary registers onto the stack.
   - Example: `` `INT 21h` `` in x86 assembly language for invoking DOS interrupt.
6. **ENTER and LEAVE:**
   - `` `ENTER` `` is an x86 instruction used for creating a stack frame.
   - `` `LEAVE` `` is used to release the stack frame.
   - Example: `` `ENTER 0, 0` `` and `` `LEAVE` `` in x86 assembly language.

Unit 4:

Basic computer architecture

Introduction:

Computer architecture refers to the design and organization of the various components of a computer system to ensure that they work together efficiently to execute instructions and perform tasks. It involves the arrangement and interconnection of hardware components, including the central processing unit (CPU), memory, input/output devices, storage devices, and the system bus.

Computer architecture encompasses several key aspects, including the instruction set architecture (ISA), which defines the set of instructions that a CPU can execute, the data representation and formats used for computation, and the memory hierarchy that determines how data is stored and accessed at different speeds and capacities.

The goal of computer architecture is to create a system that can execute a wide range of instructions and applications with optimal performance, reliability, and efficiency. It involves decisions about the organization of the CPU, the pathways for data and control signals, the type of memory used, and the communication between various components. Different computer architectures may employ diverse strategies, such as parallel processing, pipelining, and multiple levels of caching, to enhance performance.

History of computer architecture:

Computers have gone through many changes over time. The first generation of computers started around 1940 and since then there have been five generations of computers until 2023. Computers evolved over a long period of time, starting from the 16th century, and continuously improved themselves in terms of speed, accuracy, size, and price to become the modern day computer.

The different phases of this long period are known as computer generations. The first generation of computers was developed from 1940-1956, followed by the second generation from 19561963, the third generation from 1964-1971, the fourth generation from 1971 until the present, and the fifth generation are still being developed.

First Generation of Computers

The first generation used vacuum tube technology and were built between 1946 and 1959. Vacuum tubes were expensive and produced a lot of heat, which made these computers very expensive and only affordable to large organizations. Machine language was the programming language used for these computers, and they could not multitask.

The ENIAC was the first electronic general-purpose computer that used 18,000 vacuum tubes and was built in
1943 for war-related calculations. Examples of the first generation include EDVAC, IBM-650, IBM-701, Manchester Mark 1, Mark 2, etc.

Here are two of the main advantages of firstgeneration:

- The first generation was tough to hack and was quite strong.
- The first generation could perform calculations quickly, in just one-thousandth of a second.

Here are two of the main disadvantages of firstgeneration: They consumed high amounts of energy/electricity.

- They were not portable due to their weight and size.

Second Generation of Computers

The second generation of computers was developed in the late 1950s and 1960s. These computers replaced vacuum tubes with transistors making them smaller, faster and more efficient. This was done as transistors were more reliable than vacuum tubes, required less maintenance and generated less heat.

Second-generation computers were smaller and more portable, making them accessible to a wider audience. Magnetic core memory was also introduced in this generation, which was faster and more reliable. This laid the foundation for further developments, paving the way for the third generation that used integrated circuits.

Here are two of the main advantages of firstgeneration: They provided better speed and improved accuracy.

- Computers developed in this era were smaller, more reliable, and capable of using less power.

Here are two of the main disadvantages of firstgeneration:

- They were only used for specific objectives and required frequent maintenance.
- The second generation of computer used punch cards for input, which required frequent maintenance.

Third Generation of Computers

The third generation of computers emerged between 1964 and 1971. This generation used microchips or integrated circuits, making it possible to create smaller, cheaper, and much faster computers.

The third generation of computers was much faster than previous generations, with computational times reduced from microseconds to nanoseconds. New input devices like the mouse and keyboard were introduced, replacing older methods like punch cards. New functionalities, like multiprogramming and time-sharing, and remote processing, were introduced, allowing for more efficient use of computer resources.

Here are two of the main advantages of first generation:

- The use of integrated circuits made them more reliable.
- Smaller in size and required less space than previous generations.

Here are two of the main disadvantages of first generation:

- Advanced technology was needed to manufacture IC chips.
- Formal training was necessary to operate third-gen computers.

Fourth Generation of Computers

Fourth generation computers were developed in 1972 after third generation that used microprocessors. They used Very Large Scale Integrated (VLSI) circuits, which contained about 5000 transistors capable of performing complex activities and computations.

Fourth generation computers were more adaptable, had more primary storage capacity, were faster and more reliable than previous generations, and were also portable, small, and required less electricity. Intel was the first company to develop a microprocessor used in fourth generation computer.

Fourth generation computers used LSI chip technology and were incredibly powerful but also very small, leading to a societal revolution in the computer industry. This generation had the first supercomputers, used complex programming languages like C, C++, DBASE, etc., and could perform many accurate calculations.

Here are two of the main advantages of first generation:

- Fourth generation computers were smaller and more dependable.
- GUI (Graphics User Interface) technology was used in this generation to provide users with better comfort.

Here are two of the main disadvantages of first generation:

- They use complex VLSI Chips, and VLSI Chip manufacturing requires advanced technology.
- To build these computers, Integrated Circuits (ICs) were required, and to develop those, cutting-edge technology was needed.

Fifth Generation of Computers

The fifth generation of computers emerged after the fourth generation and is still being developed. Computers of fifth generation use artificial intelligence (AI) to perform various tasks. These computers use programming languages such as Python, R, C#, Java, etc., as input methods.

The fifth generation computers employ ULSI technology (Ultra Large Scale Integration), parallel processing, and AI to perform scientific computations and develop AI software. They can perform intricate tasks such as image recognition, human speech interpretation, natural language understanding, etc. Examples of fifth generation include laptops, desktops, notebooks, chromebooks, etc.

Here are two of the main advantages of first generation:

- These computers are lightweight and easy to move around.

- They are easier to repair and parallel processing technology has improved in these computers.

Here are two of the main disadvantages of first generation:
- Using it for spying on people.
- Fear of unemployment due to AI replacing jobs.

Overview of Computer Organization

Computer organization refers to the way a computer's hardware components are arranged and interact to execute instructions. It encompasses the design and structure of the computer's internal architecture, including the central processing unit (CPU), memory, input/output devices, and the system bus. The primary goal of computer organization is to create an efficient, reliable, and scalable system that can execute a wide range of tasks.

At its core, computer organization is concerned with the organization and interconnection of the hardware components to ensure effective communication and coordination. The key components include:

1. **Central Processing Unit (CPU):** Often regarded as the brain of the computer, the CPU is responsible for executing instructions stored in memory. It comprises the arithmetic logic unit (ALU) for performing calculations, the control unit for managing the execution of instructions, and registers for temporary data storage.
2. **Memory:** Memory is where data and program instructions are stored for quick access by the CPU. Computer systems typically have two types of memory - volatile RAM (Random Access Memory) for temporary data storage and non-volatile storage (like hard drives or SSDs) for long-term data storage.
3. **Input/Output (I/O) Devices:** These include peripherals like keyboards, mice, displays, and external storage devices. I/O devices facilitate communication between the computer and the external world.
4. **System Bus:** The system bus is a communication pathway that connects the CPU, memory, and I/O devices, allowing them to exchange data and instructions.

5. **Storage:** In addition to RAM, computers have long-term storage devices like hard drives or SSDs, where data and applications are stored even when the power is turned off.
6. **Cache Memory:** Cache memory is a small, high-speed type of volatile computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications, and data.

Computer organization also considers factors such as data representation, instruction set architecture, and addressing modes. Different types of computer organizations, such as von Neumann architecture and Harvard architecture, offer varying approaches to organizing memory and processing units.

In summary, computer organization is a crucial aspect of computer architecture that focuses on the structure and design of a computer system's components, ensuring efficient communication and execution of instructions. The field continues to evolve with advancements in technology, leading to the development of faster, more reliable, and energy-efficient computing systems.

Memory Hierarchy and cache

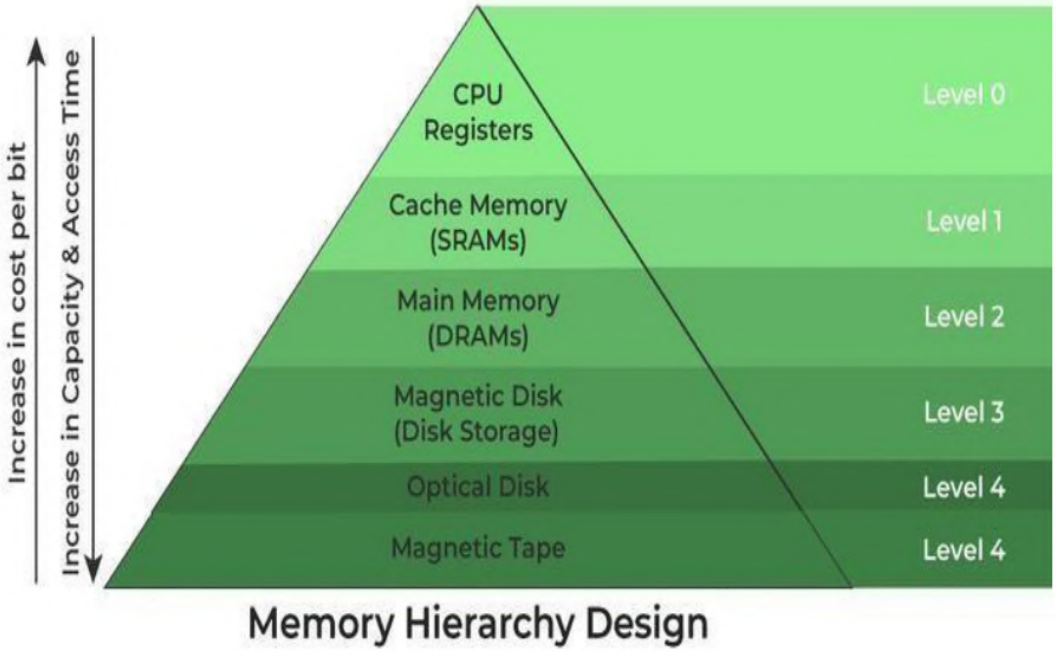
In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references. The figure below clearly demonstrates the different levels of the memory hierarchy.

Why Memory Hierarchy is Required in the System?

Memory Hierarchy is one of the most required things in Computer Memory as it helps in optimizing the memory available in the computer. There are multiple levels present in the memory, each one having a different size, different cost, etc. Some types of memory like cache, and main memory are faster as compared to other types of memory but they are having a little less size and are also costly whereas some memory has a little higher storage value, but they are a little slower. Accessing of data is not similar in all types of memory, some have faster access whereas some have slower access. Types of Memory Hierarchy

This Memory Hierarchy Design is divided into 2 main types:

- **External Memory or Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via an I/O Module.
- **Internal Memory or Primary Memory:** Comprising of Main Memory, Cache Memory &
  CPU registers. This is directly accessible by the processor.



Memory Hierarchy Design

1. Registers

Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.

2. Cache Memory

Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

3. Main Memory

Main memory, also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.

Types of Main Memory

- **Static RAM:** Static RAM stores the binary information in flip flops and information remains valid until power is supplied. It has a faster access time and is used in implementing cache memory.
- **Dynamic RAM:** It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

4. Secondary Storage

Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD), is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

5. Magnetic Disk

Magnetic Disks are simply circular plates that are fabricated with either a metal or a plastic or a magnetized material. The Magnetic disks work at a high speed inside the computer and these are frequently used.

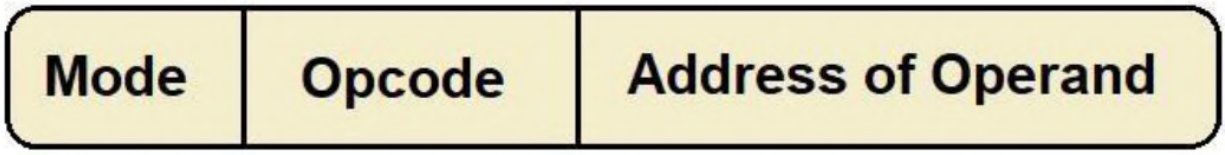
6. Magnetic Tape

Magnetic Tape is simply a magnetic recording device that is covered with a plastic film. It is generally used for the backup of data. In the case of a magnetic tape, the access time for a computer is a little slower and therefore, it requires some amount of time for accessing the strip.

Instruction codes

Instruction codes are bits that instruct the computer to execute a specific operation. An instruction comprises groups called fields. These fields include:
An instruction comprises groups called fields. These fields include:

- The Operation code (Opcode) field determines the process that needs to perform.
- The Address field contains the operand's location, i.e., register or memory location. □ The Mode field specifies how the operand locates.



Structure of an Instruction Code

The instruction code is also known as an instruction set. It is a collection of binary codes. It represents the operations that a computer processor can perform. The structure of an instruction code can vary. It depends on the architecture of the processor but generally consists of the following parts:

- Opcode: The opcode (Operation code) represents the operation that the processor must perform. It might indicate that the instruction is an arithmetic operation such as addition, subtraction, multiplication, or division.

- Operand(s): The operand(s) represents the data that the operation must be performed on. This data can take various forms, depending on the processor's architecture. It might be a register containing a value, a memory address pointing to a location in memory where the data is stored, or a constant value embedded within the instruction itself.

- Addressing mode: The addressing mode represents how the operand(s) can be interpreted. It might indicate that the operand is a direct address in memory, an indirect address (i.e. a memory address stored in a register), or an immediate value (i.e. a constant embedded within the instruction).

- Prefixes or modifiers: Some instruction sets may include additional prefixes or modifiers that modify the behavior of the instruction. For example, they may specify that the operation should be performed only if a specific condition is met or that the instruction should be executed repeatedly until a specific condition is met.

Types of Instruction Code

There are various types of instruction codes. They are classified based on the number of operands, the type of operation performed, and the addressing modes used. The following are some common types of instruction codes:

1. One-operand instructions: These instructions have one operand and perform an operation on that operand. For example, the "neg" instruction in the x86 assembly language negates the value of a single operand.

2. Two-operand instructions: These instructions have two operands and perform an operation involving both. For example, the "add" instruction in x86 assembly language adds two operands together.

3. Three-operand instructions: These instructions have three operands and perform an operation that involves all three operands. For example, the "fma" (fused multiply-add) instruction in some processors multiplies two operands together, adds a third operand, and stores the result in a fourth operand.

4. Data transfer instructions: These instructions move data between memory and registers or between registers. For example, the "mov" instruction in the x86 assembly language moves data from one location to another.

5. Control transfer instructions: These instructions change the flow of program execution by modifying the program counter. For example, the "jmp" instruction in the x86 assembly language jumps to a different location in the program.

6. Arithmetic instructions: These instructions perform mathematical operations on operands. For example, the "add" instruction in x86 assembly language adds two operands together.

7. Logical instructions: These instructions perform logical operations on operands. For example, the "and" instruction in x86 assembly language performs a bitwise AND operation on two operands.

8. Comparison instructions: These instructions compare two operands and set a flag based on the result. For example, the "cmp" instruction in x86 assembly language compares two operands and sets a flag indicating whether they are equal, greater than, or less than.

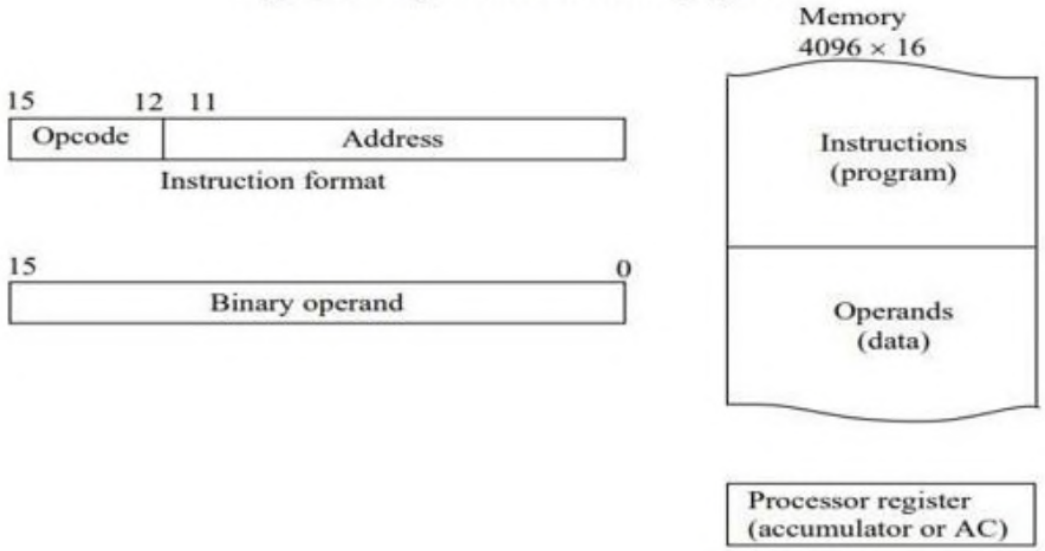
9. Floating-point instructions: These instructions perform arithmetic and other operations on floating-point numbers. For example, the "fadd" instruction in the x86 assembly language adds two floating-point numbers together.

Stored Program Organization

SPO is a fundamental concept in computer architecture that revolutionized the way computers operate. In a system following the Stored Program Organization, both data and instructions are stored in the computer's memory in the same format. This means that a computer can manipulate its own program instructions just like any other data, enabling a high degree of flexibility and programmability. The cornerstone of SPO is the von Neumann architecture, named after mathematician and computer scientist John von Neumann, who formalized the idea. In this architecture, the CPU fetches instructions and data from the same memory, allowing for seamless interaction between the two. This design principle has become the standard for modern computers, fostering the development of versatile and programmable machines that can execute a wide range of applications by simply changing the stored program in memory. The Stored Program Organization has played a pivotal role in the evolution of computing systems, enabling the development of powerful and adaptable computers that have become integral to various aspects of modern life.

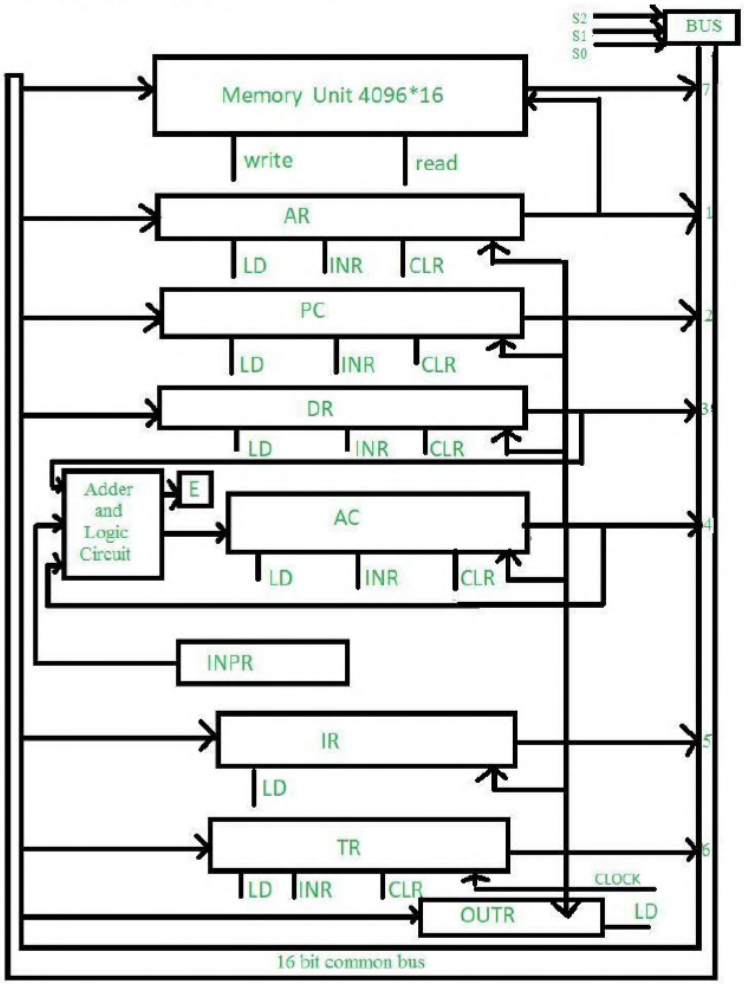
Figure 1.1 depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$. If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated op code) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand. The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory. It then executes the operation specified by the operation code.

Figure 1.1 Organization for stored program



Common bus system

A basic computer has 8 registers, memory unit and a control unit. The diagram of the common bus system is as shown below.



Connections:

The outputs of all the registers except the OUTR (output register) are connected to the common bus. The output selected depends upon the binary value of variables S2, S1 and S0. The lines from common bus are connected to the inputs of the registers and memory. A register receives the information from the bus when its LD (load) input is activated while in case of memory the Write input must be enabled to receive the information. The contents of memory are placed onto the bus when its Read input is activated.

Various Registers:

4 registers DR, AC, IR and TR have 16 bits and 2 registers AR and PC have 12 bits. The INPR and OUTR have 8 bits each. The INPR receives character from input device and delivers it to the AC while the OUTR receives character from AC and transfers it to the output device. 5 registers have 3 control inputs LD (load), INR (increment) and CLR (clear). These types of registers are similar to a binary counter.

Abbreviation	**Register name**
OUTR	Output register
Abbreviation	**Register name**
TR	Temporary register
IR	Instruction register
INPR	Input register
AC	Accumulator
DR	Data register
PC	Program counter
AR	Address register

Adder and logic circuit:

The adder and logic circuit provides the 16 inputs of AC. This circuit has 3 sets of inputs. One set comes from the outputs of AC which implements register micro operations. The other set comes from the DR (data register) which are used to perform arithmetic and logic micro operations. The result of these operations is sent to AC while the end around carry is stored in E as shown in diagram. The third set of inputs is from INPR.

Note:
The content of any register can be placed on the common bus and an operation can be performed in the adder and logic circuit during the same clock cycle.

Instruction set

An instruction set is a group of commands for a central processing unit (CPU) in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations. All CPUs have instruction sets that enable commands directing the CPU to switch the relevant transistors. The instructions tell the CPU to perform tasks. Some instructions are simple read, write and move commands that direct data to different hardware elements. The instructions are made up of a specific number of bits. For instance, The CPU's instructions might be 8 bits, where the first 4 bits make up the operation code that tells the computer what to do. The next 4 bits are the operand, which tells the computer the data that should be used. The length of an instruction set can vary from as few as 4 bits to many hundreds. Different instructions in some instruction set architectures (ISAs) have different lengths. Other ISAs have fixed-length instructions.

The following are three main ways instruction set commands are used:

1. **Data handling and memory management.** Instruction set commands are used when setting a register to a specific value, copying data from memory to a register or vice versa, and reading and writing data.

2. **Arithmetic and logic operations and activities.** These commands include add, subtract, multiply, divide and compare, which examines values in two registers to see if one is greater or less than the other.
3. **Control flow activities.** One example is branch, which instructs the system to go to another location and execute commands there. Another is jump, which moves to a specific memory location or address.

Instruction types

A computer instruction refers to a binary code that controls how a computer performs micro-operations in a series. They, together with the information, are saved in the memory. Every computer has its own set of instructions. Operation codes or Opcodes and Addresses are the two elements that they are divided into.

A computer's instructions can be any length and have any number of addresses. The arrangement of a computer's registers determines the different address fields in the instruction format. The instruction can be classified as three, two, and one address instruction or zero address instruction, depending on the number of address fields.

Three Address Instructions

A three-address instruction has the following

general format: source 1 operation, source 2

operation, source 3 operation, destination

ADD X, Y, Z

Here, X, Y, and Z seem to be the three variables that are each assigned to a distinct memory location. The operation implemented on operands is 'ADD.' The source operands are 'X' and 'Y,' while the destination operand is 'Z.'

In order to determine the three operands, bits are required. To determine one operand, n bits are required (one memory address). In the same way, 3n bits are required to define three operands (or three memory addresses). To identify the ADD operation, bits are also required.

Two Address Instructions

A two-address instruction has the following

general format: source and destination of the

operation

ADD X, Y

Here X and Y are the two variables that have been assigned to a specific memory address. The operation performed on the operands is 'ADD.' This command combines the contents of variables X and Y and stores the result in variable Y. The source operand is 'A,' while 'B' is used as both a source and a destination operand.

The two operands must be determined using bits. To define one operand, n bits are required (one memory address). To determine two operands, 2n bits are required (two memory addresses). The ADD operation also necessitates the use of bits.

One Address Instructions

One address instruction has the following general format:

operation source

INCLUDE X

Here X refers to the variable that has access to a specific memory region. The operation performed on operand A is 'ADD.' This instruction adds the value of variable A to the accumulator and then saves the result inside the accumulator by restoring the accumulator's contents.

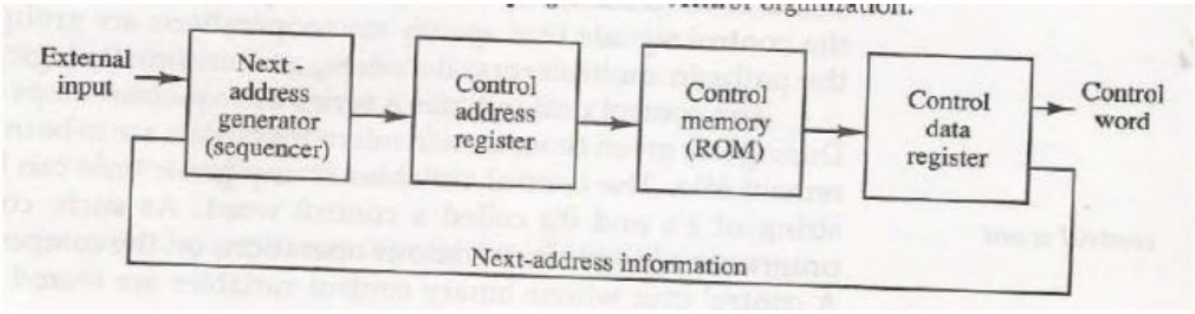
Zero Address Instructions

In zero address instructions, the positions of the operands are implicitly represented. These instructions use a structure called a pushdown stack to hold operands.

Unit -5 **Design of Microprogram Control Unit**

- The control function that specifies a microoperation is called as control variable.
- When control variable is in one binary state, the corresponding microoperation is executed. For the other binary state the state of registers does not change.
- The active state of a control variable may be either 1 state or the 0 state, depending on the application
- For bus-organized systems the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.
- **Control Word**: The control variables at any given time can be represented by a string of 1's and 0's called a **control word**.
- All control words can be programmed to perform various operations on the components of the system.
- **Microprogram control unit:** A control unit whose binary control variables are stored in memory is called a microprogram control unit.
- The control word in control memory contains within it a microinstruction.
- The microinstruction specifies one or more micro-operations for the system.
- A sequence of microinstructions constitutes a microprogram.
- The control unit consists of control memory used to store the microprogram.
- *Control memory is a permanent i.e., read only memory (ROM).*

- The general configuration of a micro-programmed control unit organization is shown as block diagram below



- The control memory is ROM so all control information is permanently stored.
- The control memory address register (CAR) specifies the address of the microinstruction and the control data register (CDR) holds the microinstruction read from memory.
- The next address generator is sometimes called a microprogram sequencer. It is used to generate the next micro instruction address.
- The location of the next microinstruction may be the one next in sequence or it may be located somewhere else in the control memory.
- So it is necessary to use some bits of the present microinstruction to control the generation of the address of the microinstruction

- Sometimes the next address may also be a function of external input conditions
- The control data register holds the present microinstruction while next address is computed and read from memory. The data register is times called a pipeline register
- A computer with a microprogrammed control unit will have two separate memories: a main memory and a control memory
- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations
- These microinstructions generate the microoperations to:

- fetch the instruction from main memory
- evaluate the effective address
- execute the operation
- return control to the fetch phase for the next instruction

Control Adress Register

In a microprocessor, the Control Address Register (CAR) is a component that helps manage and coordinate various tasks within the processor. It holds the address or location of the next

instruction or data that the processor needs to access or manipulate.

1. **Program Counter (PC):**

- Keeps track of the memory address of the next instruction to be fetched and executed.
- Guides the processor through the sequential execution of instructions.

2. **Instruction Register (IR):**

- Holds the current instruction being executed.
- Facilitates the decoding and execution of the instruction.

3. **Memory Address Register (MAR):**

- Contains the memory address for read or write operations.
- Specifies the location in memory where data is to be fetched from or stored.

4. **Memory Buffer Register (MBR):**

- Holds data temporarily during memory read or write operations.
- Acts as a buffer between the processor and memory.

5. Index Register:

- Used for indexing operations in certain addressing modes.
- Enhances the flexibility of addressing by allowing indirect addressing through an offset.

6. **Stack Pointer (SP) or Stack Register**:

- Manages the stack in a microprocessor.
- Keeps track of the top of the stack, facilitating push and pop operations.

Microprogram Sequencer

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a microprogram sequencer.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.
- The block diagram of the microprogram sequencer is shown in below figure.
- The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it

- There are two multiplexers in the circuit.
  - The first multiplexer selects an address from one of four sources and routes it into control address register CAR.
  - The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit
- The output from CAR provides the address for the control memory.
- The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR.
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.
- If the bit selected is equal to 1, the T variable is equal to 1; otherwise, it is equal to 0.
- The T value together with two bits from the BR (branch) field goes to an input logic circuit.
- The input logic in a particular sequencer will determine the type of operations that are available in the unit.
- The other three inputs to multiplexer come from
  - The address field of the present microinstruction
  - From the out of SBR
  - From an external source that maps the instruction

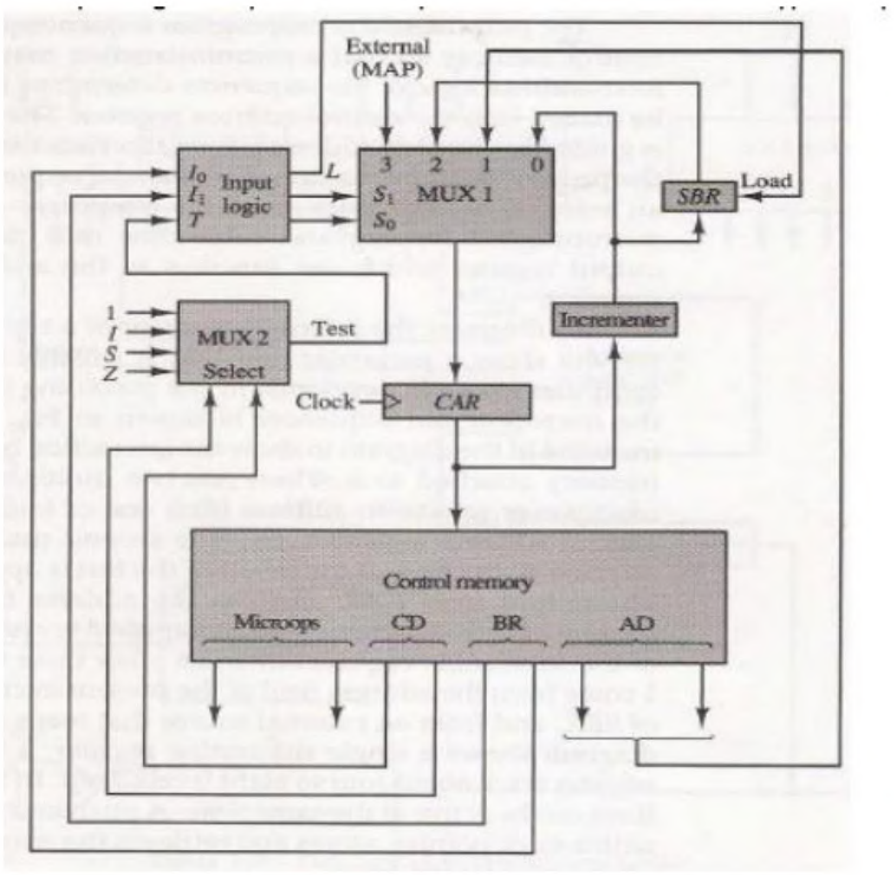


Fig: Microprogram Sequence for control memory

- The input logic circuit in above figure has three inputs I0, I1, and T, and three outputs, S0, S1, and L.
- Variables S0 and S1 select one of the source addresses for CAR. Variable L enables the load input in SBR.
- The binary values of the selection variables determine the path in the multiplexer
- For example, with S1,S0 = 10, multiplexer input number 2 is selected and establishes transfer path from SBR to CAR

The truth table for the input logic circuit is shown in Table below.

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

- Inputs I1 and I0 are identical to the bit values in the BR field.
- The bit values for S1 and S0 are determined from the stated function and the path in the multiplexer that establishes the required transfer
- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).
- The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I'_1 T$$
$$L = I'_1 T I_0$$

Address Sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a ***routine***.
- To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step -1

- An initial address is loaded into the control address register when power is turned on in the computer.
- This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.
- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- When the effective address computation routine is completed, the address of the operand is available in the memory address register.

Step-3:

- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- Each instruction has its own micro-program routine stored in a given location of control memory.
- The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a ***mapping*** process.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.

Step-4:

- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
- Micro-programs that employ subroutines will require an external register for storing the return address.
- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

CONDITIONAL BRANCH INSTRUCTIONS

In microprocessor programming, conditional branches are instructions that alter the flow of program execution based on certain conditions. These instructions allow the program to make decisions and execute different sequences of instructions depending on whether a specific condition is true or false.

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 ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
Signed compare conditions (A - B)		
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this type of branch are the bits in the operation code part of the instruction.
- For example, a computer with a simple instruction format as shown in figure 4.3 has an operation code of four bits which can specify up to 16 distinct instructions.
- Assume further that the control memory has 128 words, requiring an address of seven bits.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure 4.3.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions
- If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

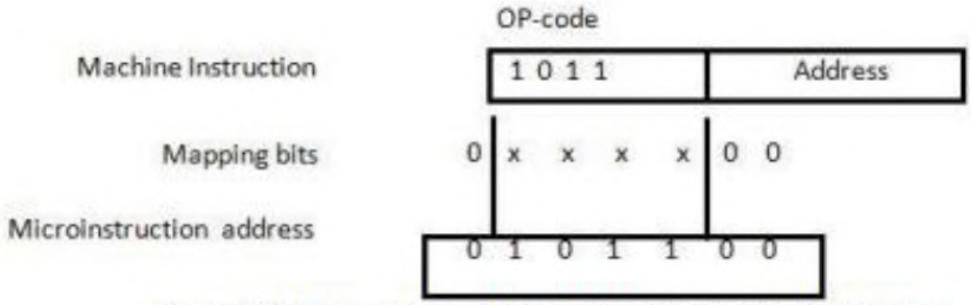


Figure 4.3: Mapping from instruction code to microinstruction address

- One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function.
- The contents of the mapping ROM give the bits for the control address register.
- In this way the microprogram routine that executes the instruction can be placed in any desired location in control memory.
- The mapping concept provides flexibility for adding instructions for control memory as the need arises.
- **The block diagram of the computer is shown in Figure 4.4. It consists of**

1. Two memory units:

  - Main memory -> for storing instructions and data, and
  - Control memory -> for storing the microprogram.

2. Six Registers:

  - Processor unit register: AC(accumulator),PC(Program* Counter), AR(Address Register), DR(Data Register)
  - Control unit register: CAR (Control Address Register), SBR(Subroutine Register)

3. Multiplexers:

- The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

4. ALU:

- The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.

1. DR can receive information from AC, PC, or memory.
2. AR can receive information from PC or DR.
3. PC can receive information only from AR.
4. Input data written to memory come from DR, and data read from memory can go only to DR

Computer Hardware Configuration

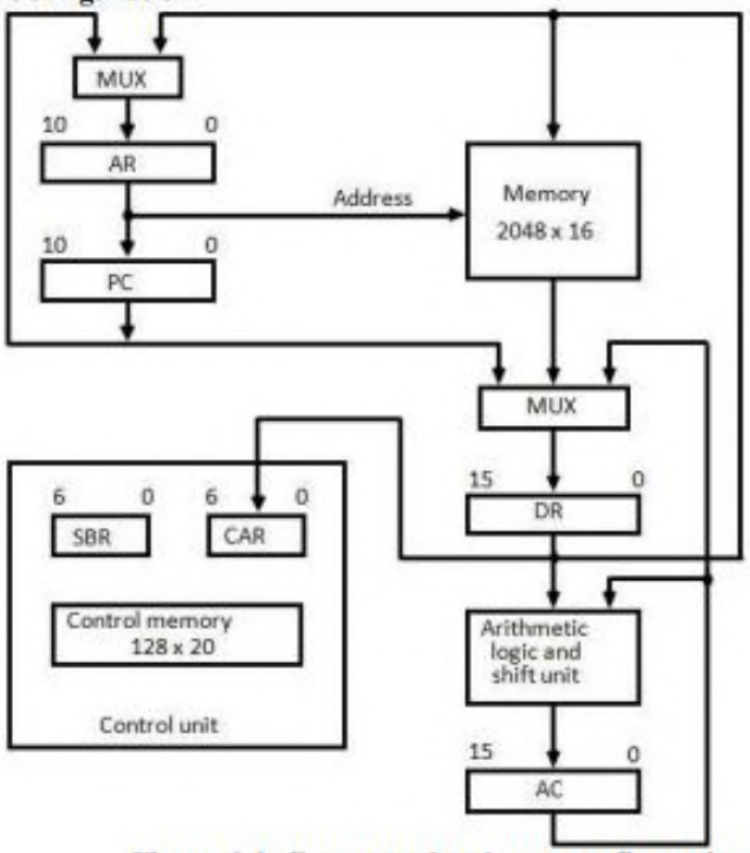


Figure 4.4: Computer hardware configuration

In the context of microprocessors like the 8085, a subroutine is a named block of code that performs a specific task and can be called from different parts of a program. The use of subroutines helps in modular programming, making code more readable, reusable, and easier to maintain.

In this example:

1. The program starts at the START label.
2. It loads a value into register B.
3. Then, it calls the MY_SUBROUTINE subroutine using the CALL instruction.
4. The subroutine adds the value in register C to the accumulator (ADD C).
5. Finally, the RET instruction is used to return from the subroutine, and the main program continues executing after the CALL instruction.

Microinstruction Format

- The microinstruction format for the control memory is shown in figure 4.5. The 20 bits of the microinstruction are divided into four functional parts as follows:
- The three fields F1, F2, and F3 specify microoperations for the computer.
- The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations
- The CD field selects status bit conditions.
- The BR field specifies the type of branch to be used.

- The AD field contains a branch address. The address field is seven bits wide, since the control memory has 128 = 27 words.

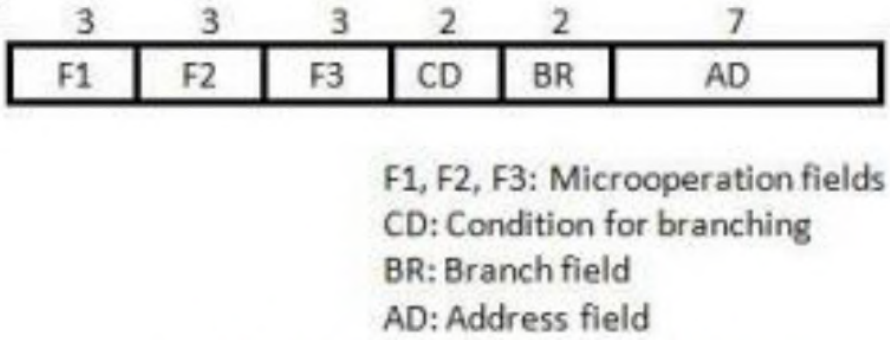


Figure 4.5: Microinstruction Format

- As an example, a microinstruction can specify two simultaneous microoperations from F2 and F3 and none from F1.

DR □M[AR] with F2 = 100
PC □ PC + 1 with F3 = 101

- The nine bits of the microoperation fields will then be 000 100 101.
- The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 4.1.

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

Table 4.1: Condition Field

- The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction shown in Table 4.2

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

Table 4.2: Branch Field

Symbolic Microinstruction.

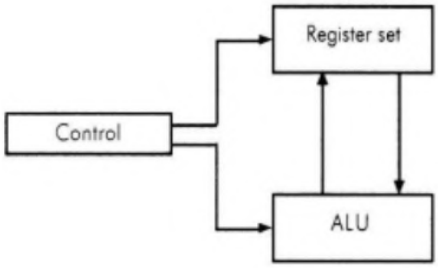
- Each line of the assembly language microprogram defines a symbolic microinstruction.
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. The fields specify the following Table 4.3

1.	Label	The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2.	Microoperations	It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.
3.	CD	The CD field has one of the letters U, I, S, or Z.
4.	BR	The BR field contains one of the four symbols defined in Table 5.2.
5.	AD	The AD field specifies a value for the address field of the microinstruction in one of three possible ways: i. With a symbolic address, this must also appear as a label. ii. With the symbol NEXT to designate the next address in sequence. iii. When the BR field contains a RET or MAP symbol, the AD field is left empty and is converted to seven zeros by the assembler.

Table 4.3: Symbolic Microinstruction

General Register Organization : ·

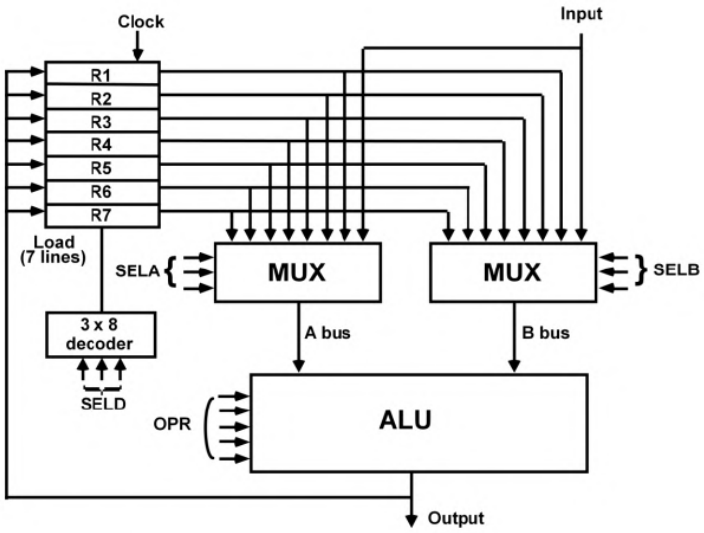
The Central Processing Unit (CPU) is called the brain of the computer that performs data processing operations. Figure 3.1 shows the three major parts of CPU



Intermediate data is stored in the register set during the execution of the instructions. The microoperations
required for executing the instructions are performed by the arithmetic logic unit whereas the control unit
takes care of transfer of information among the registers and guides the ALU. The control unit services
the transfer of information among the registers and instructs the ALU about which operation is to be
performed. The computer instruction set is meant for providing the specifications for the design of the
CPU. The design of the CPU largely, involves choosing the hardware for implementing the machine instructions.

The need for memory locations arises for storing pointers, counters, return address, temporary results and partial products. Memory access consumes the most of the time off an operation in a computer. It is more convenient and more efficient to store these intermediate values in processor registers.

A common bus system is employed to contact registers that are included in the CPU in a large number. Communications between registers is not only for direct data transfer but also for performing various micro-operations. A bus organization for such CPU register shown in Figure 3.2, is connected to two multiplexers (MUX) to form two buses A and B. The selected lines in each multiplexers select one register of the input data for the particular bus.



OPERATION OF CONTROL UNIT:

The control unit directs the information flow through ALU by:

- Selecting various Components in the system

- Selecting the Function of ALU

Example: R1 <- R2 + R3 1] MUX A selector

1. (SELA): BUS A ← R2 [2] MUX B selector
2. (SELB): BUS B ← R3 [3] ALU operation selector
3. (OPR): ALU to ADD [4] Decoder destination selector
4. (SELD): R1 ← Out Bus

Control Word

3	3	3	5
SELA	SELB	SELD	OPR

Encoding of register selection fields

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

Encoding of ALU operations

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:

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
- The stack in digital computers is essentially a memory unit with an address register that can count only. The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
- The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

Register stack:

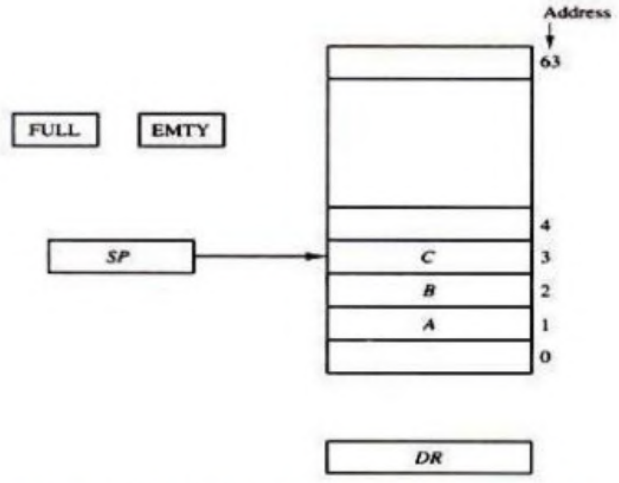


Figure 5.1: Block diagram of a 64-word stack

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite
- number of memory words or registers. Figure shows the organization of a 64- word register stack.
- The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack.
- Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.
- To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2
- To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.
- In a 64-word stack, the stack pointer contains 6 bits because 26 = 64.
- Similarly, when 000000 is decremented by 1, the result is 111111. The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.
- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 are incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.
- DR is the data register that holds the binary data to be written into or read out of the stack.

PUSH:

- If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences of microoperations:

SP ← SP + 1	Increment stack pointer
M [SP] ← DR	WRITE ITEM ON TOP OF THE STACK
IF (SP = 0) then (FULL ← 1)	Check is stack is full
EMTY ← 0	Mark the stack not empty

- The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of the stack.
- SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.
- Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.
- The first item stored in the stack is at address 1. The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to 1. This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.

POP:

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequences of microoperations:

DR ← M [SP]	Read item on top of the stack
SP ← SP - 1	Decrement stack pointer
IF (SP = 0) then (EMTY ← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full

- The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1.
- This condition is reached if the item read was in location 1.
- Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is changes to 111111, which is equivalent to decimal 63.
- In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1.

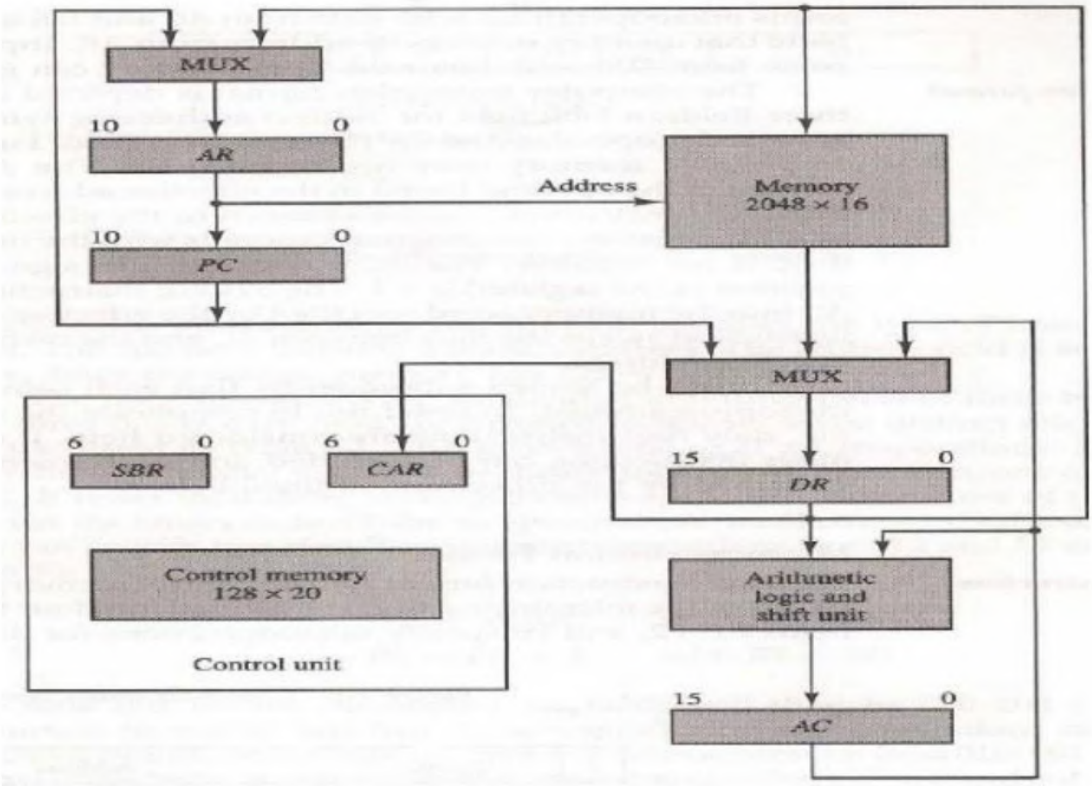
Instruction formats

A standard machine which can be directly decoded and executed by the CPU is called instruction format.

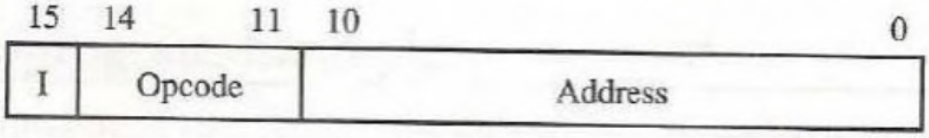
- The process of code generation for the control memory is called microprogramming
- The block diagram of the computer configuration is shown in below figure
- Two memory units:
  - Main memory – stores instructions and data
  - Control memory – stores microprogram
- Four processor registers
  - Program counter – PC
  - Address register – AR
  - Data register – DR

- Accumulator register – AC

- Two control unit registers
  - Control address register – CAR
  - Subroutine register – SBR
- Transfer of information among registers in the processor is through MUXs rather than a bus



- The computer instruction format is shown in below figure



(a) Instruction format

- Three fields for an instruction
  - 1-bit field for indirect addressing

- 4-bit opcode
- 11-bit address field

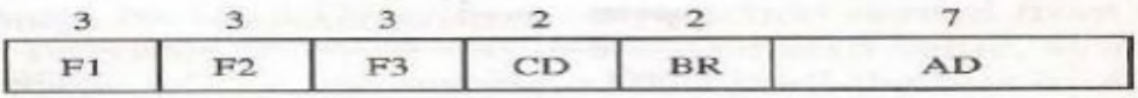
- The example will only consider the following 4 of the possible 16 memory instructions

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

- The microinstruction format for the control memory is shown in below figure.



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

- The microinstruction format is composed of 20 bits with
- four parts to it

- Three fields F1, F2, and F3 specify microoperations for the computer [3 bits each]

- ➢ The CD field selects status bit conditions [2 bits]
- ➢ The BR field specifies the type of branch to be used [2 bits]
- ➢ The AD field contains a branch address [7 bits]

- Each of the three microoperation fields can specify one of seven possibilities
- No more than three microoperations can be chosen for a microinstruction
- If fewer than three are needed, the code 000 = NOP
- The three bits in each field are encoded to specify seven distinct microoperations listed in below table

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	

- Five letters to specify a transfer-type microoperation

➢ First two designate the source register

➢ Third is a 'T'

➢ Last two designate the destination register

AC ← DR F1 = 100 = DRTAC

- The condition field (CD) is two bits to specify four status bit conditions shown below.

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*

- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next microinstruction.

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

- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next micro instruction. Each line of an assembly language microprogram defines a symbolic microinstruction and is divided into five parts

➢ The label field may be empty or it may specify a symbolic address. Terminate with a colon (:)

- The microoperations field consists of 1-3 symbols, separated by commas. Only one symbol from each field. If NOP, then translated to 9 zeros .
- The condition field specifies one of the four conditions
- The branch field has one of the four branch symbols
- The address field has three formats

  - A symbolic address – must also be a label
  - The symbol NEXT to designate the next address in sequence
  - Empty if the branch field is RET or MAP and is converted to 7 zeros

- The symbol ORG defines the first address of a microprogram routine
- ORG 64 – places first microinstruction at control memory 1000000.

RISC Pipeline

1. To use an efficient instruction pipeline

- To implement an instruction pipeline using a small number of suboperations, with each being executed in one clock cycle.
- Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection
- Therefore, the instruction pipeline can be implemented with two or three segments.
  - One segment fetches the instruction from program memory
  - The other segment executes the instruction in the ALU
  - Third segment may be used to store the result of the ALU

operation in a destination register

2. **The data transfer instructions in RISC are limited to load and store instructions**

- These instructions use register indirect addressing. They usually need three or four stages in the pipeline.
- To prevent conflicts between a memory access to fetch an instruction
- and to load or store an operand, most RISC machines use two separate buses with two memories
- Cache memory: operate at the same speed as the CPU clock

3. **One of the major advantages of RISC is its ability to execute instructions atthe rate of one per clock cycle**
    - In effect, it is to start each instruction with each clock cycle and to pipeline the processor to achieve the goal of single-cycle instruction execution
    - RISC can achieve pipeline segments, requiring just one clock cycle
4. **Compiler supported that translates the high-level language program into machine language program**
    - Instead of designing hardware to handle the difficulties associated with data conflicts and branch penalties.
    - RISC processors rely on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

Example: Three-Segment Instruction Pipeline

- Thee are three types of instructions:
    - The data manipulation instructions: operate on data in processor registers

- The data transfer instructions:
- The program control instructions:

- The *control section* fetches the instruction from program memory into an instruction register.
  - The instruction is decoded at the same time that the registers needed for the execution of the instruction are selected.
- The processor unit consists of a number of registers and an arithmetic logic unit (ALU).
- A data memory is used to load or store the data from a selected register in the register file.
- The instruction cycle can be divided into three suboperations and implemented in three segments:
  - I: Instruction fetch
    - Fetches the instruction from program memory

  - A: ALU operation
    - The instruction is decoded and an ALU operation is performed.
    - It performs an operation for a data manipulation instruction.
    - It evaluates the effective address for a load or store instruction.
    - It calculates the branch address for a program control instruction.
  - E: Execute instruction
    - Directs the output of the ALU to one of three destinations, depending on the decoded instruction.
    - It transfers the result of the ALU operation into a destination register in the register file.
    - It transfers the effective address to a data memory for loading or storing.
    - It transfers the branch address to the program counter.

RISC

- It is a reduced instruction set computer
- It has Reduced instruction
- It focuses on software
- Code size is large
- The execution time of RISC is very short

- An instruction can fit in one word
- It required register
- Transistors are used for more register

Disadvantage :

- Increased code size
- Compiler Dependency
- Learning Curve
- Large Number of register management
- Potential for increased cache usage
- Not Ideal for all workloads
- Dependency on compiler Quality

CISC

- It is a Complex instruction set computer
- It has complex instruction
- It focuses on hardware
- Code size is small
- The execution time of CISC is longer
- Instruction are larger than the size of one word
- It access memory directly
- Transistors are used for storing complex instruction

Pipeline and Vector Processing

Pipelining

1) Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments

2) The overlapping of computation is made possible by associating a register with each segment in the pipeline.

3) The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

4) Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit.
   - The register holds the data
   - The combinational circuit performs the suboperation in the particular segment

5) A clock is applied to all registers after enough time has elapsed to perform all segment activity

6) The pipeline organization will be demonstrated by means of a simple example.
   - To perform the combined multiply and add operations with a stream of numbers : Ai * Bi + Ci for i = 1, 2, 3, ..., 7

7) Each suboperation is to be implemented in a segment within a pipeline.

$R1 \leftarrow A_i, R2 \leftarrow B_i$ Input A_i and B_i
$R3 \leftarrow R1 * R2, R4 \leftarrow C_i$ Multiply and input C_i
$R5 \leftarrow R3 + R4$ Add C_i to product

8) Each segment has one or two registers and a combinational circuit as shown in Fig. 9-2

9) The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table 4-1.

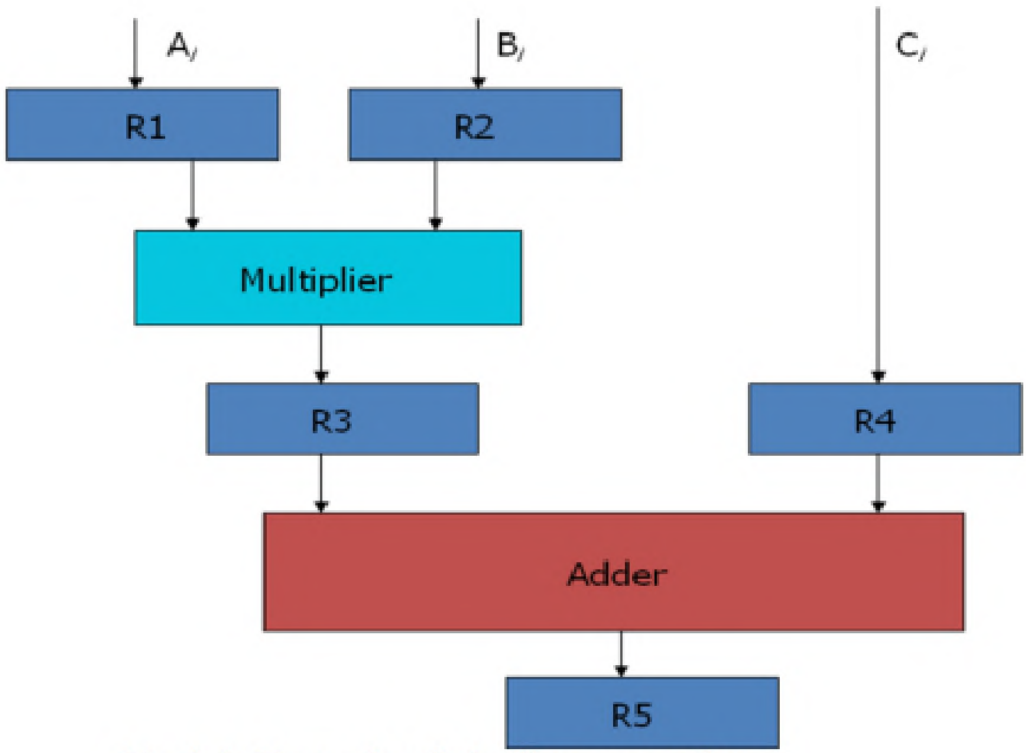


Fig 4-1: Example of pipeline processing

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	--	--	--
2	A_2	B_2	A_1*B_1	C_1	--
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1+C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2+C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3+C_3$
6	A_6	B_6	A_5*B_5	C_5	$A_4*B_4+C_4$
7	A_7	B_7	A_6*B_6	C_6	$A_5*B_5+C_5$
8	--	--	A_7*B_7	C_7	$A_6*B_6+C_6$
9	--	--	--	--	$A_7*B_7+C_7$

Table 4-1: Content of Registers in Pipeline Example

General Considerations

- Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor
- The general structure of a four-segment pipeline is illustrated in Fig. 4-2.
- We define a task as the total operation performed going through all the segments in the pipeline
- The behavior of a pipeline can be illustrated with a space-time diagram
    - It shows the segment utilization as a function of time.

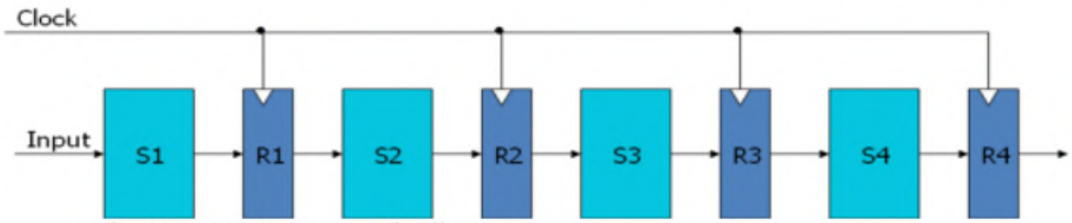


Fig 4-2: Four Segment Pipeline

- The space-time diagram of a four-segment pipeline is demonstrated in Fig. 4-3
- Where a k-segment pipeline with a clock cycle time t_p is used to execute n tasks
    - The first task T1 requires a time equal to kt_p to complete its operation
    - The remaining n-1 tasks will be completed after a time equal to $(n-1)t_p$.

- Therefore, to complete n tasks using a k-segment pipeline requires k+(n-1)clock cycles.

- Consider a non pipeline unit that performs the same operation and takes a time equal to t_n to complete each task.
- The total time required for n tasks is nt_n.

Fig 4-3: Space-time diagram for pipeline

- The speedup of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio S = ntn/(k+n-1)tp.
- If n becomes much larger than k-1, the speedup becomes S = t_n/t_n
- If we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuits, i.e., tn = ktp, the speedup reduces to S=ktp/tp=k

- This shows that the theoretical maximum speed up that a pipeline can provide is k, where k is the number of segments in the pipeline

- To duplicate the theoretical speed advantage of a pipeline process by means of multiple functional units, it is necessary to construct k identical units that will be operating in parallel
- This is illustrated in Fig. 4-4, where four identical circuits are connected in parallel

- Instead of operating with the input data in sequence as in a pipeline, the parallel circuits accept four input data items simultaneously and perform four tasks at the same time.

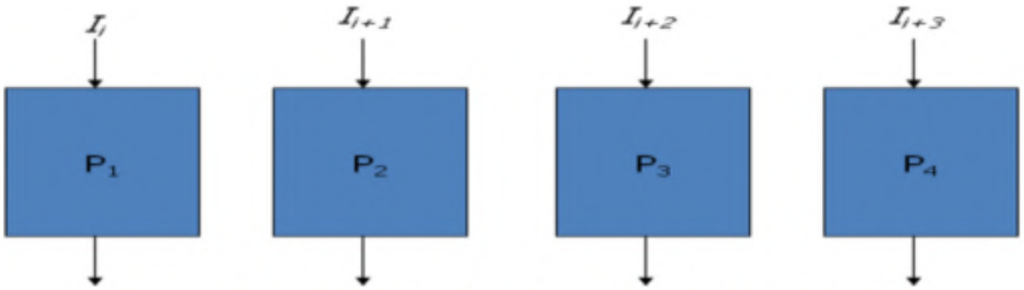


Fig 4-4: Multiple functional units in parallel

- There are various reasons why the pipeline cannot operate at its maximum theoretical rate.
  - Different segments may take different times to complete their sub operation.
  - It is not always correct to assume that a non pipe circuit has the same time delay as that of an equivalent pipeline circuit.

- There are two areas of computer design where the pipeline organization is applicable.
  - Arithmetic pipeline
  - Instruction pipeline

Parallel Processing

1) Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system
2) The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time

3) The amount of hardware increases with parallel processing, and with it, the cost of the system increases
4) Parallel processing can be viewed from various levels of complexity
   - *At the lowest level, we distinguish between parallel and serial operations by the type of registers used. e.g. shift registers and registers with parallel load.*
   - *At a higher level, it can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously*
5) Fig. 4-5 shows one possible way of separating the execution unit into eight functional units operating in parallel

- *A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.*

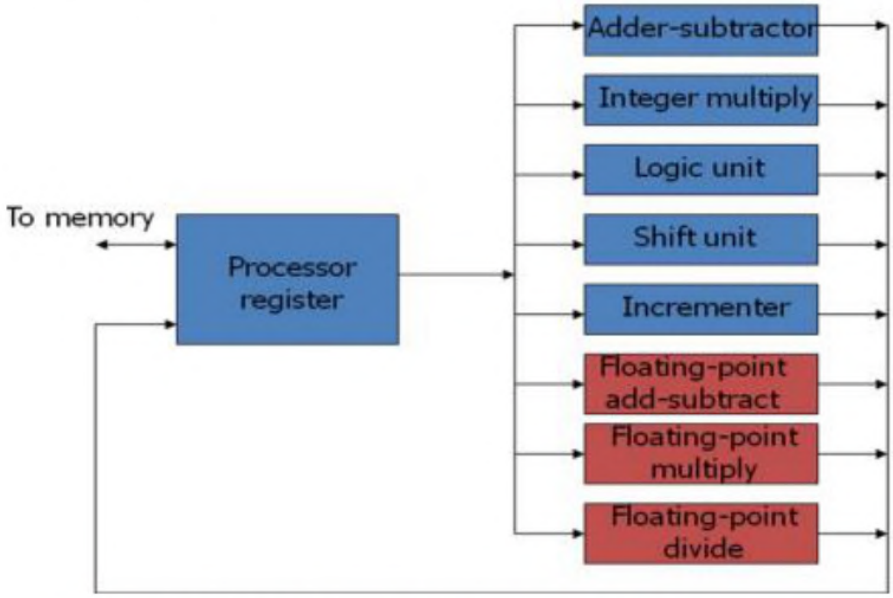


Fig 4-5: Processor with multiple functional units

6) There are a variety of ways that parallel processing can be classified.

- Internal organization of the processors
- Interconnection structure between processors
- The flow of information through the system

7) M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data stream (SIMD)
- Multiple instruction stream, single data stream (MISD)

- Multiple instruction stream, multiple data stream (MIMD)

SISD

- One instruction stream is processed at a time
- Only one set of data is operated on in each instruction.
- Found in early and simpler computers.
- No inherent parallelism.

SIMD

- One instruction stream is processed at multiple time
- Each processing unit operates on a different set of data simultaneously.
- Parallelism achieved by executing the same instruction across multiple data sets.
- Examples include GPUs and some specialized processors

MISD

- Multiple instruction stream is processed at single time
- Each instruction stream provides a different set of operations on the same data.
- Rare in practical implementations due to complexity and limited benefits.
- Theoretical concept explored for fault tolerance and redundancy.

MIMD

- Multiple instruction stream is pr0ocessed at multiple time
- Each processing unit operates on its own set of data.
- Increased throughput and potential for better scalability.
- Allows for a variety of tasks to be performed simultaneously.
- Examples include clusters, grids, and many-core processors.

- Enables true parallel processing with diverse tasks executed concurrently.

---------- ***Arithmetic Pipeline***----------

- Pipeline arithmetic units are usually found in very high speed computers
  - Floating–point operations, multiplication of fixed-point numbers, and similar computations in scientific problem
- Floating–point operations are easily decomposed into sub operations.
- An example of a pipeline unit for floating-point addition and subtraction is showed in the following:
  - The inputs to the floating-point adder pipeline are two normalized floating-point binary number

    $X = A \times 2^a$

    $Y = B \times 2^b$
  - A and B are two fractions that represent the mantissas
  - a and b are the exponents
- The floating-point addition and subtraction can be performed in four segments, as shown in Fig. 4-6.
- The suboperations that are performed in the four segments are:
  - *Compare the exponents*
    - The larger exponent is chosen as the exponent of the result.
  - *Align the mantissas*
    - The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right.
  - *Add or subtract the mantissas*
  - *Normalize the result*
    - When an overflow occurs, the mantissa of the sum or difference is shifted right and the exponent incremented by one.
    - If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

- The following numerical example may clarify the suboperations performed in each segment.
- The comparator, shift, adder, subtractor, incrementer, and decrementer in the floating-point pipeline are implemented with combinational circuits.
- Suppose that the time delays of the four segments are t_1=60ns, t_2=70ns, t_3=100ns, t_4=80ns, and the interface registers have a delay of t_r=10ns
  - Pipeline floating-point arithmetic delay: $t_p=t_3+t_r$=110ns
  - Nonpipeline floating-point arithmetic delay: $t_n=t_1+t_2+t_3+t_4+t_r$=320ns
  - Speedup: 320/110=2.9

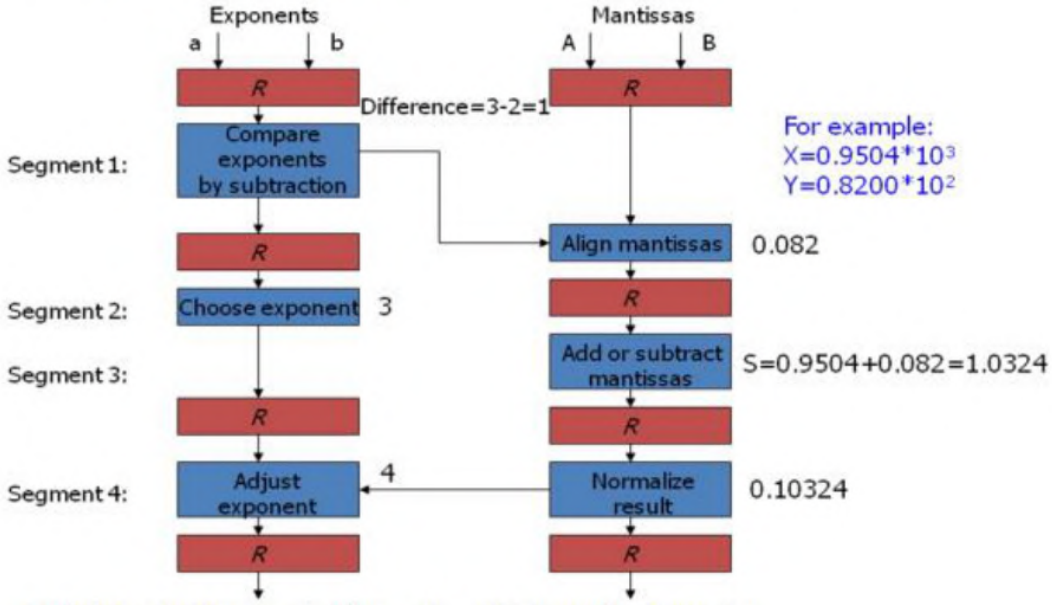


Fig 4-6: Pipeline for floating point addition and subtraction

----------Instruction Pipeline----------

- Pipeline processing can occur not only in the *data stream* but in the *instruction* as well.
- Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a *two-segment* pipeline.
- Computers with complex instructions require other phases in addition to above phases to process an instruction completely.
- In the most general case, the computer needs to process each instruction with the following sequence of steps.
  - Fetch the instruction from memory.
  - Decode the instruction.

- Calculate the effective address.
- Fetch the operands from memory.
- Execute the instruction.
- Store the result in the proper place.

- There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate.
  - Different segments may take different times to operate on the incoming information.
  - Some segments are skipped for certain operations.
  - Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

Example: Four-Segment Instruction Pipeline

- Assume that:
  - The decoding of the instruction can be combined with the calculation of the effective address into one segment.
  - The instruction execution and storing of the result can be combined into one segment.
- Fig 4-7 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.
  - Thus up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.
- An instruction in the sequence may be causes a branch out of normal sequence.

  - In that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted.
  - Similarly, an interrupt request will cause the pipeline to empty and start again from a new address value.

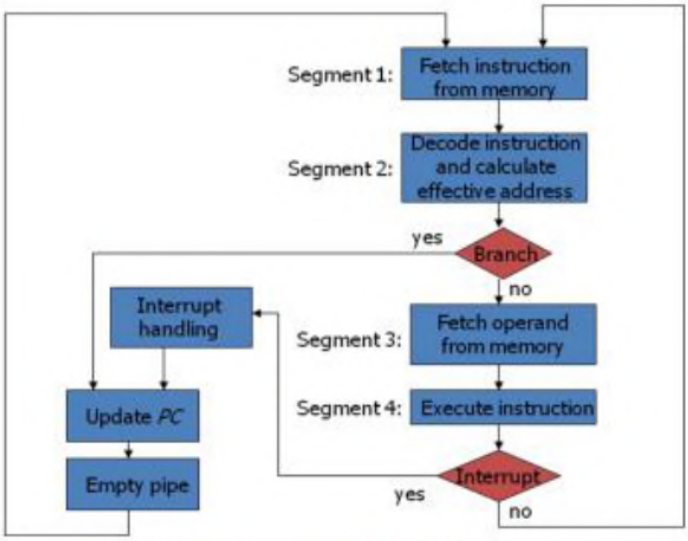


Fig 4-7: Four-segment CPU pipeline

- Fig. 9-8 shows the operation of the instruction pipeline.

	Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
	1	FI	DA	FO	EX									
Instruction:	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	—	—	FI	DA	FO	EX			
	5					—	—	—	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Fig 4-8: Timing of Instruction Pipeline

- FI: the segment that fetches an instruction
- DA: the segment that decodes the instruction and calculate the effective address
- FO: the segment that fetches the operand
- EX: the segment that executes the instruction

Vector Operations

- Many scientific problems require arithmetic operations on large arrays of numbers.
- A vector is an ordered set of a one-dimensional array of data items.
- A vector V of length n is represented as a row vector by V=[v1,v2,…,Vn].
- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

- This is implemented in machine language by the following sequence of operations.

```
      Initialize I=0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I)+B(I)
      Increment I = I + 1
      If I <= 100 go to 20
      Continue
```

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.

  C(1:100) = A(1:100) + B(1:100)

- A possible instruction format for a vector instruction is shown in Fig. 4-11.
  - This assumes that the vector operands reside in *memory*.
- It is also possible to design the processor with a large number of *registers* and store all operands in registers prior to the addition operation.
  - The base address and length in the vector instruction specify a group of CPU registers.

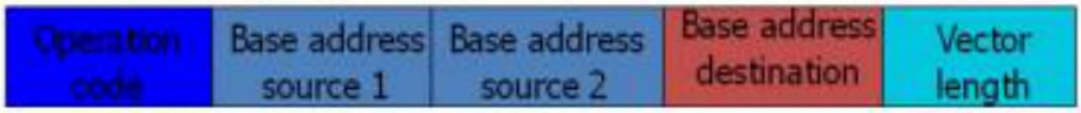


Fig 4-11: Instruction format for vector processor

Matrix Multiplication

- The multiplication of two n x n matrices consists of n^2 inner products or n^3 multiply-add operations.
  - Consider, for example, the multiplication of two 3 x 3 matrices A and B.
  - $c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$
  - This requires three multiplication and (after initializing c_{11} to 0) three additions.
- In general, the inner product consists of the sum of *k* product terms of the form $C = A_1B_1 + A_2B_2 + A_3B_3 + \ldots + A_kB_k$.
  - In a typical application k may be equal to 100 or even 1000.
- The inner product calculation on a pipeline vector processor is shown in Fig. 4-12.

$$C = A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \cdots$$
$$+ A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots$$
$$+ A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \cdots$$
$$+ A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \cdots$$

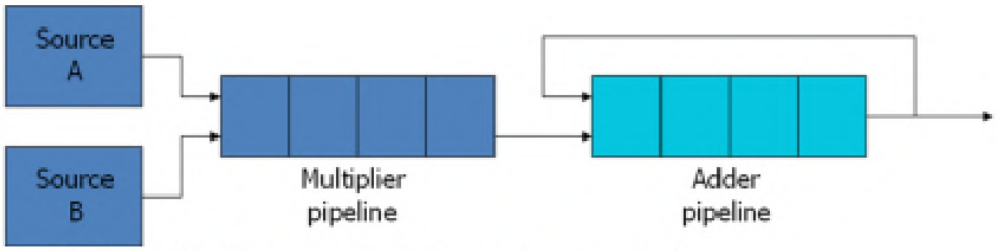


Fig 4-12: Pipeline for calculating an inner product

Memory Interleaving

- *Pipeline* and *vector processors* often require simultaneous access to memory from two or more sources.
  - An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments.
  - An arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time.
- Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses.
  - A memory module is a memory array together with its own address and data registers.

- Fig. 4-13 shows a memory unit with four modules.

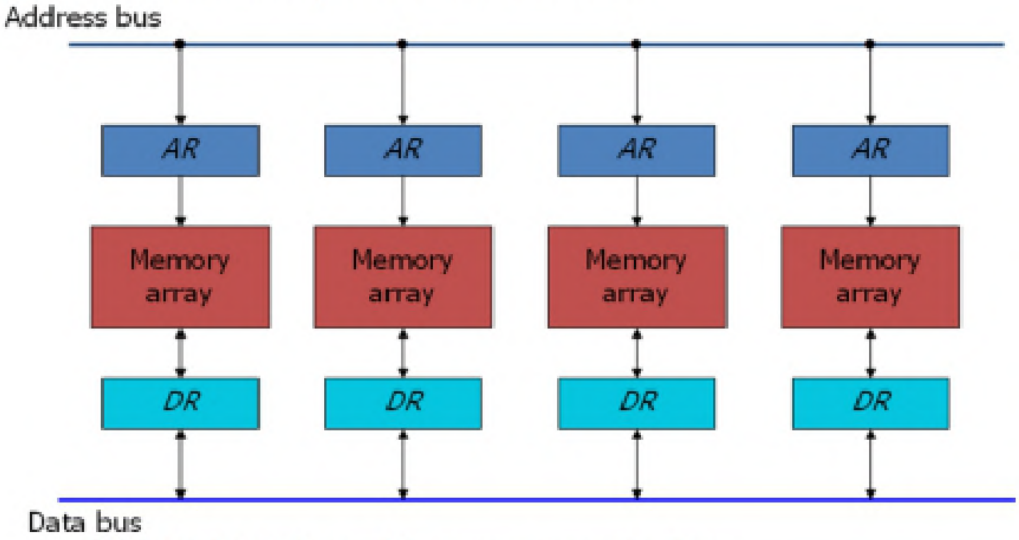


Fig 4-13: Multiple module memory organization

- The advantage of a modular memory is that it allows the use of a technique called *interleaving*.
- In an interleaved memory, different sets of addresses are assigned to different memory modules.
- By staggering the memory access, the effective memory cycle time can be *reduced by a factor close to the number of modules*.

Unit 6. Computer Arithmetic

Data Representation

Digital computers store and process information in binary form as digital logic has only two values "1" and "0" or in other words "True or False" or also said as "ON or OFF". This system is called radix 2. We human generally deal with radix 10 i.e. decimal. As a matter of convenience there are many other representations like Octal (Radix 8), Hexadecimal (Radix 16), Binary coded decimal (BCD), Decimal etc.

Every computer's CPU has a width measured in terms of bits such as 8 bit CPU, 16 bit CPU, 32 bit CPU etc. Similarly, each memory location can store a fixed number of bits and is called memory width. Given the size of the CPU and Memory, it is for the programmer to handle his data representation. Most of the readers may be knowing that 4 bits form a Nibble, 8 bits form a byte. The word length is defined by the Instruction Set Architecture of the CPU. The word length may be equal to the width of the CPU.

The memory simply stores information as a binary pattern of 1's and 0's. It is to be interpreted as what the content of a memory location means. If the CPU is in the Fetch cycle, it interprets the fetched memory content to be instruction and decodes based on Instruction format. In the Execute cycle, the information from memory is considered as data. As a common man using a computer, we think computers handle English or other alphabets, special characters or numbers. A programmer considers memory content to be data types of the programming language he uses. Now recall figure 1.2 and 1.3 of chapter 1 to reinforce your thought that conversion happens from computer user interface to internal representation and storage.

Data Representation in Computers

Information handled by a computer is classified as instruction and data. A broad overview of the internal representation of the information is illustrated in figure 3.1. No matter whether it is data in a numeric or non-numeric form or integer, everything is internally represented in Binary. It is up to the programmer to handle the interpretation of the binary pattern and this interpretation is called *Data Representation*. These data representation schemes are all standardized by international organizations.

Choice of Data representation to be used in a computer is decided by

- The number types to be represented (integer, real, signed, unsigned, etc.)
- Range of values likely to be represented (maximum and minimum to be represented)
- The Precision of the numbers i.e. maximum accuracy of representation (floating point single precision, double precision etc)
- If non-numeric i.e. character, character representation standard to be chosen. ASCII, EBCDIC, UTF are examples of character representation standards.
- The hardware support in terms of word width, instruction.

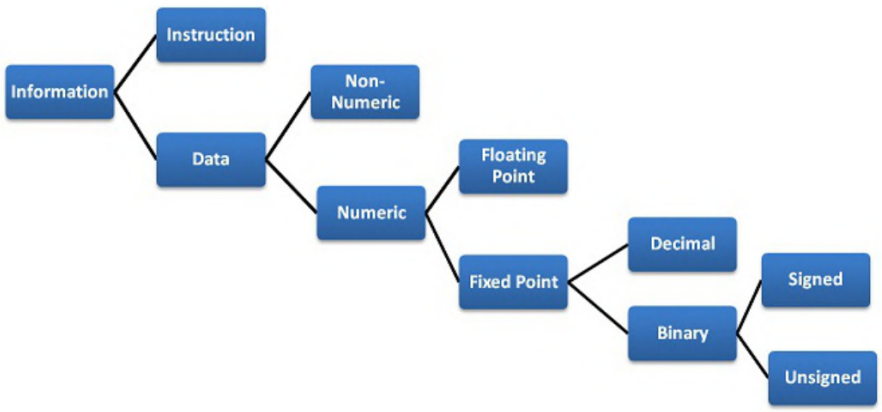


Figure.3.1 Typical Internal data representation types

Before we go into the details, let us take an example of interpretation. Say a byte in Memory has value "0011 0001". Although there exists a possibility of so many interpretations as in figure 3.2, the program has only one interpretation as decided by the programmer and declared in the program.

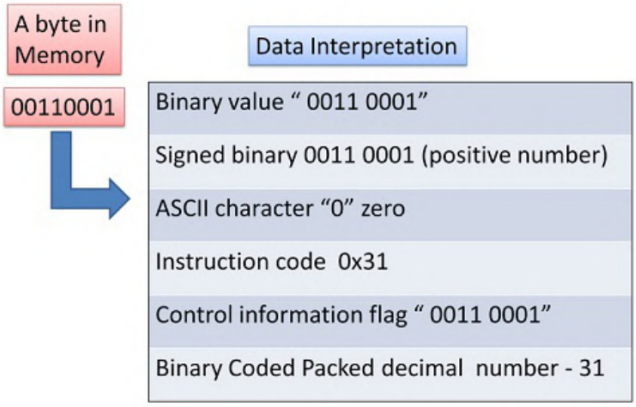


Figure.3.2 Data Interpretation

Fixed point Number Representation

Fixed point numbers are also known as whole numbers or Integers. The number of bits used in representing the integer also implies the maximum number that can be represented in the system hardware. However for the efficiency of storage and operations, one may choose to represent the integer with one Byte, two Bytes, Four bytes or more. This space allocation is translated from the definition used by the programmer while defining a variable as integer short or long and the Instruction Set Architecture.

In addition to the bit length definition for integers, we also have a choice to represent them as below:

- **Unsigned Integer**: A positive number including zero can be represented in this format. All the allotted bits are utilised in defining the number. So if one is using 8 bits to represent the unsigned integer, the range of values that can be represented is 28 i.e. "0" to "255". If 16 bits are used for representing then the range is 216 i.e. "0 to 65535".
- **Signed Integer**: In this format negative numbers, zero, and positive numbers can be represented. A sign bit indicates the magnitude direction as positive or negative. There are three possible representations for signed integer and these are **Sign Magnitude format, 1's Compliment format and 2's Complement format**.

Signed Integer – Sign Magnitude format: Most Significant Bit (MSB) is reserved for indicating the direction of the magnitude (value). A "0" on MSB means a positive number and a "1" on MSB means a negative number. If n bits are used for representation, n-1 bits indicate the absolute value of the number. Examples for n=8:

Examples for n=8:

0010 1111 = + 47 Decimal (Positive number)

1010 1111 = - 47 Decimal (Negative Number)

0111 1110 = +126 (Positive number)

1111 1110 = -126 (Negative Number)

0000 0000 = + 0 (Postive Number)

1000 0000 = - 0 (Negative Number)

Although this method is easy to understand, Sign Magnitude representation has several shortcomings like

- Zero can be represented in two ways causing redundancy and confusion.
- The total range for magnitude representation is limited to 2n-1, although n bits were accounted.

- The separate sign bit makes the addition and subtraction more complicated. Also, comparing two numbers is not straightforward.

Signed Integer – 1's Complement format: In this format too, MSB is reserved as the sign bit. But the difference is in representing the Magnitude part of the value for negative numbers (magnitude) is inversed and hence called 1's Complement form. The positive numbers are represented as it is in binary. Let us see some examples to better our understanding.

Examples for n=8:

0010 1111 = + 47 Decimal (Positive number)

1101 0000 = - 47 Decimal (Negative Number)

0111 1110 = +126 (Positive number)

1000 0001 = -126 (Negative Number)

0000 0000 = + 0 (Postive Number)

1111 1111 = - 0 (Negative Number)

Converting a given binary number to its 2's complement form

Step 1. -x = x' + 1 where x' is the one's complement of x.

Step 2 Extend the data width of the number, fill up with sign extension i.e. MSB bit is used to fill the bits.

Example: **-47 decimal over 8bit representation**

```
Binary equivalent of + 47 is      0010 1111
Binary equivalent of – 47 is      1010 1111 (Sign Magnitude Form)
1's complement equivalent is      1101 0000
2's complement equivalent is      1101 0001
```

As you can see zero is not getting represented with redundancy. There is only one way of representing zero. The other problem of the complexity of the arithmetic operation is also eliminated in 2's complement representation. Subtraction is done as Addition.

More exercises on number conversion are left to the self-interest of readers.

Floating Point Number system

The maximum number at best represented as a whole number is 2^n. In the Scientific world, we do come across numbers like Mass of an Electron is 9.10939 x 10-31 Kg. Velocity of light is 2.99792458 x 108 m/s. Imagine to write the number in a piece of paper without exponent and converting into binary for computer representation. Sure you are tired!!. It makes no sense to write a number in non- readable form or non-processible form. Hence we write such large or small numbers using exponent and mantissa. This is said to be Floating Point representation or real number representation. he real number system could have infinite values between 0 and 1.

Representation in computer

Unlike the two's complement representation for integer numbers, Floating Point number uses Sign and Magnitude representation for both *mantissa* and *exponent*. In the number 9.10939 x 1031, in decimal form, +31 is Exponent, 9.10939 is known as *Fraction*. Mantissa, Significand and fraction are synonymously used terms. In the computer, the representation is binary and the binary point is not fixed. For example, a number, say, 23.345 can be written as 2.3345 x 101 or 0.23345 x 102 or 2334.5 x 10-2. The representation 2.3345 x 101 is said to be in normalised form.

Floating-point numbers usually use multiple words in memory as we need to allot a sign bit, few bits for exponent and many bits for mantissa. There are standards for such allocation which we will see sooner.

IEEE 754 Floating Point Representation

We have two standards known as Single Precision and Double Precision from IEEE. These standards enable portability among different computers. Figure 3.3 picturizes Single precision while figure 3.4 picturizes double precision. Single Precision uses 32bit format while double precision is 64 bits word length. As the name suggests double precision can represent fractions with larger accuracy. In both the cases, MSB is sign bit for the mantissa part, followed by Exponent and Mantissa. The exponent part has its sign bit.

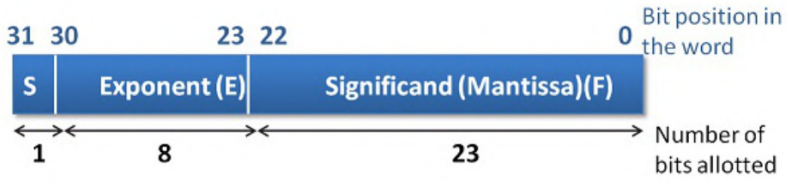


Figure.3.3 IEEE 754 Single Precision Floating Point representation Standard

It is to be noted that in Single Precision, we can represent an exponent in the range -127 to +127. It is possible as a result of arithmetic operations the resulting exponent may not fit in. This situation is called ***overflow*** in the case of positive exponent and ***underflow*** in the case of negative exponent. The Double Precision format has 11 bits for exponent meaning a number as large as -1023 to 1023 can be represented. The programmer has to make a choice between Single Precision and Double Precision declaration using his knowledge about the data being handled.

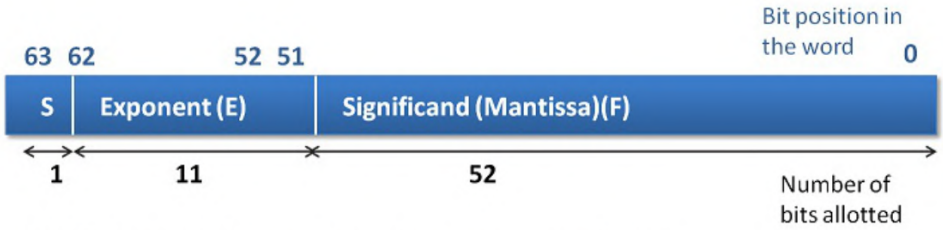


Figure 3.4 IEEE 754 Double Precision Floating Point representation Standard

The Floating Point operations on the regular CPU is very very slow. Generally, a special purpose CPU known as Co-processor is used. This Co-processor works in tandem with the main CPU. The programmer should be using the float declaration only if his data is in real number form. Float declaration is not to be used generously.

Decimal Numbers Representation

Decimal numbers (radix 10) are represented and processed in the system with the support of additional hardware. We deal with numbers in decimal format in everyday life. Some machines implement decimal arithmetic too, like floating-point arithmetic hardware. In such a case, the CPU uses decimal numbers in BCD (binary coded decimal) form and does BCD arithmetic operation. BCD operates on radix 10. This hardware operates without conversion to pure binary. It uses a nibble to represent a number in packed BCD form. BCD operations require not only special hardware but also decimal instruction set.

Exceptions and Error Detection

All of us know that when we do arithmetic operations, we get answers which have more digits than the operands (Ex: 8 x 2= 16). This happens in computer arithmetic operations too. When the result size exceeds the allotted size of the variable or the register, it becomes an error and exception. The exception conditions associated with numbers and number operations are *Overflow, Underflow, Truncation, Rounding and Multiple Precision*. These are detected by the associated hardware in arithmetic Unit. These exceptions apply to both Fixed Point and Floating Point operations. Each of these exceptional conditions has a flag bit assigned in the Processor Status Word (PSW). We may discuss more in detail in the later chapters.

Character Representation

Another data type is non-numeric and is largely character sets. We use a human-understandable character set to communicate with computer i.e. for both input and output. Standard character sets like EBCDIC and ASCII are chosen to represent alphabets, numbers and special characters. Nowadays Unicode standard is also in use for non-English language like Chinese, Hindi, Spanish, etc. These codes are accessible and available on the internet. Interested readers may access and learn more.

6.2 INPUT-OUTPUT INTERFACE

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be need.
3. Data codes and formats in peripherals differ form the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units because they interface between the processor bus and the peripheral device. In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

6.2.1 I/O BUS AND INTERFACE MODULES

A typical communication link between the processor and several peripherals is shown in Fig. 6-1 The I/O bus consists of data lines, address lines, and control lines. The magnetic disk, printer, and terminal are employed in practically any general-purpose computer. The magnetic tape is used in some computers for backup storage. Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device. For example, the printer controller controls the paper motion, the print timing, and the selection of printing characters. A controller may be housed separately or may be physically integrated with the peripheral.

The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled their interface.

At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface

selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit. The interpretation of the command depends on the peripheral that the processor is addressing. There are four types of commands that an interface may receive. They are classified as control, status, status, data output, and data input.

A control command is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction. The particular control command issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operation.

A status command is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

I/O Bus and Interface Modules

- **The I/O bus consists of data lines, address lines and control lines.**

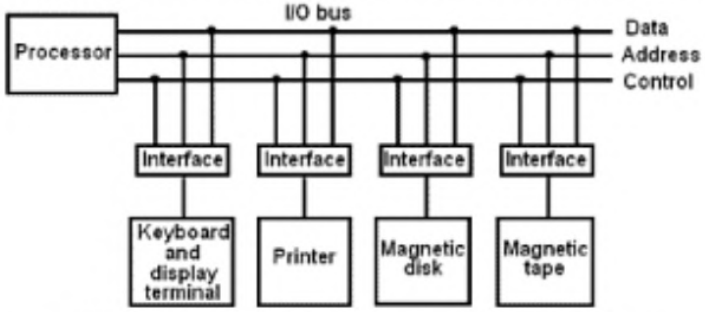


Fig: Connection of I/O bus to input-output devices

6.2.2 I/O VERSUS MEMORY BUS

In addition to communicating with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address, and read/write control lines. There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to the central processing unit (CPU). The memory communicates with both the CPU and the IOP through a memory bus. The IOP communicates also with the input and output devices through a separate I/O bus with its own address, data and control lines. The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory.

Isolated vs. Memory-Mapped I/O

Isolated I/O uses a separate set of lines for reading and writing data to I/O devices, while memory-mapped I/O uses the same address space for both memory and I/O devices.

Isolated I/O is more complex to implement than memory-mapped I/O, but it offers several advantages, such as:

- Improved security: Because I/O devices have their own separate address space, it is more difficult for them to interfere with the memory or other I/O devices.
- Greater flexibility: I/O devices can be added or removed from the system without affecting the memory address space.
- Improved reliability: If an I/O device fails, it is less likely to affect the memory or other I/O devices.

Memory-mapped I/O is simpler to implement than isolated I/O, but it has some disadvantages, such as:

- Reduced security: Because I/O devices share the same address space as memory, it is easier for them to interfere with the memory or other I/O devices.
- Reduced flexibility: I/O devices cannot be added or removed from the system without affecting the memory address space.
- Reduced reliability: If an I/O device fails, it is more likely to affect the memory or other I/O devices.

In summary, isolated I/O is a more secure and flexible option than memory-mapped I/O, but it is also more complex to implement. Memory-mapped I/O is simpler to implement, but it is less secure and flexible.

Asynchronous Data Transfer

Asynchronous data transfer is a communication method where data is transmitted or received without a synchronized clock, allowing devices to operate independently in terms of timing. Types of Asynchronous Data Transfer are.

1. Strobe Control

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source of the destination unit. Below figure shows a source-initiated transfer.

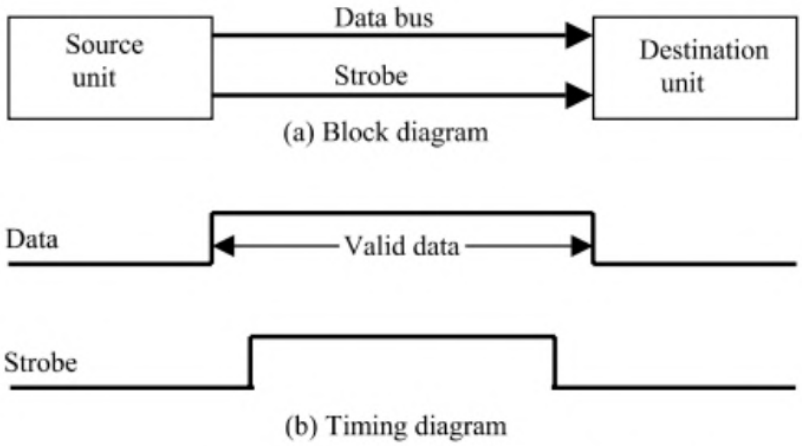


Figure 6-3 Source-initiated strobe for data transfer.

The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

As shown in the timing diagram of Fig. 6-3(b), the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse. The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data. Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers. The source removes the data from the bus a brief period after it disables its strobe pulse. Actually, the source does not have to change the information in the data bus. The fact that the strobe signal is disabled indicates that the data bus does not contain valued data. New valid data will be available only after the strobe is enabled again.

Figure 6-4 shows a data transfer initiated by the destination unit. In this case the destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. The falling edge of the

strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe. The source removes the data from the bus after a predeter-mined time interval.

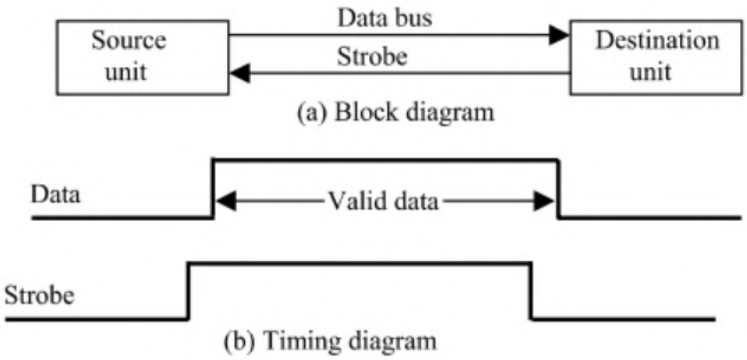


Figure 6-4 Destination-initiated strobe for data transfer.

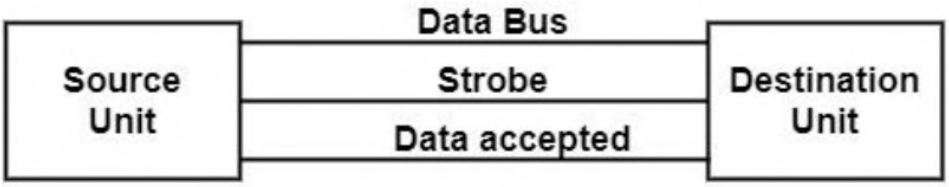
which is the destination, that this is a write operation. Similarly, the strobe of fig. 6-4 could be a memory-read control signal from the CPU to a memory unit. The destination, the CPU, initiates the read operation to inform the memory, which is the source, to place a selected word into the data bus.

The transfer of data between the CPU and an interface unit is similar to the strobe transfer just described. Data transfer between an interface and an I/O device is commonly controlled by a set of handshaking lines.

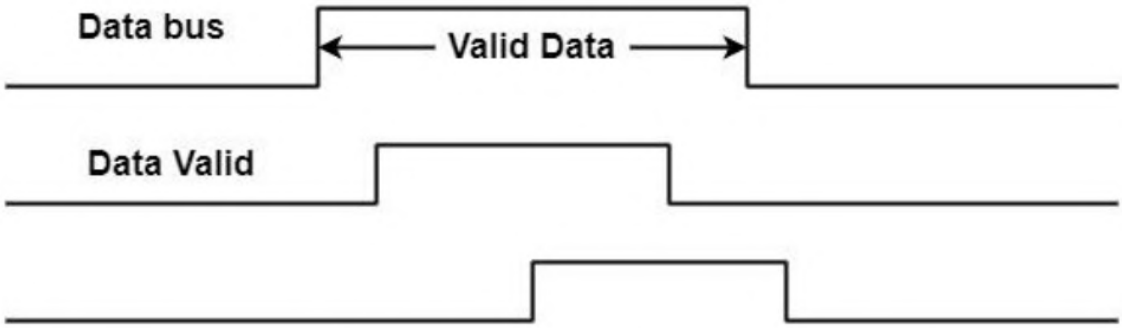
Handshaking

1. **Handshaking in I/O Control:**
   - Handshaking is an I/O control method used to synchronize I/O devices with the microprocessor.
   - It ensures the microprocessor operates with an I/O device at the device's data transfer rate.
2. **Cost Efficiency:**
   - Many I/O devices handle data at a lower cost than the microprocessor.
   - Handshaking allows the microprocessor to coordinate with I/O devices efficiently.
3. **Drawback of Strobe Approach:**
   - Strobe approach lacks feedback; the source unit initiating the transfer doesn't know if the destination unit received the data, and vice versa.

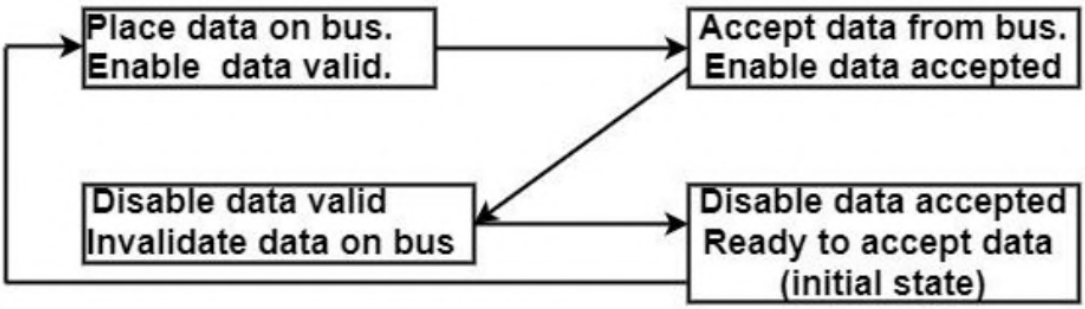
4. **Handshake Solution:**
   - Handshaking introduces a second control signal for feedback, addressing the issue in the strobe approach.
5. **Two-Wire Handshaking:**
   - Two control lines facilitate the handshaking process:
     - One line, in the direction of data flow, updates the destination unit about the presence of valid data on the bus (from source to destination).
     - The other line, in the opposite direction, allows the destination unit to inform the source if it can accept information.
6. **Control Sequence:**
   - The sequence of control during the transfer depends on the unit initiating the transfer.
   - Source unit uses one line to update the destination about valid data.
   - Destination unit uses the other line to inform the source whether it can accept information.



(a) Block Diagram



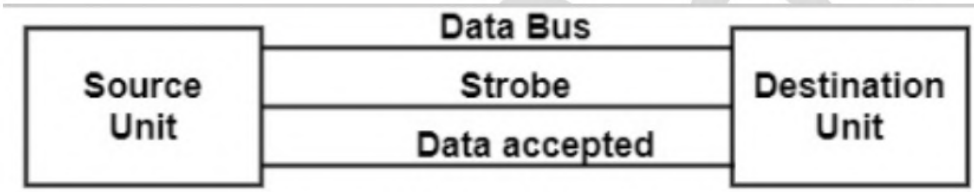
(b) Timing Diagram



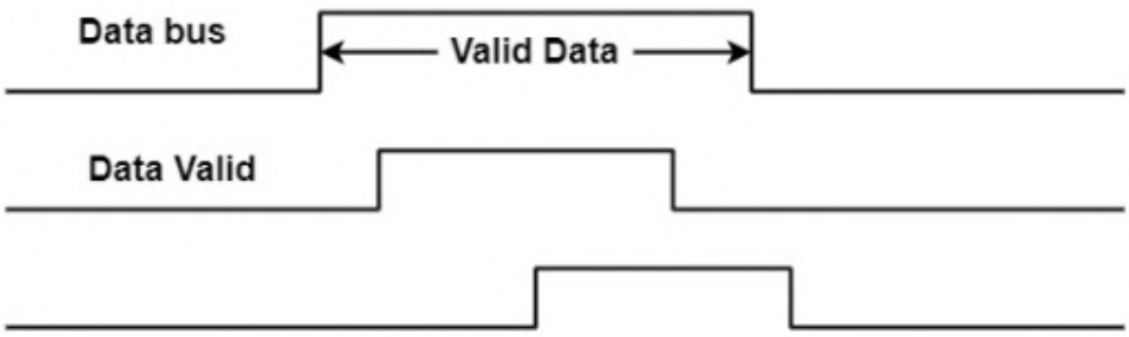
(c) Sequence of events

Source -Initiated transfer using Handshaking

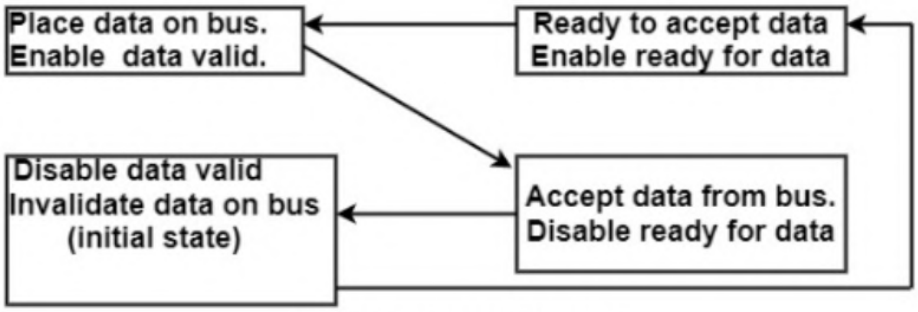
1. The system exhibits four possible states during data transfer, detailed in part (c).
2. Source initiation involves the source unit locating information, activating its "Data Valid" signal.
3. Data acceptance is signaled by the destination unit activating the "Data Accepted" signal after obtaining data.
4. Destination-initiated transfer introduces a modified signal, "Ready for Data," generated by the destination unit.
5. In this scenario, the source unit refrains from locating data until it receives the "Ready for Data" signal.
6. The sequence emphasizes coordination through handshaking, ensuring acknowledgment and controlled data transfer between source and destination units.



(a) Block Diagram



(b) Timing Diagram



(c) Sequence of events

Destination -Initiated transfer using Handshaking

The sequence of events in both cases would be equal if it can consider the ready-for-data signal as the complement of data accepted. The only difference between the source-initiated and the destination-initiated transfer is in their choice of the initial state.

Modes of transfer

Modes of transfer refer to different methods or approaches for transferring data between the central processing unit (CPU) and peripheral devices in a computer system. There are three main modes of transfer:

1) **Programmed I/O (Input/Output):**

Description: In Programmed I/O, the CPU actively manages and controls the data transfer between peripherals and memory.

Process: The CPU issues commands and directly controls the data transfer, often entering a wait state until the I/O operation completes.

Advantages: Simple implementation and precise control over data transfer.

Disadvantages: Resource-intensive as the CPU is heavily involved, and it can be slower due to constant CPU intervention

2) **Interrupt-Initiated I/O**:

Description: In Interrupt-Initiated I/O, peripheral devices can request the attention of the CPU when they are ready for data transfer.

Process: Devices initiate an interrupt, signaling the CPU to pause its current task and handle the I/O request.

Advantages: Efficient CPU usage as it responds to device needs, suitable for systems with multiple devices.

Disadvantages: Adds complexity to the system due to interrupt handling, and there may be potential latency if multiple devices compete for CPU attention.

3) **Direct Memory Access (DMA):**

- **Description**: In DMA, a separate DMA controller manages the data transfer directly between peripheral devices and memory without continuous CPU involvement.

OR

- Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

Process: The CPU sets up DMA parameters, but the actual data transfer is handled independently by the DMA controller.

Advantages: Enables high-speed transfer, particularly efficient for large data blocks, and frees up the CPU for other tasks during data transfer.

Disadvantages: Involves complex setup compared to programmed I/O, and the CPU relinquishes some control during data transfer.

Characteristics:

1. **Controller:** DMA is managed by a separate hardware component known as the DMA controller. This controller takes over the control bus from the CPU during data transfer operations.

2. **CPU Initiation:** The CPU initiates the DMA operation by providing necessary information to the DMA controller, such as the source and destination addresses in memory, and the length of the data to be transferred.
3. **Independent Operation:** Once configured, the DMA controller performs data transfers independently of the CPU. This independence enhances overall system performance by allowing the CPU to execute other instructions while data transfers are in progress.
4. **Efficiency:** DMA is efficient for transferring large blocks of data between memory and peripherals, reducing the time and resources required for data movement.
5. **Speed:** Since the CPU is not involved in each data transfer, DMA can significantly improve the speed of data movement between devices and memory.
6. **Interrupts:** After completing a data transfer, the DMA controller may generate an interrupt to signal the CPU about the completion of the operation. This allows the CPU to perform additional processing or to handle other tasks.

Input-Output Processor (IOP):

Explanation:

An Input-Output Processor (IOP) is a specialized processor or controller responsible for managing input and output operations in a computer system. It serves as an intermediary between the CPU and peripheral devices, handling the details of data transfer and communication with external devices.

Characteristics:

1. **Dedicated Function:** The primary function of an IOP is to handle the input and output operations, freeing the CPU from managing these tasks directly.
2. **Peripheral Communication:** The IOP communicates with various peripherals, such as disk drives, printers, and communication ports, to facilitate data transfer between the CPU and external devices.

3. **Data Buffering:** IOPs often include data buffers to temporarily store data being transferred between the CPU and peripherals. This buffering helps smooth out speed mismatches between the CPU and slower peripherals.
4. **Parallel Processing:** Some IOPs may have the capability for parallel processing, allowing them to perform multiple I/O operations concurrently.
5. **Interrupt Handling:** IOPs generate interrupts to notify the CPU when an I/O operation is complete or requires attention. This allows the CPU to respond promptly to I/O events.
6. **Offloading CPU:** Similar to DMA, IOPs contribute to offloading tasks from the CPU, enabling it to focus on executing application instructions without being heavily involved in managing peripheral devices.

Direct Memory Access (DMA) vs Input-Output Processor (IOP)

Aspect	Direct Memory Access (DMA)	Input-Output Processor (IOP)
Initiation	Initiated by CPU for specific data transfers.	Operates independently, handling various I/O operations.
Control	Controlled by a DMA controller during transfers.	Independently manages communication for I/O devices.
Dependency	Reduces CPU dependency during data transfers.	Offloads I/O tasks from the CPU, enhancing overall system efficiency.
Scope	Focused on efficient memory-to-peripheral data transfer.	Manages a broader range of I/O operations for various peripherals.
Data Movement	Optimized for high-speed data transfer between memory and peripherals.	Manages data movement between CPU and diverse peripherals.

Interrupts	May generate interrupts to signal CPU about transfer completion.	Typically generates interrupts for I/O events, notifying the CPU.
Buffering	Primarily designed for direct and efficient data transfers.	May include data buffers for temporary storage during data transfer.
Parallel Processing	Primarily designed for sequential data transfers, one operation at a time.	Some IOPs may have parallel processing capabilities for concurrent I/O operations.
Examples	Commonly used in scenarios like graphics processing or bulk data storage.	Examples include disk controllers, network interface controllers, and specialized processors for specific I/O devices.
Overall Function	Facilitates efficient data transfer between memory and peripherals.	Manages a variety of input and output operations, acting as an intermediary between CPU and peripherals.