UNIT 5

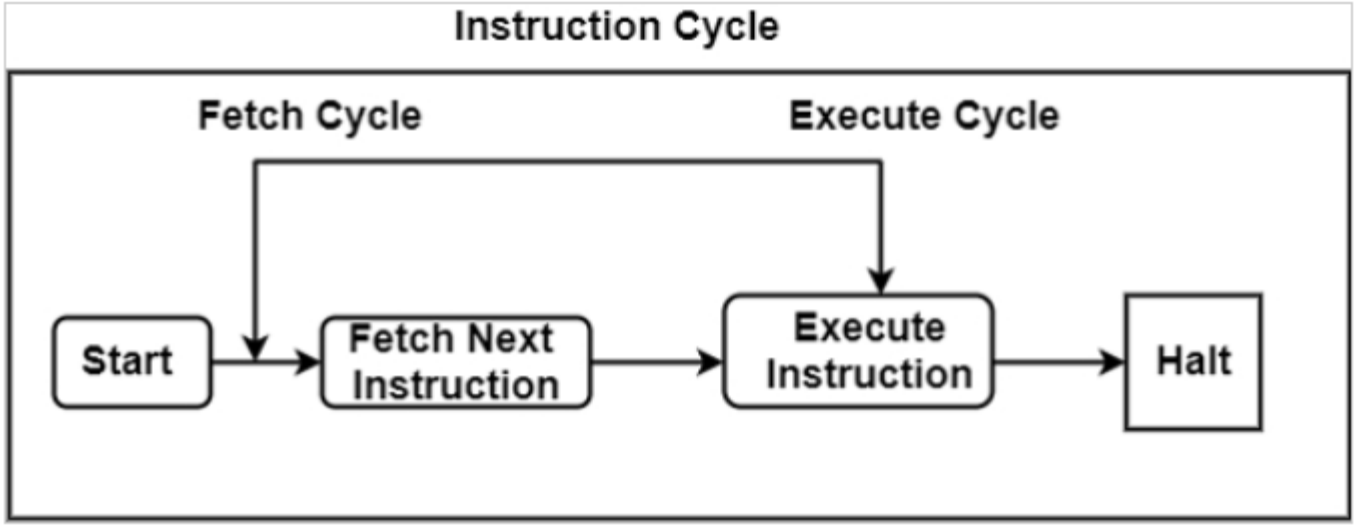
Central Processing Unit

Central Processing Unit (6hrs)

- Basic Instruction Cycle and Instruction set,
- Formats and addressing,
- Processor Organization and Register Organization,
- Instruction Pipelining,
- Co-processors,
- Pipeline processors,
- RISC and CISC computers.

Basic Instruction Cycle and Instruction set

- A program consisting of the memory unit of the computer includes a series of instructions. The program is implemented on the computer by going through a cycle for each instruction.
- In the basic computer, each instruction cycle includes the following procedures −
- It can fetch instruction from memory.
- It is used to decode the instruction.
- It can read the effective address from memory if the instruction has an indirect address.
- It can execute the instruction.
- After the following four procedures are done, the control switches back to the first step and repeats the similar process for the next instruction. Therefore, the cycle continues until a **Halt** condition is met. The figure shows the phases contained in the instruction cycle.



Basic Instruction Cycle and Instruction set

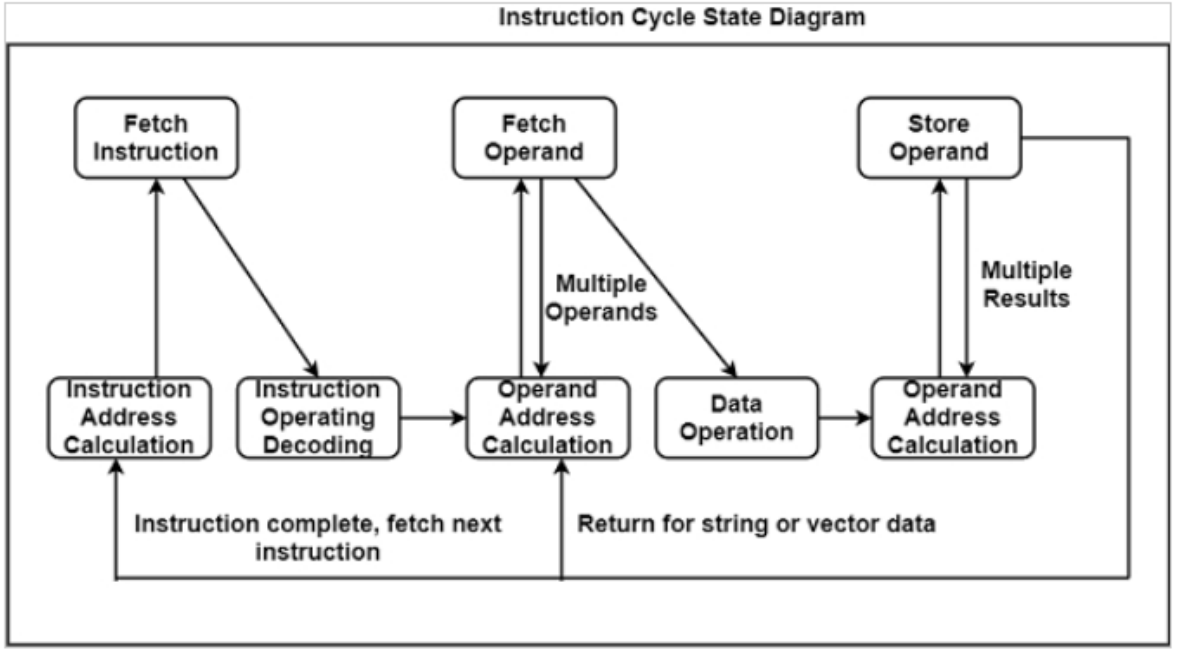
- **Registers Involved In Each Instruction Cycle**
- Following are the different types of registers involved in each instruction cycle:
- **Memory address registers(MAR):** It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory Buffer Register(MBR):** It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- **Program Counter(PC):** Holds the address of the next instruction to be fetched.
- **Instruction Register(IR):** Holds the last instruction fetched.

Basic Instruction Cycle and Instruction set

- **Fetch Cycle**
- The address instruction to be implemented is held at the program counter. The processor fetches the instruction from the memory that is pointed by the PC.
- Next, the PC is incremented to display the address of the next instruction. This instruction is loaded onto the instruction register. The processor reads the instruction and executes the important procedures.
- **Execute Cycle**
- The data transfer for implementation takes place in two methods are as follows −
- **Processor-memory** − The data sent from the processor to memory or from memory to processor.
- **Processor-Input/Output** − The data can be transferred to or from a peripheral device by the transfer between a processor and an I/O device.
- In the execute cycle, the processor implements the important operations on the information, and consistently the control calls for the modification in the sequence of data implementation. These two methods associate and complete the execute cycle.

State Diagram for Instruction Cycle

The figure provides a large aspect of the instruction cycle of a basic computer, which is in the design of a state diagram. For an instruction cycle, various states can be null, while others can be visited more than once.

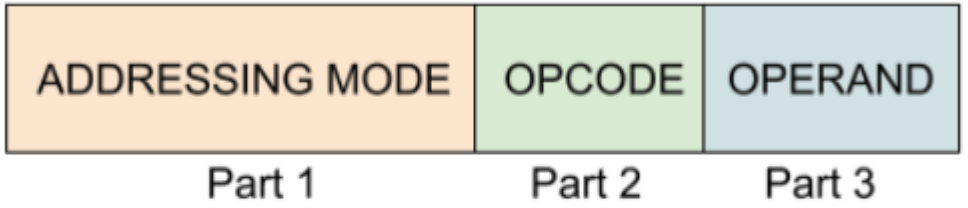


Basic Instruction Cycle and Instruction set

- **Instruction Address Calculation** − The address of the next instruction is computed. A permanent number is inserted to the address of the earlier instruction.
- **Instruction Fetch** − The instruction is read from its specific memory location to the processor.
- **Instruction Operation Decoding** − The instruction is interpreted and the type of operation to be implemented and the operand(s) to be used are decided.
- **Operand Address Calculation** − The address of the operand is evaluated if it has a reference to an operand in memory or is applicable through the Input/Output.
- **Operand Fetch** − The operand is read from the memory or the I/O.
- **Data Operation** − The actual operation that the instruction contains is executed.
- **Store Operands** − It can store the result acquired in the memory or transfer it to the I/O.

Instruction Formats and addressing

- The computer program consists of a number of instructions that directs the CPU to perform specific operations. However, the CPU needs to know the details such as which operation is to be performed, on which data, and the location of the data. The information is provided by the instruction format.
- The CPU starts the program execution by fetching the program instructions one by one from the main memory (RAM). The control unit of the CPU decodes the program instruction.



Part 1: Addressing Mode - Rule for Operand
Part 2: OPCODE - For Control Unit - Which operation to perform
Part 3: OPERAND - For ALU - On data operation to be performed

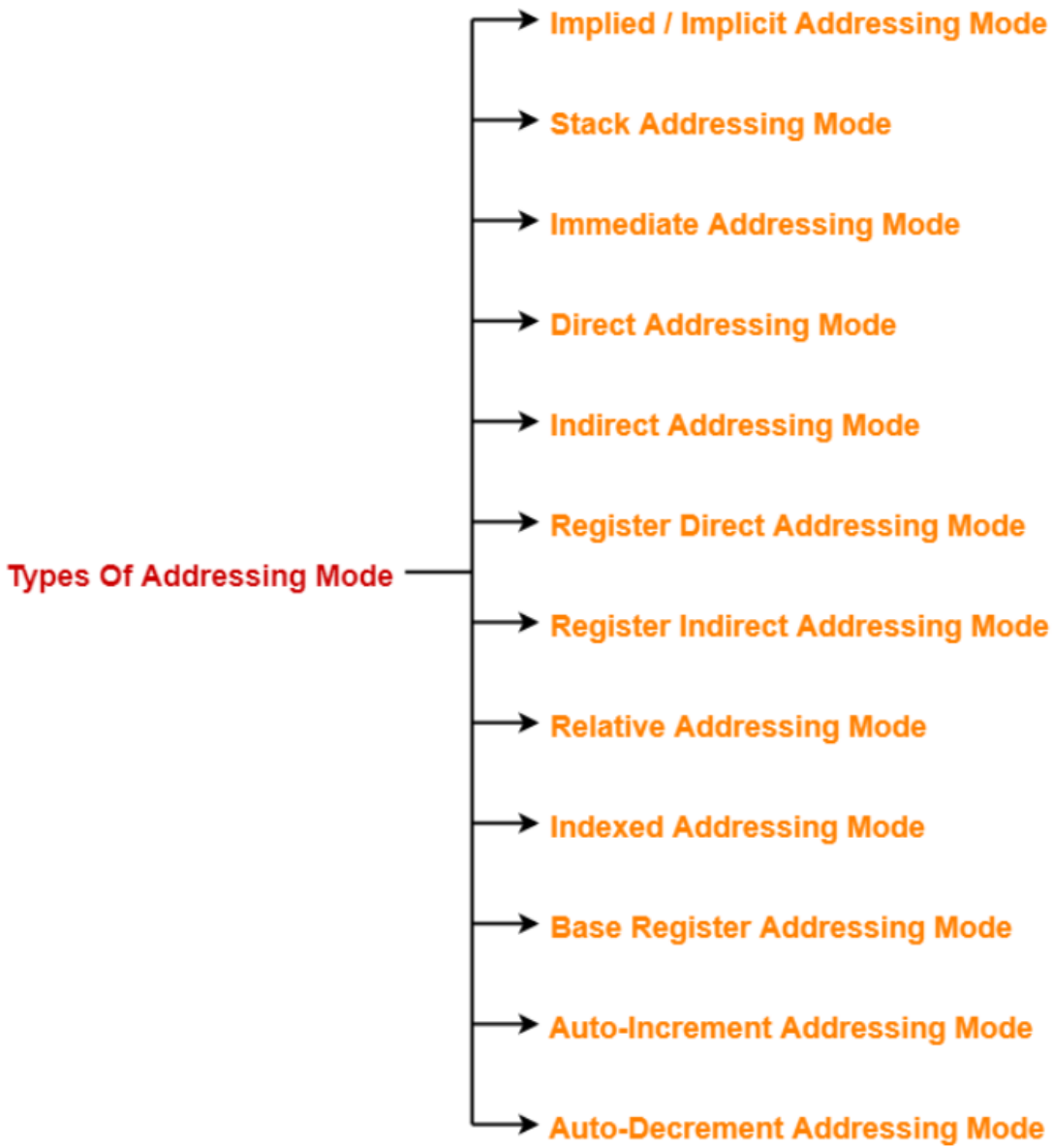
- The control unit of the CPU decodes the instruction based on the instruction format. It is the instruction format that provides the details of the operation to be performed (opcode), the effective address of the operand, and the data (operand) on which the operation is to be performed.
- The instruction format defines the layout and structure of the program instruction that can be decoded by the CPU and then perform the desired operation on the data.

Formats and addressing

- **Addressing Mode:** The addressing mode specifies the rules for the CPU while operating on the OPERAND part of the machine instruction. The addressing mode allows specifying whether the OPERAND value is direct data or it is an indirect referencing. The OPERAND bits can either represent a direct value, main memory address or CPU register number. It is the addressing mode that indicates the type of the OPERAND value. If the addressing mode is specified as indirect then the OPERAND contains a memory address that points to the actual data.
- **OPCODE:** OPCODE specifies which operation is to be performed by the CPU while executing the instruction. The OPCODE directs the control unit of the CPU to operate on the data (OPERAND) as supported by the instruction set architecture (ISA) of the processor chip.
- **OPERAND:** OPERAND simply means the data on which the CPU performs the desired operation. The OPERAND specifies either the data itself or a reference to the data as a memory address that contains the actual data. The CPU decodes the OPERAND as specified in the addressing mode. There can be different types of addressing modes used in the instruction format.

Addressing Modes

- The operands of the instructions can be located either in the main memory or in the CPU registers. If the operand is placed in the main memory, then the instruction provides the location address in the operand field. Many methods are followed to specify the operand address. The different methods/modes for specifying the operand address in the instructions are known as addressing modes.
- **Types of Addressing Modes**
- **Implied Mode** − In this mode, the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. All register reference instructions that use an accumulator are implied-mode instructions.
- **Immediate Mode** − In this mode, the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field instead of an address field. The operand field includes the actual operand to be used in conjunction with the operation determined in the instruction. Immediate-mode instructions are beneficial for initializing registers to a constant value.
- **Register Mode** − In this mode, the operands are in registers that reside within the CPU. The specific register is selected from a register field in the instruction. A k-bit field can determine any one of the 2k registers.

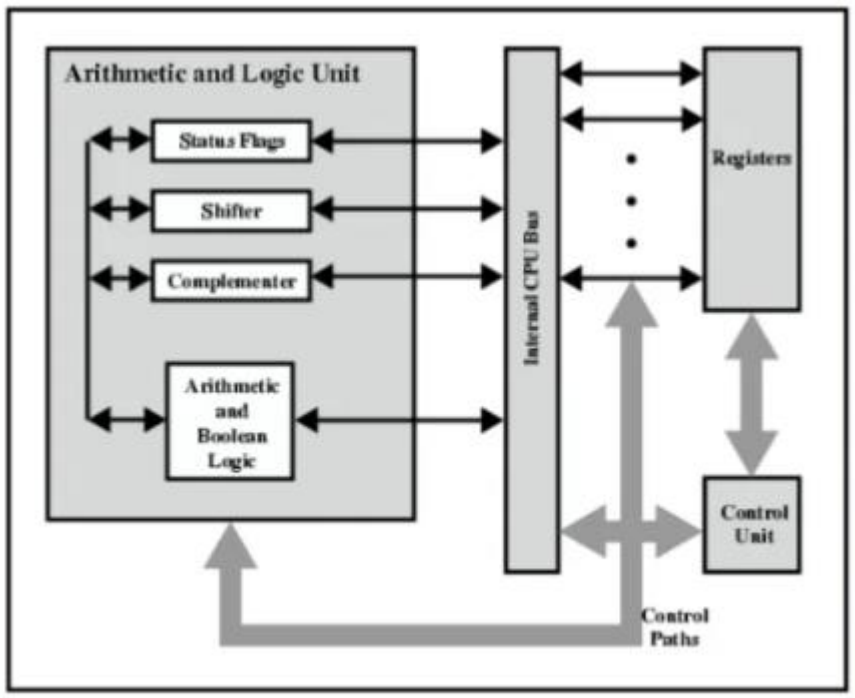

Types Of Addressing Mode
Implied / Implicit Addressing Mode
Stack Addressing Mode
Immediate Addressing Mode
Direct Addressing Mode
Indirect Addressing Mode
Register Direct Addressing Mode
Register Indirect Addressing Mode
Relative Addressing Mode
Indexed Addressing Mode
Base Register Addressing Mode
Auto-Increment Addressing Mode
Auto-Decrement Addressing Mode

Addressing Modes

- **Register Indirect Mode** − In this mode, the instruction defines a register in the CPU whose contents provide the address of the operand in memory. In other words, the selected register includes the address of the operand rather than the operand itself.
- A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
- **Auto increment or Auto decrement Mode** &minuend; This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register defines a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be obtained by using the increment or decrement instruction.
- **Direct Address Mode** − In this mode, the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction, the address field specifies the actual branch address.
- **Indirect Address Mode** − In this mode, the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- **Indexed Addressing Mode** − In this mode, the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory.

Processor Organization and Register Organization

Processor Organization



- The register in the processor perform two roles:

1. **User-visible register:** Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.
2. **Control and status registers:** Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

Processor Organization and Register Organization

User-visible Registers

CATEGORIES:-

- General Purpose
- Data
- Address
- Condition Codes

General Purpose

- **General Purpose Registers** can be assigned to a variety of functions by the programmer
- Mostly these registers contain the operand for any opcode.
- In some cases these are used for addressing purpose.

Processor Organization and Register Organization

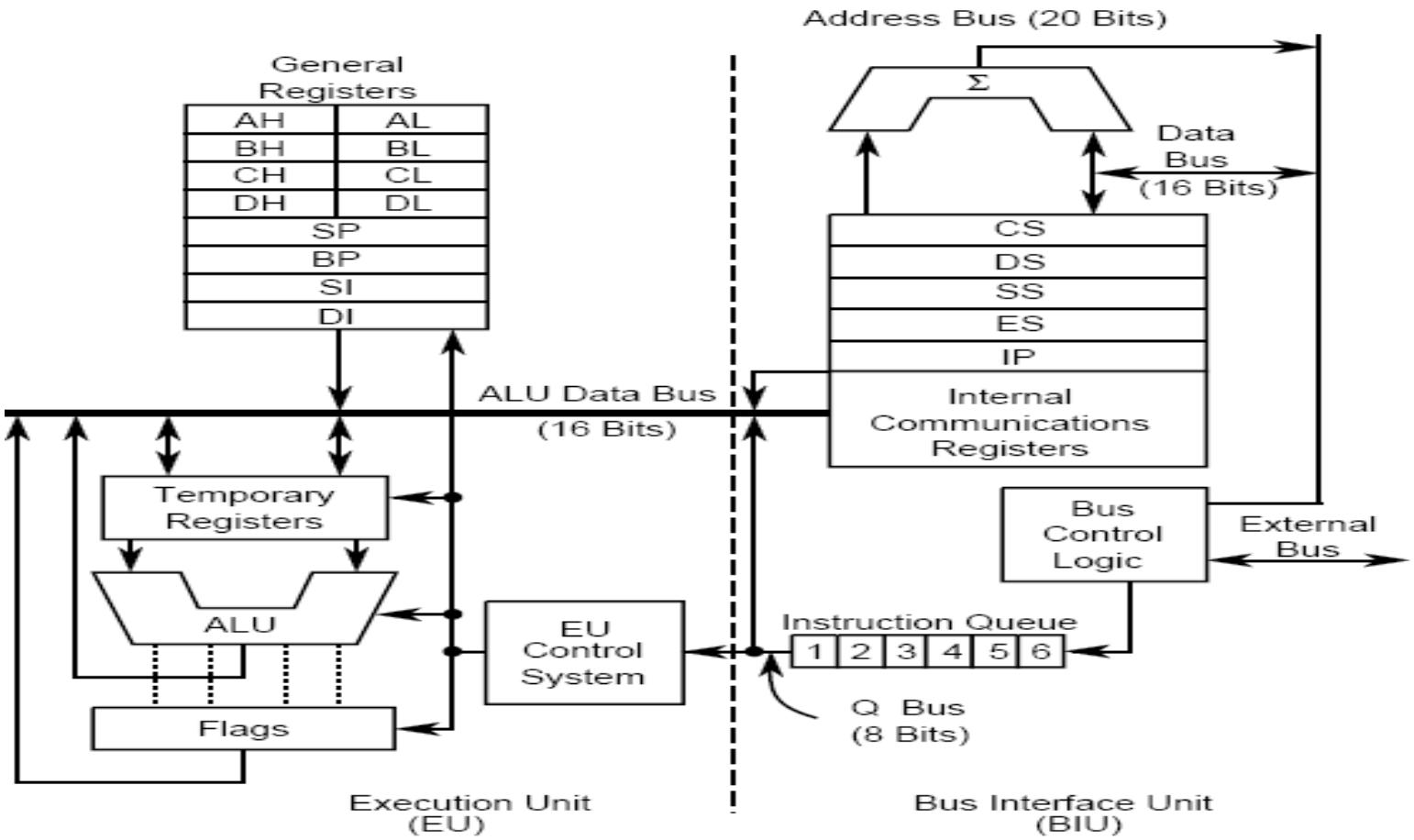
Data Registers

- **Data Register** to hold data and cannot be employed in the calculation of an operand address
- Eg. Accumulator.

Address Registers

- **Address Register** they may be devoted to a particular addressing mode
  - **Segment pointers** :a segment register holds the address of the base of the segment
  - **Index registers** :are used for indexed addressing and may be autoindexed.
  - **Stack Pointer:** If there is user-visible stack addressing, then typically there is a dedicated register that points to the top of the stack.

Architecture Diagram of 8086 Processor



EU(Execution Unit)& BIU(Bus Interface Unit)

- The 8086 CPU logic has been partitioned into two functional units namely **Bus Interface Unit (BIU)** and **Execution Unit (EU**)

- The major reason for this separation is to increase the processing speed of the processor

- The **BIU** has to **interact** with **memory, input** and **output devices** in fetching the instructions and data required by the EU.

- **EU** is responsible for **executing** the instructions of the programs and to carry out the required processing .

Bus Interface Unit

- The **BIU** has
  - Instruction stream byte **QUEUE**
  - A set of **Segment Registers**
  - **Instruction Pointer**

BIU – Instruction Byte Queue

- 8086 instructions vary from 1 to 6 bytes
- Therefore fetch and execution are taking place concurrently in order to improve the performance of the microprocessor
- The BIU feeds the instruction stream to the execution unit through a 6 byte prefetch queue
- This prefetch queue can be considered as a form of loosely coupled pipelining

BIU – Instruction Byte Queue

- Execution and decoding of certain instructions do not require the use of buses
- While such instructions are executed, the BIU fetches up to six instruction bytes for the following instructions (the subsequent instructions)
- The BIU store these pre fetched bytes in a first-in-first out register by name instruction byte queue
- When the EU is ready for its next instruction, it simply reads the instruction byte(s) for the instruction from the queue in BIU

Addressing Modes 8086

- Implied Addressing – The data value/data address is implicitly associated with the instruction
- Register Addressing – The data is specified by referring the register or the register pair in which the data is present
- Immediate Addressing – The data itself is provided in the instruction
- Direct Addressing – The instruction operand specifies the memory address where data is located

Addressing Modes

- **Register indirect addressing** – The instruction specifies a register containing an address, where data is located .
- **Based** - 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.
- **Indexed** - 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

Addressing Modes

- **Based Indexed** - the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides
- **Based Indexed with displacement** - 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides

Classification of Instructions

- **The 8086 microprocessor supports 8 types of instructions – Data Transfer Instructions**
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions) Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Data Transfer Instructions

General-Purpose	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push registers onto stack
POPA	Pop registers off stack
XCHG	Exchange byte or word
XLAT	Translate byte

Data Transfer Instructions

Input/Output	
IN	Input byte or word
OUT	Output byte or word
Address Object and Stack Frame	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
ENTER	Build stack frame
LEAVE	Tear down stack frame
Flag Transfer	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags from stack
POPF	Pop flags off stack

Arithmetic Instructions

Addition	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
Subtraction	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction

Arithmetic Instructions

Multiplication	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiplication
Division	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to double-word

Logical Instructions

Logicals	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
Shifts	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
Rotates	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

String Instructions

REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
MOVSB/MOVSW	Move byte string/word string
MOVS	Move byte or word string
INS	Input byte or word string
OUTS	Output byte or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string

Program Transfer Instructions

Conditional Transfers	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign

Program Transfer Instructions

Unconditional Transfers	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
Iteration Control	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX=0
Interrupts	
INT	Interrupt
INTO	Interrupt if overflow
BOUND	Interrupt if out of array bounds
IRET	Interrupt return

Processor Control Instructions

Flag Operations	
STC	Set Carry flag
CLC	Clear Carry flag
CMC	Complement Carry flag
STD	Set Direction flag
CLD	Clear Direction flag
STI	Set Interrupt Enable flag
CLI	Clear Interrupt Enable flag
External Synchronization	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to external processor
LOCK	Lock bus during next instruction
No Operation	
NOP	No operation

Assembler Directives

- **Assembler directives give instruction** to the assembler where as other instructions discussed in the above section give instruction to the 8086 microprocessor
- Assembler directives are specific for a particular assembler
- However all the popular assemblers like the Intel 8086 macro assembler, the turbo assembler and the IBM macro assembler use common assembler directives

Important Directives

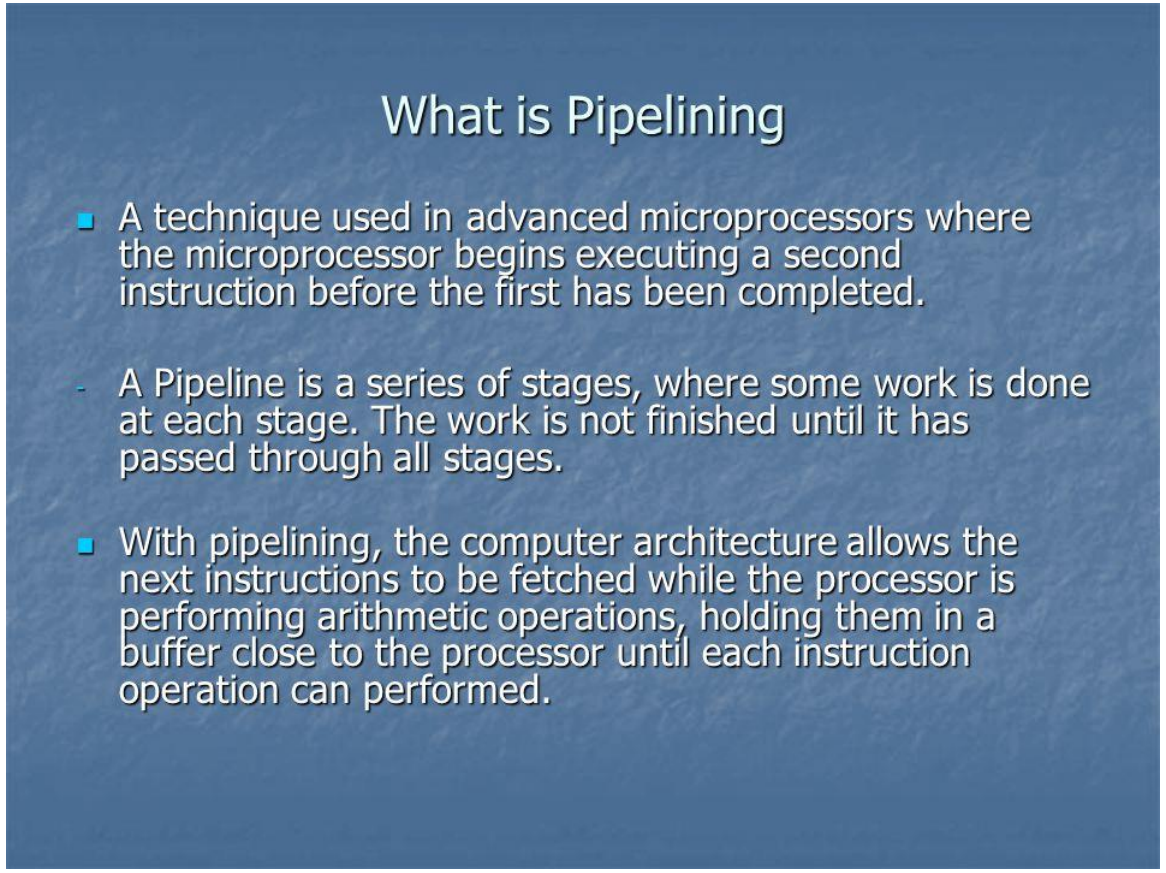
- The **ASSUME** directive tell the assembler the name of the logical segment it should use for a specified segment

- The **DB** directive is used to declare a byte-type variable or to set aside one or more storage locations of type byte in memory (Define Byte)

- The **DD** directive is used to declare a variable of type double word or to reserve memory locations which can be accessed as type double word (Define Double word)

- The **DQ** directive is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage in memory (Define Quad word)

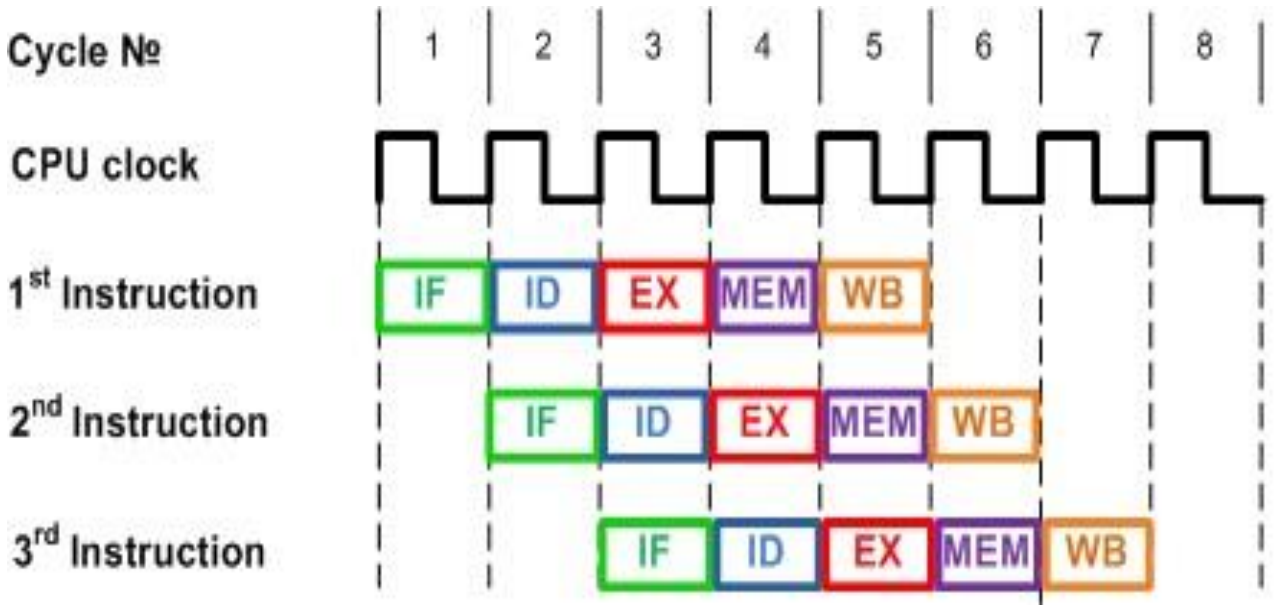
Instruction Pipelining



- In a pipelined processor, a pipeline has two ends, the input end and the output end.
- **Design of a basic pipeline**
- Between these ends, there are multiple stages/segments such that the output of one stage is connected to the input of the next stage and each stage performs a specific operation.
- Interface registers are used to hold the intermediate output between two stages.
- These interface registers are also called latch or buffer.
- All the stages in the pipeline along with the interface registers are controlled by a common clock.

Instruction Pipelining

- To improve the performance of a CPU we have two options:
- 1) Improve the hardware by introducing faster circuits.
- 2) Arrange the hardware such that more than one operation can be performed at the same time.
- Since there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option.
- *Wrt. 8086*
- *16 instructions are pipelined*
- *FIFO*
- *Pipeline will be dumped in case of a branch instruction.*
- *Next instructions from different location will be fetched.*



IF-Instruction Fetch
ID instruction Decoded
EX-Execute instruction
Mem-Memory
WB-Write Back

Instruction Pipelining

- **Stage 1 (Instruction Fetch)** In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.
- **Stage 2 (Instruction Decode)** In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.
- **Stage 3 (Instruction Execute)** In this stage, ALU operations are performed.
- **Stage 4 (Memory Access)** In this stage, memory operands are read and written from/to the memory that is present in the instruction.
- **Stage 5 (Write Back)** In this stage, computed/fetched value is written back to the register present in the instructions.

- *E.g. 0FFFC ADD A,B*
- *PC(Address) – Instruction decoded- operand fetched from reg B- Instruction executed (A+B)– Result written in Reg A*

Co-processors

- Maths Co-Processor-8087
- I/O Co-Processor-8089
- (Master-Slave Concept)

Pipeline processors

- A pipelined processor is **one whose computational capabilities are divided into several sequential stages, each of which may be working with an independent set of data at the same instant of time**.
- Such processors are capable of handling large streams of data at very high rates.

RISC and CISC computers

- **Reduced Instruction Set Architecture (RISC) –**
  The main idea behind this is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.
- **Complex Instruction Set Architecture (CISC) –**
  The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it's complex.
- Both approaches try to increase the CPU performance
- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.

- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

Earlier when programming was done using assembly language, a need was felt to make instruction do more tasks because programming in assembly was tedious and error-prone due to which CISC architecture evolved but with the uprise of high-level language dependency on assembly reduced RISC architecture prevailed.

RISC and CISC computers

Characteristic of RISC –

- Simpler instruction, hence simple instruction decoding.
- Instruction comes undersize of one word.
- Instruction takes a single clock cycle to get executed.
- More general-purpose registers.
- Simple Addressing Modes.
- Fewer Data types.
- A pipeline can be achieved.

Characteristic of CISC –

- Complex instruction, hence complex instruction decoding.
- Instructions are larger than one-word size.
- Instruction may take more than a single clock cycle to get executed.
- Less number of general-purpose registers as operations get performed in memory itself.
- Complex Addressing Modes.
- More Data types.

Difference between RISC and CISC Processors

RISC	CISC
Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and microprogrammed control unit
Transistors are used for more registers	Transistors are used for storing complex Instructions
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small
An instruction executed in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word	Instructions are larger than the size of one word