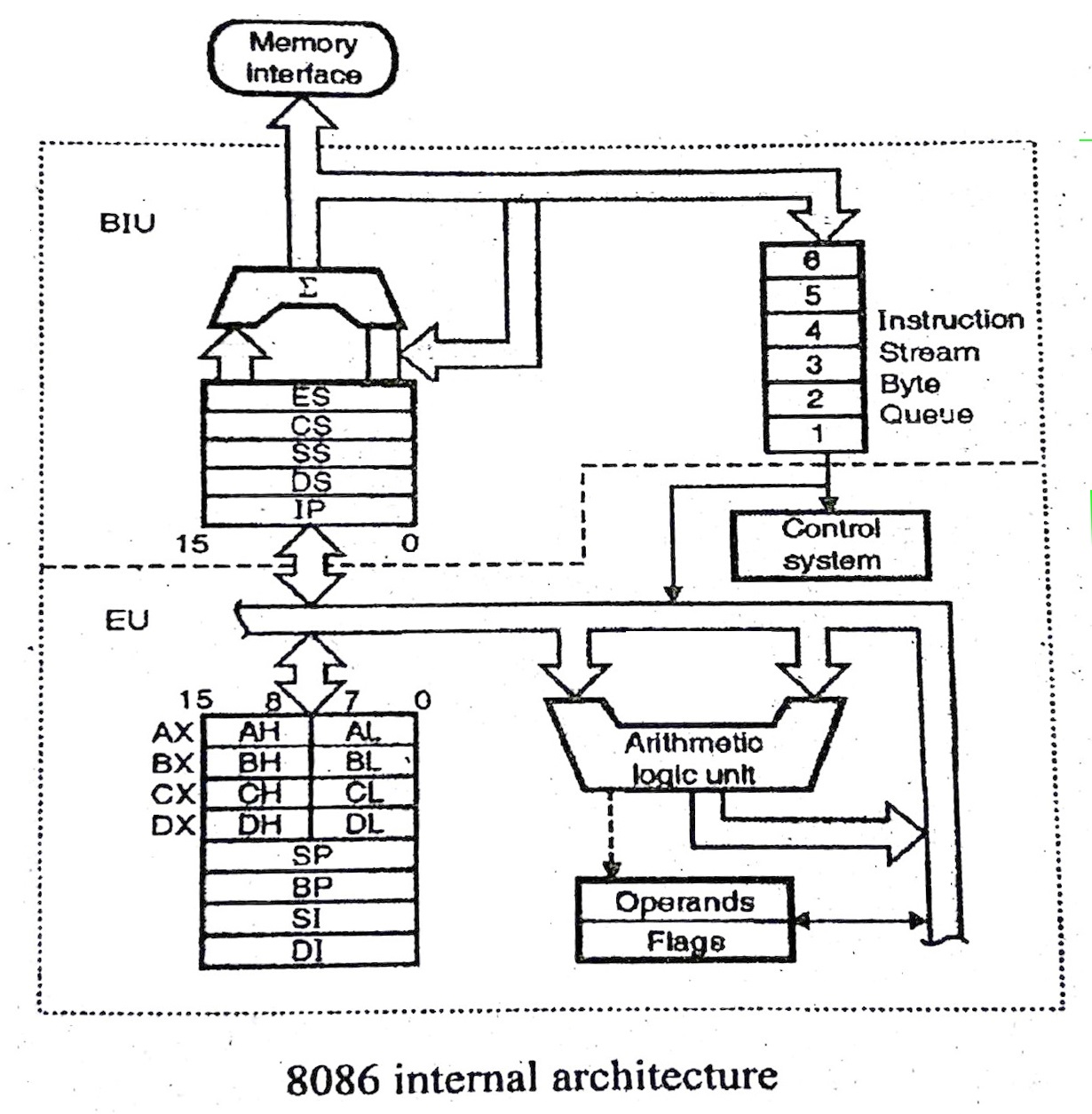
Mid term

1. Explain the internal architecture of the 8086 microprocessor using a block diagram



The internal architecture of Intel 8086 is divided into 2 units: **The Bus Interface Unit (BIU)**, and **The Execution Unit (EU)**. These are explained as following below.

### 1. The Bus Interface Unit (BIU):

.

BIU mainly contains the **4 Segment registers**, the **Instruction Pointer**, a pre-fetch queue, and an **Address Generation Circuit**.

**Instruction Pointer (IP):**

* It is a *16-bit register*. It holds offset of the next instructions in the*Code Segment.*
* IP is incremented after every instruction byte is fetched.
* IP gets a new value whenever a branch instruction occurs.
* CS is multiplied by 10H to give the 20-bit physical address of the Code Segment.
* The address of the next instruction is calculated by using the formula CS x 10H + IP.

= 1000H

then CS x 10H = 43210H + offset = ***44210H***

Here Offset = Instruction Pointer(IP)

.

**Code Segment register: (16 Bit register):**CS holds the base address for the Code Segment. All programs are stored in the Code Segment and accessed via the IP.

**Data Segment register: (16 Bit register):**DS holds the base address for the Data Segment.

**Stack Segment register: (16 Bit register):**SS holds the base address for the Stack Segment.

**Extra Segment register:** **(16 Bit register):**ES holds the base address for the Extra Segment.

*Please note that segments are present in memory and segment registers are present in Microprocessor.  
Segment registers store starting address of each segments in memory.*

*Physical Address = Segment Address x 10H + Offset Address*

**6 Byte Pre-fetch Queue:**

* It is a 6-byte queue (FIFO).
* Fetching the next instruction (by BIU from CS) while executing the current instruction is called pipelining.
* Gets flushed whenever a branch instruction occurs.
* The pre-Fetch queue is of 6-Bytes only because the maximum size of instruction that can have in 8086 is 6 bytes. Hence to cover up all operands and data fields of maximum size instruction in 8086 Microprocessor there is a Pre-Fetch queue is 6 Bytes.
* The pre-Fetch queue is connected with the control unit which is responsible for decoding op-code and operands and telling the execution unit what to do with the help of timing and control signals.
* The pre-Fetch queue is responsible for pipelining and because of that 8086 microprocessor is called fetch, decode, execute type microprocessor. Since there are always instructions present for decoding and execution in this queue the speed of execution in the microprocessor is gradually increased.
* **When there is a 2-byte space in the instruction pre-fetch queue then only the next instruction will be pushed into the queue** otherwise if only a 1-byte space is vacant then there will not be any allocation in the queue. It will wait for a spacing of 2 bytes in subsequent queue decoding operations.
* Instruction pre-fetch queue works in a sequential manner so if there is any branch condition then in that situation pre-fetch queue fails. Hence to avoid chaos instruction queue is flushed out when any branch or conditional jumps occur.

### 2.prefetch unit:

The Prefetch Unit in the 8086 microprocessor is a component responsible for fetching instructions from memory and storing them in a queue. The prefetch unit allows the 8086 to perform multiple instruction fetches in parallel, improving the overall performance of the microprocessor.

The prefetch unit consists of a buffer and a program counter that are used to fetch instructions from memory. The buffer stores the instructions that have been fetched and the program counter keeps track of the memory location of the next instruction to be fetched. The prefetch unit fetches several instructions ahead of the current instruction, allowing the 8086 to execute instructions from the buffer rather than from memory.

This parallel processing of instruction fetches helps to reduce the wait time for memory access, as the 8086 can continue to execute instructions from the buffer while it waits for memory access to complete. This results in improved overall performance, as the 8086 is able to execute more instructions in a given amount of time.

The prefetch unit is an important component of the 8086 microprocessor, as it allows the microprocessor to work more efficiently and perform more instructions in a given amount of time. This improved performance helps to ensure that the 8086 remains competitive in its performance and capabilities, even as technology continues to advance.

### 3. The Execution Unit (EU):

The main components of the EU are General purpose registers, the ALU, Special purpose registers, the Instruction Register and Instruction Decoder, and the Flag/Status Register.

1. Fetches instructions from the Queue in BIU, decodes, and executes arithmetic and logic operations using the ALU.
2. Sends control signals for internal data transfer operations within the microprocessor.(Control Unit)
3. Sends request signals to the BIU to access the external module.
4. It operates with respect to T-states (clock cycles) and not machine cycles.

[8086 has four 16-bit general purpose registers](https://www.geeksforgeeks.org/general-purpose-registers-8086-microprocessor/)

AX, BX, CX, and DX which store intermediate values during execution. Each of these has two 8-bit parts (higher and lower).

* **AX register:** **(Combination of AL and AH Registers)**  
  It holds operands and results during multiplication and division operations. Also an accumulator during String operations.
* **BX register: (Combination of BL and BH Registers)**  
  It holds the memory address (offset address) in indirect addressing modes.
* **CX register: (Combination of CL and CH Registers)**  
  It holds the count for instructions like a loop, rotates, shifts and string operations.
* **DX register: (Combination of DL and DH Registers)**  
  It is used with AX to hold 32-bit values during multiplication and division.

**Arithmetic Logic Unit (16-bit):**Performs**8 and 16-bit**arithmetic and logic operations.

**Special purpose registers (16-bit):**

 Special purpose registers are called Offset registers also. Which points to specific memory locations under each segment.

We can understand the concept of segments as Textbook pages. Suppose there are 10 chapters in one textbook and each chapter takes exactly 100 pages. So the book will contain 1000 pages. Now suppose we want to access page number 575 from the book then 500 will be the segment base address which can be anything in the context of microprocessors like Code, Data, Stack, and Extra Segment. So 500 will be segment registers that are present in Bus Interface Unit (BIU). And 500 + 75 is called an offset register through which we can reach on specific page number under a specific segment.

*Hence 500 is the segment base address and 75 is an offset address or (Instruction Pointer, Stack Pointer, Base Pointer, Source Index, Destination Index) any of the above according to their segment implementation.*

* **Stack Pointer:** Points to Stack top. Stack is in Stack Segment, used during instructions like PUSH, POP, CALL, RET etc.
* **Base Pointer:** BP can hold the offset addresses of any location in the stack segment. It is used to access random locations of the stack.
* **Source Index:** It holds offset address in Data Segment during string operations.
* **Destination Index:**It holds offset address in Extra Segment during string operations.

**Instruction Register and Instruction Decoder:**

The EU fetches an opcode from the queue into the instruction register. The instruction decoder decodes it and sends the information to the control circuit for execution.

[**Flag/Status register (16 bits)**](https://www.geeksforgeeks.org/flag-register-8086-microprocessor/)**:**

It has 9 flags that help change or recognize the state of the microprocessor.

**6 Status flags:**

1. Carry flag(CF)
2. Parity flag(PF)
3. Auxiliary carry flag(AF)
4. Zero flag(Z)
5. Sign flag(S)
6. Overflow flag (O)

Status flags are updated after every arithmetic and logic operation.

### 5.control unit :

The Control Unit in the 8086 microprocessor is a component that manages the overall operation of the microprocessor. The control unit is responsible for controlling the flow of instructions through the microprocessor and coordinating the activities of the other components, including the Decode Unit, Execution Unit, and Prefetch Unit.

### The 8086 microprocessor uses three different buses to transfer data and instructions between the microprocessor and other components in a computer system. These buses are:

**1.Address Bus:** The address bus is used to send the memory address of the instruction or data being read or written. The address bus is 16 bits wide, allowing the 8086 to address up to 64 kilobytes of memory.

**2.Data Bus:** The data bus is used to transfer data between the microprocessor and memory. The data bus is 16 bits wide, allowing the 8086 to transfer 16-bit data words at a time.

**3.Control Bus:** The control bus is used to transfer control signals between the microprocessor and other components in the computer system. The control bus is used to send signals such as read, write, and interrupt requests, and to transfer status information between the microprocessor and other components.

The buses in the 8086 microprocessor play a crucial role in allowing the microprocessor to access and transfer data from memory, as well as to interact with other components in the computer system. The 8086’s ability to use these buses efficiently and effectively helps to ensure that it remains competitive in its performance and capabilities, even as technology continues to advance.

b. What does addressing mode mean? Discuss the several addressing modes used by the 8086 microprocessor.

Addressing mode mean: An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.

The different ways in which a source operand is denoted in an instruction is known as **addressing modes**. There are 8 different addressing modes in 8086 programming −

## Immediate addressing mode

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

### **Example**

MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH

## Register addressing Mode

It means that the register is the source of an operand for an instruction.

### **Example**

MOV CX, AX ; copies the contents of the 16-bit AX register into

; the 16-bit CX register),

ADD BX, AX

## Direct addressing mode

The addressing mode in which the effective address of the memory location is written directly in the instruction.

### **Example**

MOV AX, [1592H], MOV AL, [0300H]

## Register indirect addressing mode

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

### **Example**

MOV AX, [BX] ; Suppose the register BX contains 4895H, then the contents

; 4895H are moved to AX

ADD CX, {BX}

## Based addressing mode

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

### **Example**

MOV DX, [BX+04], ADD CL, [BX+08]

## Indexed addressing mode

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

### **Example**

MOV BX, [SI+16], ADD AL, [DI+16]

## Based-index addressing mode

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

### **Example**

ADD CX, [AX+SI], MOV AX, [AX+DI]

## Based indexed with displacement mode

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

### **Example**

MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]

c. What are machine language and high level language?

A **machine language** is the only language that a computer directly understands, it is usually written in zeros (0) and ones (1). A program instruction in machine language may look something like this 111010110010001 whereas, a **high-level language** is a programming language that uses English and mathematical symbols, like +, -, % and many others, in its instructions.

1. what is the Flag register? Discuss the role of five flag registers in an ALU operation using an example. Show ans 3.a.

The **flag register** is a special register. It is an 8-bit, 16-bit, or 32-bit register and does not make any sense as a unit. Instead, each bit has the same meaning. The flag register bits operate independently; their combined value has no meaning.

Sure! Let's discuss the role of five common flag registers (Zero Flag, Carry Flag, Sign Flag, Overflow Flag, and Parity Flag) in ALU (Arithmetic Logic Unit) operations using an example of addition.

Consider a simple ALU that performs 8-bit addition. The ALU takes two 8-bit input operands, adds them together, and produces an 8-bit result.

Zero Flag (ZF):

The Zero Flag is set if the result of the addition operation is zero. In our example, if the result of the addition is 0, the Zero Flag will be set. This flag helps in checking if the addition resulted in a zero value, which can be useful for conditional branching or comparisons.

Carry Flag (CF):

The Carry Flag is set if there is a carry-out or borrow from the most significant bit (MSB) during addition. In our example, if the addition operation results in a carry from the MSB, the Carry Flag will be set. This flag is used to handle carry or overflow conditions in multi-byte arithmetic operations.

Sign Flag (SF):

The Sign Flag reflects the sign of the result. In our example, if the result of the addition is negative (the MSB is 1), the Sign Flag will be set. This flag helps determine the sign of the result and can be used for conditional checks or signed arithmetic operations.

Overflow Flag (OF):

The Overflow Flag is set if the addition operation produces a result that is too large to be represented within the available number of bits. In our example, if the addition operation causes an overflow, the Overflow Flag will be set. This flag helps detect signed arithmetic overflow conditions.

Parity Flag (PF): The Parity Flag indicates whether the result of the addition operation has an even or odd number of set bits. In our example, if the addition operation results in an even number of set bits in the result, the Parity Flag will be set. This flag can be used for parity checking or error detection.

Let's say we want to add two 8-bit values: A = 11001101 and B = 10101010.

1. Zero Flag (ZF): If the result of the addition (A + B) is 00000000, the Zero Flag will be set, indicating that the result is zero.
2. arry Flag (CF): If there is a carry-out from the MSB during the addition, the Carry Flag will be set. For example, if A = 11110000 and B = 00001111, the addition (A + B) will result in a carry-out, and the Carry Flag will be set.
3. Sign Flag (SF): If the MSB of the result is 1, indicating a negative value, the Sign Flag will be set. For instance, if A = 10000001 and B = 01111111, the addition (A + B) will yield a negative result, and the Sign Flag will be set.
4. Overflow Flag (OF): If the addition operation causes an overflow, such as when adding two positive numbers resulting in a negative value or adding two negative numbers resulting in a positive value, the Overflow Flag will be set. For example, if A = 01111111 and B = 01111111, the addition (A + B) will result in an overflow, and the Overflow Flag will be set.
5. Parity Flag (PF): If the result of the addition operation has an even number of set bits, the Parity Flag will be set. For example, if A = 11001100 and B = 00110011, the addition (A + B) will yield a result with an even number of

b. Draw the block diagram of 8086 microcomputer system overview and explain how data is transferred with in the 8086 microcomputer system.show the answer in 1.b.

## 

## Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

### **Instruction to transfer a word**

* **MOV** − Used to copy the byte or word from the provided source to the provided destination.
* **PPUSH** − Used to put a word at the top of the stack.
* **POP** − Used to get a word from the top of the stack to the provided location.
* **PUSHA** − Used to put all the registers into the stack.
* **POPA** − Used to get words from the stack to all registers.
* **XCHG** − Used to exchange the data from two locations.
* **XLAT** − Used to translate a byte in AL using a table in the memory.

1. What is memory segmentation ? how different memory segments are accessed in 8086 microprocessor? Show the ans in 3.c.

**Definition**: [Memory](https://ecomputernotes.com/fundamental/input-output-and-memory/memory) segmentation is the process in which we divide the primary [memory](https://ecomputernotes.com/fundamental/input-output-and-memory/memory) of the [computer](https://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) into segments. It is a memory management technique. Each segment can be allocated to a process

**Segmentation** is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast.

**Need for Segmentation –**  
The Bus Interface Unit (BIU) contains four 16 bit special purpose registers (mentioned below) called as Segment Registers.

* **Code segment register (CS):** is used for addressing memory location in the code segment of the memory, where the executable program is stored.
* **Data segment register (DS):** points to the data segment of the memory where the data is stored.
* **Extra Segment Register (ES):** also refers to a segment in the memory which is another data segment in the memory.
* **Stack Segment Register (SS):** is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.

3.2 (2021) Solution

## What is Microprocessor?

A type of computer processors that incorporates [the functions of a computer’s center processing unit (CPU)](https://onlineclassnotes.com/what-is-cpu-what-are-functions-of-cpu/) onto a single integrated circuit is called a microprocessor

1.a. What are the function of microprocessor?

Among various functions of microprocessor some are as follows

1. Controlling all other parts of the machine and sending timing signals.
2. Transferring data between memory and I/O devices
3. Fetching data and instructions from memory
4. Decoding instruction
5. Performing arithmetical and logical operations
6. Executing programs stored in memory
7. Performing communication among the I/O devices etc.

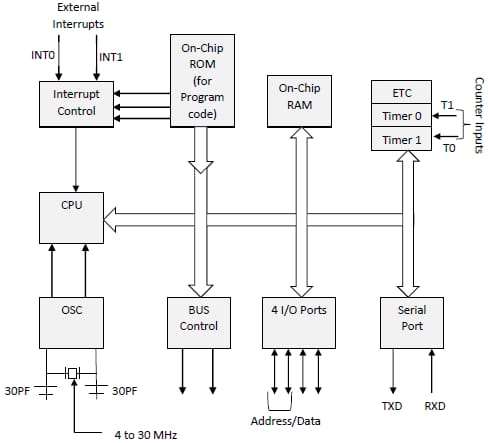
## What is Microcontroller?

A **microcontroller** is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave’s information, receiving remote signals, etc.

a.Difference between Microprocessor and Microcontroller

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Microprocessor** | **Microcontroller** |
| **Definition** | Microprocessors can be understood as the heart of a computer system. | Microcontrollers can be understood as the heart of an embedded system. |
| **What is it?** | A microprocessor is a processor where the memory and I/O component are connected externally. | A microcontroller is a controlling device wherein the memory and I/O output component are present internally. |
| **Circuit complexity** | The circuit is complex due to external connection. | Microcontrollers are present on chip memory. The circuit is less complex. |
| **Memory and I/O components** | The memory and I/O components are to be connected externally. | The memory and I/O components are available. |
| **Compact system compatibility** | Microprocessors can’t be used in compact system. | Microcontrollers can be used with a compact system. |
| **Efficiency** | Microprocessors are not efficient. | Microcontrollers are efficient. |
| **Zero status flag** | Microprocessors have a zero status flag. | Microcontroller doesn’t have a zero status flag. |
| **Number of registers** | Microprocessors have less number of registers. | Microcontrollers have more number of registers. |
| **Applications** | Microprocessors are generally used in personal computers. | Microcontrollers are generally used in washing machines, and air conditioners. |

b. Draw block diagram of microprocessor 8086



# **c.Different addressing modes**

# **Microprocessor - 8086 Addressing Modes**

2.a What do you mean by 8 bit and 16 bit microprocessor?

The main difference between an 8-bit and a 16-bit microprocessor lies in their data bus width and instruction set architecture. Here are the key distinctions:

1. Data Bus Width:
   * 8-Bit Microprocessor: An 8-bit microprocessor has an 8-bit data bus, which means it can process and transfer data in 8-bit chunks. This limits the maximum amount of data that can be processed or transferred in a single operation to 8 bits.
   * 16-Bit Microprocessor: A 16-bit microprocessor has a 16-bit data bus, allowing it to process and transfer data in 16-bit chunks. This doubles the amount of data that can be processed or transferred compared to an 8-bit microprocessor.
2. Instruction Set Architecture:
   * 8-Bit Microprocessor: An 8-bit microprocessor typically has a simpler instruction set architecture with instructions that operate on 8-bit data. The instructions are designed to work with smaller data sizes, limiting the range and complexity of operations that can be performed.
   * 16-Bit Microprocessor: A 16-bit microprocessor has a more advanced instruction set architecture that supports operations on 16-bit data. It provides a broader range of instructions and can handle more complex operations and calculations.
3. Memory Addressing:
   * 8-Bit Microprocessor: An 8-bit microprocessor has a limited address space, often with a maximum addressable memory of 64 KB (2^16). This limits the amount of memory that can be directly accessed by the microprocessor without additional techniques like bank switching.
   * 16-Bit Microprocessor: A 16-bit microprocessor has a larger address space, typically supporting up to 1 MB (2^20) or more of directly addressable memory. This allows for more extensive memory access without the need for additional techniques.
4. Performance:
   * 8-Bit Microprocessor: Due to its narrower data bus and simpler architecture, an 8-bit microprocessor usually has lower processing power and throughput compared to a 16-bit microprocessor. It is more suitable for simple applications with less demanding computational requirements.
   * 16-Bit Microprocessor: With a wider data bus and more advanced architecture, a 16-bit microprocessor generally offers higher performance and better computational capabilities. It is capable of handling more complex tasks and executing instructions at a faster rate.
5. Cost and Power Consumption:
   * 8-Bit Microprocessor: 8-bit microprocessors are often more cost-effective and have lower power consumption compared to their 16-bit counterparts. They are commonly used in cost-sensitive and power-constrained applications where lower performance is acceptable.
   * 16-Bit Microprocessor: 16-bit microprocessors are typically more expensive and consume more power than 8-bit microprocessors. They are preferred for applications that require higher performance, larger memory addressing, and more advanced operations.

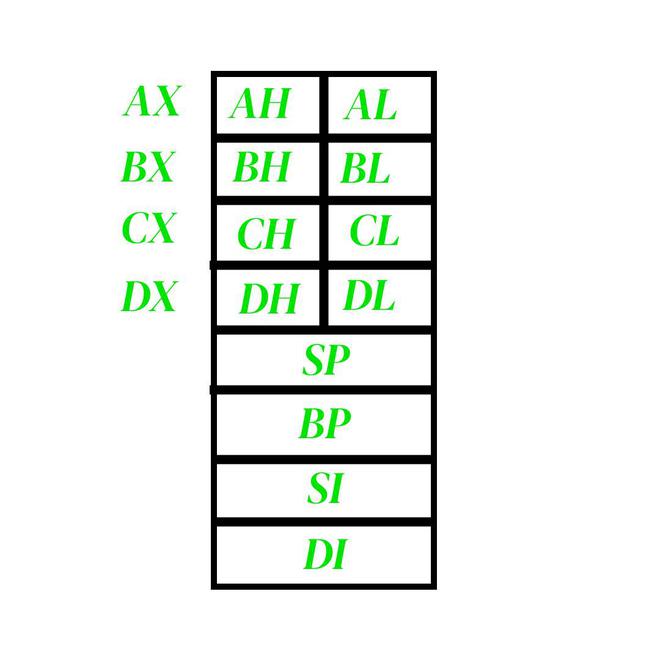
##Difference between 8-Bit and 16-Bit Microcontrollers

The following table highlights all the important differences between 8-bit microcontroller and 16-bit microcontroller −

|  |  |  |
| --- | --- | --- |
| **Parameter** | **8-Bit Microcontroller** | **16-Bit Microcontroller** |
| **Definition** | A type of microcontroller that is capable of processing 8-bits of data at a time is called an 8-bit microcontroller. | A type of microcontroller that is capable of processing 16-bits of data at a time is called a 16-bit microcontroller. |
| **Data width** | 8-bit microcontroller has a data width equal to 8-bits. Hence, its CPU can process only 8-bit in parallel. | 16-bit microcontroller has a data width equal to 16-bits, which means its CPU can process 16-bits of data in parallel. |
| **Processing power** | 8-bit microcontrollers have low processing power; hence they can handle only simple instructions. | 16-bit microcontrollers have high processing power; thus, they are capable of handling complex instructions. |
| **Memory capacity** | 8-bit microcontroller has low memory capacity. | 16-bit microcontroller has high storage capacity. |
| **Clock speed** | 8-bit microcontrollers have low clock speeds. Hence, they process data slowly. | 16-bit microcontrollers have higher clock speeds, allowing them to process data faster. |
| **Complexity** | 8-bit microcontrollers have comparatively less complex system architecture. | 16-bit microcontrollers have complex system architecture. |
| **Programming language** | 8-bit microcontrollers are generally programmed using low-level assembly language and high-level C language. | 16-bit microcontrollers are programmed using high-level languages such as C, C++. |
| **Power consumption** | 8-bit microcontrollers consume less power. | 16-bit microcontrollers consume more power than 8-bit microcontrollers. |
| **Instruction set** | 8-bit microcontrollers have simple instruction sets, making them suitable to execute simple operations. | 16-bit microcontrollers have complex instruction sets. Hence, they are suitable to execute complex operations. |
| **Efficiency** | 8-bit microcontrollers have low efficiency. | 16-bit microcontrollers are more efficient than 8-bit microcontrollers. |
| **Suitability** | 8-bit microcontrollers are suitable for simple and small size applications. | 16-bit microcontrollers are suitable for complex and large size applications. |
| **Number of I/O** | 8-bit microcontrollers support lesser number of I/O peripherals. | 16-bit microcontrollers support more I/O peripherals. |
| **Range of values** | In 8-bit microcontroller, a range of values from 0 to 255 can be handled by an instruction cycle. | In 16-bit microcontroller, a range of values from 0 to 65,535 can be handled by an instruction cycle. |
| **Price** | 8-bit microcontrollers are less expensive. | 16-bit microcontrollers are more expensive. |
| **Examples** | Examples of most popular 8-bit microcontrollers are Microchip’s PIC series, Atmel’s AVR series, Intel 8031, Intel 8051, etc. | Examples of most popular 16-bit microcontrollers are Intel 8096, Microchip PIC2x, Infineon CX16x series. |
| **Applications** | 8-bit microcontrollers are mainly used in simple applications like home and office appliances, medical instruments, toys, etc. | 16-bit microcontrollers are mainly used in complex applications like in industrial automation, robotics, control systems, automobiles, telecommunication systems, etc |

B. Describe the general purpose registers

General-purpose registers are used to store temporary data within the microprocessor. There are 8 general-purpose registers in the 8086 microprocessor.



**1. AX:** This is the accumulator. It is of 16 bits and is divided into two 8-bit registers AH and AL to also perform 8-bit instructions. It is generally used for arithmetical and logical instructions but in 8086 microprocessor it is not mandatory to have an accumulator as the destination operand. Example:

ADD AX, AX (AX = AX + AX)

**2. BX:**This is the base register. It is of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions. It is used to store the value of the offset. Example:

MOV BL, [500] (BL = 500H)

**3. CX:** This is the counter register. It is of 16 bits and is divided into two 8-bit registers CH and CL to also perform 8-bit instructions. It is used in looping and rotation. Example:

MOV CX, 0005

LOOP

**4. DX:** This is the data register. It is of 16 bits and is divided into two 8-bit registers DH and DL to also perform 8-bit instructions. It is used in the multiplication and input/output port addressing. Example:

MUL BX (DX, AX = AX \* BX)

**5. SP:** This is the stack pointer. It is of 16 bits. It points to the topmost item of the stack. If the stack is empty the stack pointer will be (FFFE)H. Its offset address is relative to the stack segment.

**6. BP –** This is the base pointer. It is of 16 bits. It is primarily used in accessing parameters passed by the stack. Its offset address is relative to the stack segment.

**7. SI –** This is the source index register. It is of 16 bits. It is used in the pointer addressing of data and as a source in some string-related operations. Its offset is relative to the data segment.

**8. DI –** This is the destination index register. It is of 16 bits. It is used in the pointer addressing of data and as a destination in some string-related operations. Its offset is relative to the extra segment.

2.c. What happens when the microprocessor is powered up

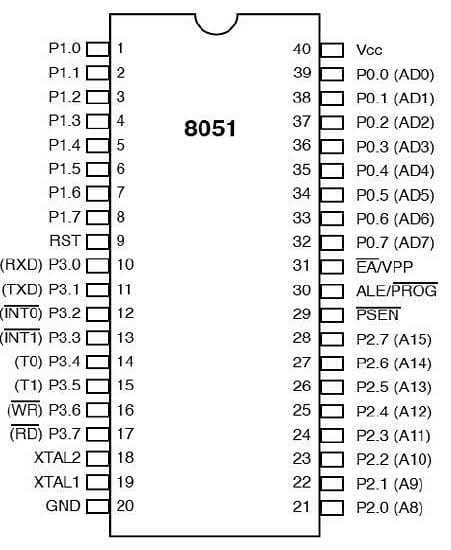
ChatGPT

When a microprocessor is powered up, several key events occur:

1. Reset: The microprocessor enters a reset state, where it initializes its internal registers and sets its program counter (PC) to a predefined memory address. The reset state ensures that the microprocessor starts executing instructions from a known starting point.
2. Fetch-Decode-Execute Cycle: The microprocessor begins executing instructions by following the fetch-decode-execute cycle. It fetches an instruction from memory based on the value in the PC, decodes the instruction to determine its meaning, and then executes the instruction.
3. Clock Synchronization: The microprocessor relies on a clock signal to synchronize its internal operations. The power-up process typically involves establishing a stable clock signal, which determines the rate at which instructions are executed.
4. Configuration: Some microprocessors may require configuration or initialization steps to set parameters such as cache sizes, memory settings, and input/output configurations. These settings are often done through registers or configuration pins specific to the microprocessor architecture.
5. Peripheral Initialization: If the microprocessor interfaces with external devices or peripherals, such as memory modules, input/output devices, or communication interfaces, it may initialize and configure those peripherals during the power-up process.
6. Self-Test and Diagnostics: Some microprocessors have built-in self-test routines that check the integrity and functionality of various internal components. These tests help ensure that the microprocessor is operating correctly and can identify potential issues.
7. Operating System Boot: In systems where a microprocessor is part of a larger computing device, such as a computer or embedded system, the power-up process may involve loading and booting the operating system. The operating system takes control of the microprocessor, provides higher-level functionality, and prepares the system for user interaction.

3.a. what is the flag register? discuss the role of five flag register is in ALU operation using an example

3.b. draw pin diagram of 8086 microcomputer



3.b. how data is transferred with in the 8086 microcomputer system

3.c. What is memory segmentation? how different memory segments are accessed in 8086

Ans in before.

4.A. give the main task of AX and IP registers for 8086 micro microprocessor

In the Intel 8086 microprocessor, the AX (Accumulator) and IP (Instruction Pointer) registers serve important roles in the execution of instructions and data manipulation.

1. AX (Accumulator) Register: The AX register is a 16-bit general-purpose register used for various arithmetic and data manipulation operations. It is also known as the primary accumulator. The main tasks of the AX register include:

* Arithmetic Operations: The AX register is involved in arithmetic operations such as addition, subtraction, multiplication, and division. Many arithmetic instructions, like ADD, SUB, MUL, and DIV, use the AX register implicitly or explicitly to store the results.
* Data Movement: The AX register is used for moving data between memory locations and other registers. It is commonly used to transfer data between the microprocessor and memory or between different registers.
* Data Storage: The AX register can store temporary data during the execution of instructions or hold intermediate results during computations.
* Return Values: In function calls and subroutines, the AX register is often used to hold the return value from a function.
* Special Purpose: In some instructions, like the "LOOP" instruction, the CX (Counter) register decrements while the AX register is used to compare and control loop iterations.

1. IP (Instruction Pointer) Register: The IP register is a 16-bit register that stores the offset or memory address of the next instruction to be executed in the code segment (CS). The main task of the IP register is to keep track of the current instruction being executed. When an instruction is fetched and executed, the IP is automatically incremented to point to the next instruction. The IP register works in conjunction with the CS segment register to determine the physical address of the next instruction.

* Instruction Pointer Management: The IP register is automatically incremented during the instruction fetch and execution cycle. It ensures that the processor always knows the location of the next instruction to be executed.
* Program Flow Control: By modifying the value of the IP register, a programmer can change the program flow, implement loops, conditionals, and other control structures.
* Branching and Jumping: The IP register is crucial for executing jump and branch instructions, allowing the program to jump to different parts of the code based on specific conditions.
* Program Execution: The IP register plays a fundamental role in executing the program's instructions in sequence, step by step.

4.b. find the memory address of the next instruction executed by the micro processor, when operated in the real mode, for the CS=2301 Hand IP=75FDH

ChatGPT

In real mode of the Intel 8086 microprocessor, the physical memory address of the next instruction to be executed can be calculated using the CS (Code Segment) and IP (Instruction Pointer) registers.

Given CS = 2301H and IP = 75FDH, the physical memory address of the next instruction can be calculated as follows:

1. Concatenate CS and IP to form a 20-bit physical address: Physical Address = (CS << 4) + IP
2. Convert CS and IP to decimal values: CS = 2301H = 8961 (in decimal) IP = 75FDH = 30109 (in decimal)
3. Calculate the physical address: Physical Address = (8961 << 4) + 30109 Physical Address = 143376 + 30109 Physical Address = 173485

Therefore, when operated in real mode with CS = 2301H and IP = 75FDH, the physical memory address of the next instruction to be executed by the microprocessor is 173485.

4.b find the memory address of the next instruction executed by the micro processor, when operated in the real mode, for the CS=3476H Hand IP=IA00H

ChatGPT

To find the memory address of the next instruction executed by the microprocessor in real mode, we need to calculate the physical address using the CS (Code Segment) and IP (Instruction Pointer) registers.

Given CS = 3476H and IP = IA00H, let's proceed with the calculation:

1. Concatenate CS and IP to form a 20-bit physical address: Physical Address = (CS << 4) + IP
2. Convert CS and IP to decimal values: CS = 3476H = 13430 (in decimal) IP = IA00H = 40960 (in decimal)
3. Calculate the physical address: Physical Address = (13430 << 4) + 40960 Physical Address = 214880 + 40960 Physical Address = 255840

Therefore, when operated in real mode with CS = 3476H and IP = IA00H, the physical memory address of the next instruction to be executed by the microprocessor is 255840.

4.c. why we need initialize data segment

ChatGPT

The data segment, also known as the initialized data segment or simply the "data segment," is a specific area of a computer program's memory where initialized global and static variables are stored. When a program is loaded into memory, the data segment is allocated to hold the initial values of these variables.

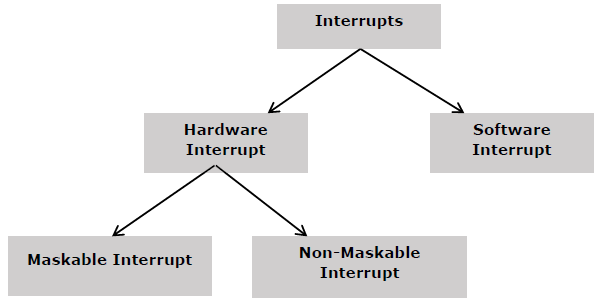
Here are a few reasons why initializing the data segment is important:

1. Variable Initialization: Initializing variables ensures that they have defined values from the start of the program. This is crucial because variables without initial values can lead to unpredictable behavior and bugs in the program. By initializing variables in the data segment, you can establish a known starting state for your program.
2. Consistency: The data segment allows you to maintain consistency in the values of global and static variables across different parts of your program. If you declare a variable in one file and access it in another, initializing it in the data segment ensures that it has the same initial value regardless of where it is accessed.
3. Efficiency: Initializing variables in the data segment is typically more efficient than initializing them at runtime. When the program is loaded, the initialized values are already stored in memory, eliminating the need for additional instructions or computations to set them to their initial states.
4. Readability and Maintainability: By explicitly initializing variables in the data segment, it becomes easier to understand and maintain the program. The initial values are visible and can be modified directly if needed, making the code more readable and less error-prone.

5.a. what is interrupt in microprocessor?

**Interrupt** is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in a 8086 microprocessor −



* 1. Maskable and Non-Maskable Interrupts

Here, you will learn about the head-to-head comparison between the Maskable and Non-Maskable Interrupts. Some main differences between the Maskable and Non-Maskable Interrupts are as follows:

|  |  |  |
| --- | --- | --- |
| **Features** | **Maskable Interrupts** | **Non-Maskable Interrupts** |
| **Definition** | Maskable interrupts are those that may be accepted or ignored by the CPU based on their priority. | The CPU must accept the non-maskable interrupts. |
| **Priority** | It aids in the management of low-priority jobs. | It aids in the management of high-priority jobs. |
| **Interrupt Execution** | When a maskable interrupt occurs, the system may manage it after it runs the present instructions. | The system may hold the current state and interrupts in the stack when the non-maskable interrupt occurs, allowing the processor to manage the interrupt. |
| **Processing** | The maskable interrupt is handled immediately if it had a higher priority than the currently running instruction when it occurred. Otherwise, the interrupt is processed after the current execution is completed. | The non-maskable interrupt must be processed immediately by suspending the present execution. |
| **Vector Address** | The vector address of this interrupt might be changed by programming the Interrupt Controller. | The vector address of this interrupt is fixed, and the processor manufacturer predefines it. |
| **Response Time** | It has a higher response time. | It has a very low response time. |
| **Usage** | The device controllers mainly utilize these. | These are mainly utilized by watchdog timers and during power failure. |
| **Examples** | Some common instances of maskable interrupts include the RST6.5, RST7.5, and RST5.5 of the 8085 microprocessor. | The Trap of the 8085 microprocessor is an instance of a non-maskable interrupt. |

* 1. distinguish between logical address and physical address

### Difference between Logical Address and Physical Address in Operating System

|  |  |  |
| --- | --- | --- |
| **S.No** | **Logical Address** | **Physical Address** |
| 1 | Logical address is rendered by CPU. | Physical address is like a location that is present in the main memory. |
| 2 | It is a collection of all logical addresses rendered by the CPU. | It is a collection of all physical addresses mapped to the connected logical addresses. |
| 3 | Logical address of the program is visible to the users. | We cannot view the physical address of the program. |
| 4 | Logical address is generated by the CPU. | Physical address is computed by MMU. |
| 5 | We can easily utilise the logical address to access the physical address. | We can use the physical address indirectly. |

* 1. 1.LEA: The LEA (Load Effective Address) instruction is a way of obtaining the address which arises from any of the Intel processor's memory addressing modes. it moves the contents of the designated memory location into the target register.

2.MOV: The MOV instruction is used to copy the data of item, which is referred to as the second operand of the instruction (Constant value, register content, or memory content), into the location, which is referred to by its first operand (memory or register)

3. PUSH: This is a single-byte instruction. The contents of the register pair specified in the operand are copied into the stack. The stack pointer is decremented and the contents of the higher-order register pair (such as B in BC, D in DE) are copied on the stack.

4.POP: POP − Used to get a word from the top of the stack to the provided location. PUSHA − Used to put all the registers into the stack. POPA − Used to get words from the stack to all registers. XCHG − Used to exchange the data from two locations.

## 6.a. Difference Between Synchronous and Asynchronous Transmission

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Synchronous** | **Asynchronous** |
| 1. | In Synchronous transmission a common clock is shared by the transmitter and receiver to achieve synchronisation while data transmission. | In Asynchronous transmission each character contains its own start and stop bits. |
| 2. | In Synchronous transmission data is sent in frames or blocks. | In Asynchronous transmission data is sent in the form of bytes or characters. |
| 3. | Synchronous transmission is faster, as a common clock is shared by the sender and receiver. | Asynchronous transmission is slower as each character has its own start and stop bit. |
| 4. | Synchronous transmission is costlier. | Asynchronous transmission is cheaper. |
| 5. | It is easy to design. | It is complex. |
| 6. | In synchronous transmission there is no gap between the data as they share a common clock. | In asynchronous transmission there is a gap between the data due to the start and stop bit feature. |

6.b. draw and explain block diagram of microcontroller

**CPU –** The CPU (Central Processing Unit) comprising of ALU (Arithmetic and Logic Units) and Control units

**ALU -** Arithmetic and Logic Unit performing the arithmetic and logical operations. These operations are multiplication, addition, subtraction, logical AND,OR etc. To do these operations one operand must be in Accumulator, and another may in B register or in general purpose register. The ALU operation results are mostly placed in A register. Some results are placed in B register too.

**OSC -**Oscillator provides clock for controller operation .Crystal oscillator is used for providing perfect clock and stability. So this microcontroller uses crystal oscillator. For this purpose, the crystals linked to the pins are used.

**INTERRUPT CONTROLLER**

Microcontroller operation needs some interrupts. Here mainly five interrupts are used. The controller controls the operation interrupts. ie some interrupts may be allowed, some others are disabled and priority assigned and changed. The five interrupts are:

**BUS CONTROL**

For both Address and data, lower byte address buses are used. The BUS control controls the BUS usage. There are 3 control signals: PSEN , EA, and ALE. The signals like Program Store Enable (PSEN), Address Latch Enable (ALE) and External Access (EA),  are used for external memory interfacing.

**ON CHIP RAM**

The 8051 has 4 kilobyte of inbuilt ROM. It is otherwise called program memory. Usually program-code is stored in ROM. To store program into ROM, programmer is needed. If more area is required in ROM, an external ROM may be connected. Maximum of 64kb ROM memory can be used.

**ON CHIP ROM**

The 8051 has an inbuilt of 128 byte RAM, some version have 256 byte and are used as data memory. If the system needs more memory, external RAM may be connected up to 64kb.In 128 byte RAM chip, 00h to 71Fh are the address range .In this range ,00H to 1FH are the general purpose registers . 20H to 2F are the bit Addressable area and rest of this is byte addressable .This is used as general purpose scratch pad. In 256 byte RAM chip, another 128 bytes are used for Special Function Registers.

**I/O PORTS**

There are four IO ports in 8051.These are named as P0, P1, P2, P3.All are bidirectional. Each port have its address, output driver, latch and input buffer. Each port is output by default.

**SERIAL PORT**

TXD and RXD are utilised for serial port. Each pin has separate buffer registers named as SBUF.

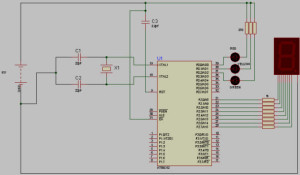
**TIMER/COUNTER**

There are two types of counter/timer in 8051.These counter/timers are name as Timer/Counter 0 and Timer/Counter 1. Each one can be used as either Timer or Counter. 16 bit timer register are used for counting in timer operation or counter operation. Clock pulses are counted in Timer operation and external events are counted in counter operation.

## 6.c. how do you use a microcontroller to design a project traffic signal control?

## Traffic Light Controller using Microcontroller

The main objective of this traffic light controller is to provide sophisticated control and coordination to confirm that traffic moves as smoothly and safely as possible. This project makes use of [LED lights](https://www.watelectronics.com/what-is-rgb-led-circuit-its-working/) for indication purpose and a microcontroller is used for auto changing of signal at specified range of time interval. LED lights gets automatically turns on and off by making corresponding port pin of the microcontroller “HIGH”.



**Circuit Diagram of Traffic Light Controller**

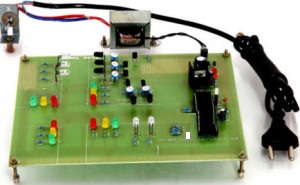
### **Working of the Traffic Light Controller**

In the above circuit diagram of traffic light controller,a seven-segment display is used as a counter display, and three LEDs are used for the purpose of traffic light control. An 8051 Microcontroller is the brain of this whole project and is used to initiate the traffic signal at the intersections on road. This circuit diagram makes use of a crystal oscillator for generating frequency clock pulses. The LEDs are interfaced to the Port zero of the microcontroller and are powered with 5v power supply. Seven-segment display is connected to the port2 pins of the [8051 microcontroller](https://www.watelectronics.com/8051-microcontroller-architecture/)with a common anode configuration.

The LEDs get automatically switched on and off by making the corresponding port pins of the microcontroller high, based on the 8051 microcontroller and its programming done by using KEIL software. At a particular period of time, only the green light holds ON and the other lights remains OFF, and after sometime, the changeover traffic light control from green to red takes place by making the succeeding change for glowing of yellow LED. This process continues as a cycle and the timing for changing the LEDs can be displayed with the use of a seven-segment LED display in this project.

This traffic light control system can be further enhanced in such a way to control the traffic signals automatically based on the traffic density on roads with the help of IR sensor modules with automatic turnoff if there are no vehicles on either side of the road which leads to power consumption.

## Density Based Traffic Signal System



**Prototype of Traffic Light Control System**

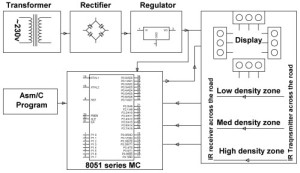
A prototype of traffic light control system is made by using Infra-Red sensors along with major components Microcontroller and LEDs which are used for controlling traffic signals based on the density of the traffic. The junction taken into consideration is a four-side junction with the traffic flow on each side is only in one direction. This traffic light control system consists of the following three main components:

**Display Unit:** It consists of 3 LEDs: Green, Red and Orange on each side of the junction– a total of 12 LEDs of three colors are placed at the junction.

**Detector Unit:** It consists of a combined arrangement of photo diode and IR LED at every junction for detecting the presence of vehicles based on the change in resistance.

**Controller Unit:**This control unit contains a microcontroller which receives the output data from the IR Sensors and controls the glowing of LEDs based on the programming.

The aim of this project is to design and develop a density based dynamic traffic signal system. The signal timing changes automatically by sensing the traffic density at every junction.



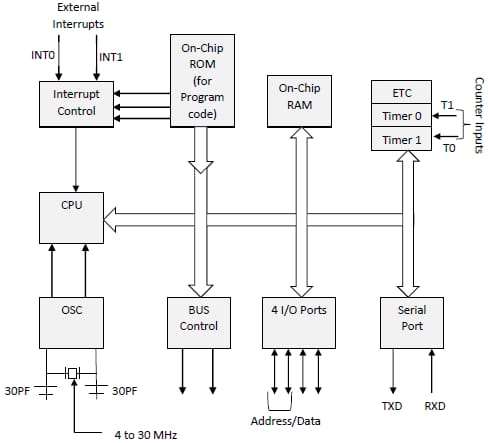
**Block Diagram of Density Based Traffic Signal System**

This traffic light control system works on the concept of fixed time allocation at each side of the junction which cannot be changed as per varying traffic density. Timings allotted at every junction are fixed .Sometimes higher traffic density at one side of the junction demands longer time duration for green signal compared to the standard allotted time.

This proposed system makes use of an 8051 microcontroller which is properly interfaced with the IR sensors to change the junction timing automatically to accommodate the movement of vehicles for avoiding unnecessary waste of time at each junction.

The [sensors used in this project](https://www.watelectronics.com/different-types-of-sensors-with-applications/) IR and photo diodes are in line of sight configuration across the loads to detect the density at the traffic signal. The density of these vehicles is measured in three zones i.e., low, medium and high,respectively – based on certain time duration allotted at respective zones,accordingly.

This article is about a project on traffic light controller and Density based traffic [light system](https://www.watelectronics.com/light-sensor-circuit-working-operation/)used to control the traffic at each junction with the help of microcontroller and LEDs.



7.a. AX,BX where AX contains 8000h and BX contains 0001h and calculate status flag,parity flag,zero flag,carry flag and overflow flag.

To calculate the status flags (sign flag, parity flag, zero flag, carry flag, and overflow flag) after performing specific arithmetic or logical operations with AX and BX, we need to know the exact operation being performed (addition, subtraction, etc.). However, in this scenario, the values of AX and BX are given, and we are not provided with any specific operation.

To determine the status flags, we need to perform some operation with AX and BX and then evaluate the flags. Let's perform a simple addition operation between AX and BX and calculate the status flags accordingly:

Given: AX = 8000h (32768 in decimal) BX = 0001h (1 in decimal)

Step 1: Perform the addition operation: AX + BX = 8000h + 0001h = 8001h (32769 in decimal)

Step 2: Evaluate the status flags based on the result (8001h):

1. Sign Flag (SF): The sign flag is set if the most significant bit (bit 15) of the result is 1 (indicating a negative result). In this case, since the most significant bit of 8001h is 0, the sign flag is cleared (SF = 0).
2. Parity Flag (PF): The parity flag is set if the result has an even number of 1 bits. We need to check the parity of the lower 8 bits of the result (8001h). The binary representation of 8001h is 1000 0000 0000 0001. There is only one 1 bit in the lower 8 bits, so the parity flag is cleared (PF = 0).
3. Zero Flag (ZF): The zero flag is set if the result is zero. In this case, the result (8001h) is not zero, so the zero flag is cleared (ZF = 0).
4. Carry Flag (CF): The carry flag is set if there is a carry-out from the most significant bit during addition. Since there is no carry-out in this addition operation, the carry flag is cleared (CF = 0).
5. Overflow Flag (OF): The overflow flag is set when the result of a signed operation is too large or too small to be represented in the destination register. Since this is an unsigned addition operation, there is no signed overflow, so the overflow flag is cleared (OF = 0).

So, after performing the addition operation between AX (8000h) and BX (0001h), the status flags are as follows:

SF = 0 (Cleared) PF = 0 (Cleared) ZF = 0 (Cleared) CF = 0 (Cleared) OF = 0 (Cleared)

Top of Form

7.c. what do you mean by stack 100h in assembly language

. STACK 100h : is a segment directive which defines 100h words as program STACK. The linker sets the values of SS and SP. . DATA : is a segment directive, followed by one or more data allocation directives to define the variable and constant used by program.

8.a. Write pseudo-code divide -1250 by 7

Here is a pseudo-code algorithm to divide -1250 by 7 using integer division:

Set dividend = -1250

Set divisor = 7

// Handle negative dividend

if dividend < 0 then

Set sign = -1

Set dividend = abs(dividend)

else

Set sign = 1

// Initialize quotient and remainder to 0

Set quotient = 0

Set remainder = 0

// Perform division

for i = 15 to 0 do

Set remainder = remainder << 1

Set remainder[0] = dividend[i] // Get the next bit of the dividend

if remainder >= divisor then

Set remainder = remainder - divisor

Set quotient[i] = 1

// Handle negative quotient

if sign == -1 then

Set quotient = -quotient

// Output quotient

Output quotient

8.b. describe the general format of 16 machine instruction with explanation of each component

ChatGPT

In a 16-bit machine instruction, the general format consists of several components that serve different purposes. Here's an overview of the typical components and their explanations:

1. Opcode (Operation Code): The opcode specifies the operation to be performed by the instruction. It determines the type of operation, such as arithmetic, logical, branching, or data transfer. The opcode is usually represented by a binary code.
2. Operand(s): The operand(s) provide the data or address on which the operation is performed. They can be registers, memory addresses, immediate values, or a combination of these.
3. Register Designators: Register designators indicate the specific registers involved in the instruction. These can be source registers, destination registers, or both, depending on the operation. The designators could be binary codes representing registers, such as R0, R1, R2, etc.
4. Immediate Values: Immediate values are constants or data directly included in the instruction itself. They are often used as operands for immediate arithmetic or logical operations, and they can be represented as binary or decimal values.
5. Addressing Modes: Addressing modes determine how the operand(s) are accessed. They specify the location of the data in memory or the registers used. Some common addressing modes include direct addressing, indirect addressing, immediate addressing, and indexed addressing.
6. Control Bits: Control bits provide additional information or settings related to the instruction execution. They can indicate things like conditional execution, interrupt handling, memory access modes, or other special instructions.
7. Data Size: The data size component specifies the size of the operands involved in the instruction. It determines whether the operation is performed on 8-bit, 16-bit, or other-sized data. This information is crucial for proper data handling and manipulation.
8. Conditional Execution: Conditional execution bits define whether the instruction is executed based on specific conditions or flags. These bits control whether the instruction is skipped or executed, depending on the state of the processor's condition flags, such as carry flag, zero flag, etc.

8.C. escribe ADD,SUB,MUL,DIV in 8086 microprocessor

ChatGPT

In the 8086 microprocessor, the instructions ADD, SUB, MUL, and DIV are used for arithmetic operations. Here's a description of each instruction:

1. ADD (Addition): The ADD instruction is used to perform addition between operands. It adds the source operand to the destination operand and stores the result in the destination operand. The ADD instruction supports various addressing modes, allowing you to perform addition on registers, memory locations, and immediate values.

Syntax: ADD destination, source

SUB (Subtraction): The SUB instruction is used to perform subtraction between operands. It subtracts the source operand from the destination operand and stores the result in the destination operand. Like the ADD instruction, the SUB instruction supports various addressing modes.

Syntax:

SUB destination, source

MUL (Multiplication): The MUL instruction is used to perform unsigned multiplication. It multiplies an 8-bit or 16-bit operand with the contents of the AX register and stores the result in the AX register (for 8-bit operands) or DX:AX registers (for 16-bit operands). The operand can be a register or a memory location.

Syntax:

MUL operand

DIV (Division): The DIV instruction is used to perform unsigned division. It divides the contents of the DX:AX registers (for 16-bit dividend) or AX register (for 8-bit dividend) by the specified operand and stores the quotient in the AX register and the remainder in the DX register (for 16-bit dividend) or AH register (for 8-bit dividend). The operand can be a register or a memory location.

Syntax:

DIV operand