**Name: Waleed Afzal**

**Roll No: 23P-0566**

**Section: 3D BS(CS)**

**COAL ASSIGNMENT # 01**

**Question 1:  
What are the segments that memory is divided in the 8088 architecture, write their names and names of their corresponding offset registers.   
  
Ans:**

In the 8088 architecture, memory is divided into four segments, and each segment has a corresponding segment register and an offset register used for addressing within the segment.

1. **Code Segment (CS)**:

* **Segment Register**: CS (Code Segment)
* **Offset Register**: IP (Instruction Pointer)

1. **Data Segment (DS)**:

* **Segment Register**: DS (Data Segment)
* **Offset Registers**: SI (Source Index), DI (Destination Index), BX (Base Register)

1. **Stack Segment (SS)**:

* **Segment Register**: SS (Stack Segment)
* **Offset Register**: SP (Stack Pointer) and BP (Base Pointer)

1. **Extra Segment (ES)**:

* **Segment Register**: ES (Extra Segment)
* **Offset Register**: DI (Destination Index)

**Question2:  
Code:  
org 0x0100;** *Set the origin of the program to 0x0100*

**mov ax, [num];** *Move the value at the memory location 'num' into the AX register*

**mov dx, ax;** *Copy the value from AX into the DX register*

**add dx, -1;** *Subtract 1 from the value in DX*

**mov cx, dx;** *Copy the value from DX into CX (CX will work as the loop counter)*

**l1:** ; *making a loop*

**add [num], ax;** *Add the value in AX to the value stored at memory location 'num'*

**add cx, -1;** *Decrease the loop counter (CX) by 1*

**jnz l1;** *If CX is not zero, jump back to the label 'l1' (continue the loop)*

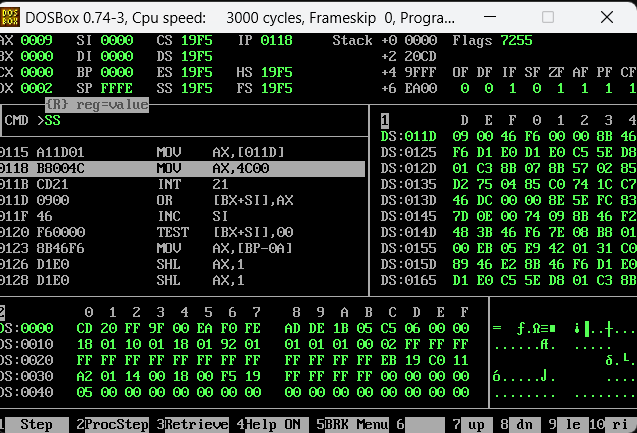
**mov ax, [num];** *After the loop, move the final value of 'num' into AX*

**mov ax, 0x4c00;** *this is the code for program termination*

**int 0x21;**

**num dw 3**; *Define the memory location 'num' and store the initial value 3*

**Output:**

****

**Question 03:  
How does Intel 8086 microprocessor convert a 16 bit logical address pair into a 20 bit physical address? Show with an example.**

**Ans:**To Calculate Physical address from logical address:

We will add a zero in the end of the our Segment Address (Core Segment) and then add it with IP Address so this is how we will find out physical address.  
  
***Example:***

CS 🡪 1234

IP 🡪 0110

🡪 Now adding zero in the end of CS Value 🡺 12340

🡺 Now Add that CS Value to IP:

**1 2 3 4 0**

**+ 0 1 1 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1 2 4 5 0**

**PHYSICAL ADDRESS IS: 12450**

**Question 4:**

**Code:**

org 0x100

mov ax, 0FFFFh

mov bx, 0FFFFh

add ax,bx ; for carry flag

mov ax, 0100

mov bx, 0100

sub ax,bx ;for zero flag

;overflow flag

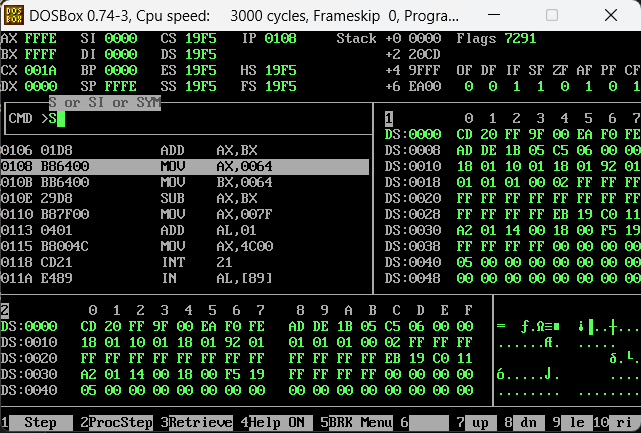
mov ax, 127

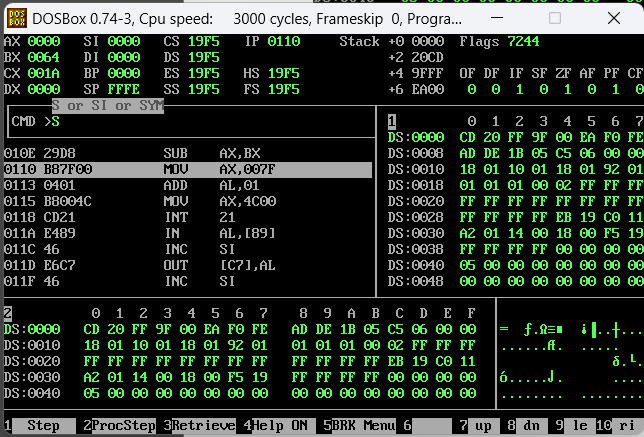
add al, 1

mov ax, 0x4c00

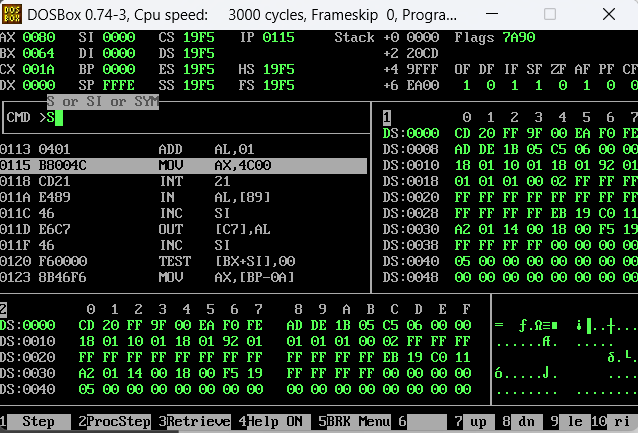
int 0x21

**Outputs:**

**🡪Carry Flag:  
**

**🡪Zero Flag:**

**🡪OverFlow Flag:**

****

**Question 05:**

**What are the contents of memory locations 100, 101, 102, and 103 if the word 0xE32A is stored at offset 100 and the word 0xB801 is stored at offset 102?**

**Ans:**

Intel 8086 Processor work on little endian format, so according to that the data will be stored as:

* **Memory location 100: 2A**
* **Memory location 101: E3**
* **Memory location 102: 01**
* **Memory location 103: B8**

**Question 06:**

**What is the difference between little endian and big endian formats? Which format is used by the Intel 8088 microprocessor?**

***Difference Between Little Endian and Big Endian:***

**1. Little Endian:**

- In little endian format, the **least significant byte (LSB)** is stored at **the lowest memory address**, and **the most significant byte (MSB)** is stored at the **highest address.**

- Example: If you store the number **0x1234** in memory, it would be stored as:

**- Lower address: 0x34 (LSB)**

**- Higher address: 0x12 (MSB)**

**2. Big Endian:**

- In big endian format, the **most significant byte (MSB**) is stored at the **lowest memory address,** and **the least significant byte (LSB)** is stored at the **highest address.**

- Example: If you store the number **0x1234** in memory, it would be stored as:

**- Lower address: 0x12 (MSB)**

**- Higher address: 0x34 (LSB)**

**🡪Which format is used by the Intel 8088 microprocessor**?

The Intel 8088 microprocessor uses little endian format.

**Question 07:**

**What are the first and the last physical memory addresses accessible using the following segment values?**

**i) 0AB0**

**ii) 0F5F**

**iii) 1002**

**iv) 0101**

**v) E010**

**Ans:**

**i) 0AB0:**

First Add 0 in the end of this Segment Value 🡺 **0AB00**

**🡺 Now Suppose First Offset Is = *0000* 🡸**

**🡺 Adding *0AB00* into *0000*:**

**0 A B 0 0**

**+ 0 0 0 0**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**0 A B 0 0**

**🡺 Now Suppose First Offset Is = *0101* 🡸**

**🡺 Adding *0AB00* into *0101*:**

**0 A B 0 0**

**+ 0 1 0 1**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**0 A C 0 1**

**Solution of Part (i) First and Last Physical Address are:  
🡪 First Address: 0AB00**

**🡪 Last Address: 0AC01**

**ii) 0F5F:**

First Add 0 in the end of this Segment Value 🡺 **0F5F0**

**🡺 Now Suppose First Offset Is = *0000* 🡸**

**🡺 Adding *0AB00* into *0000*:**

**0 A B 0 0**

**+ 0 0 0 0**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**0 F 5 F 0**

**🡺 Now Suppose First Offset Is = *0101* 🡸**

**🡺 Adding *0AB00* into *0101*:**

**0 F 5 F 0**

**+ 0 1 0 1**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**0 F 6 F 1**

**Solution of Part (ii) First and Last Physical Address are:  
🡪 First Address: 0F5F0**

**🡪 Last Address: 0F6F1**

**iii) 1002:**

First Add 0 in the end of this Segment Value 🡺 **10020**

**🡺 Now Suppose First Offset Is = *0000* 🡸**

**🡺 Adding *0AB00* into *0000*:**

**1 0 0 2 0**

**+ 0 0 0 0**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1 0 0 2 0**

**🡺 Now Suppose First Offset Is = *0101* 🡸**

**🡺 Adding *0AB00* into *0101*:**

**1 0 0 2 0**

**+ 0 1 0 1**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1 0 1 2 1**

**Solution of Part (iii) First and Last Physical Address are:  
🡪 First Address: 10020**

**🡪 Last Address: 10121**

**iv) 0101:**

First Add 0 in the end of this Segment Value 🡺 **01010**

**🡺 Now Suppose First Offset Is = *0000* 🡸**

**🡺 Adding *0AB00* into *0000*:**

**0 1 0 1 0**

**+ 0 0 0 0**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**0 1 0 1 0**

**🡺 Now Suppose First Offset Is = *0101* 🡸**

**🡺 Adding *0AB00* into *0101*:**

**0 1 0 1 0**

**+ 0 1 0 1**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**0 1 1 1 1**

**Solution of Part (iv) First and Last Physical Address are:  
🡪 First Address: 01010**

**🡪 Last Address: 01111**

**v) E010:**

First Add 0 in the end of this Segment Value 🡺 **E0100**

**🡺 Now Suppose First Offset Is = *0000* 🡸**

**🡺 Adding *0AB00* into *0000*:**

**E 0 1 0 0**

**+ 0 0 0 0**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**E 0 1 0 0**

**🡺 Now Suppose First Offset Is = *0101* 🡸**

**🡺 Adding *0AB00* into *0101*:**

**E 0 1 0 0**

**+ 0 1 0 1**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**E 0 2 0 1**

**Solution of Part (V) First and Last Physical Address are:  
🡪 First Address: E0100**

**🡪 Last Address: E0201**

**Question 8:**

**Give the value of the zero flag, the carry flag, the sign flag, and the overflow flag after each of the following instructions if AX is initialized with 0x1254 and BX is initialized with 0x0FFF.**

**Ans:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Line Number** | **Zero Flag** | **Carry Flag** | **Sign Flag** | **OverFlow Flag** |
| **1.** | **0** | **0** | **1** | **0** |
| **2.** | **0** | **1** | **0** | **0** |
| **3.** | **1** | **1** | **0** | **0** |

**QUESTION 9:**

**How much memory (RAM) can be accommodated in MBs by processors that have an address bus of length 16, 32 and 64 bit respectively?.**

🡺 **16-bit Address Bus:**

* The processor can generate **216** (2 to the power of 16) unique addresses.
* **216** =65,5362= 65,536
* Each address points to 1 byte of memory.
* Therefore, the total memory capacity is 65,536 bytes.
* To convert bytes to megabytes:

65,536 Bytes / (1024 bytes/KB x 1024 KB/MB) **= 0.0625 MB (64KB)**

**Memory Capacity: 0.0625 MB (or 64 KB)**

🡺**32-bit Address Bus:**

* The processor can generate **232** unique addresses.
* **232** =4,294,967,296 addresses.
* Each address points to 1 byte of memory.
* Therefore, the total memory capacity is 4,294,967,296 bytes.
* To convert bytes to megabytes:

4,294,967,296 bytes / (1024 bytes/KB x 1024 KB/MB) = **4096 MB or (4GB)**

**Memory Capacity: 4,096 MB (or 4 GB)**

🡺**64-bit Address Bus:**

* The processor can generate **264** unique addresses.
* **264** =18,446,744,073,709,551,616
* Each address points to 1 byte of memory.
* Therefore, the total memory capacity is 18,446,744,073,709,551,616 bytes.
* To convert bytes to megabytes:
* 18,446,744,073,709,551,616 bytes / (1024 bytes/KB x 1024 KB/MB) = **17,592,186,044 MB (or about 16 Exabytes)**
* **Memory Capacity: 17,592,186,044 MB (or about 16 Exabytes)**

**Question 10:**

**Parity bit checks for single-bit errors. It’s an extra bit added to a string of binary code to ensure data integrity. Electronic voting has been introduced in Pakistan and Ali is a strong candidate for being elected as Mayor. His current vote count is 1023(1111111111). Unfortunately, a high energy proton from space hits the processor of one of the voting machines causing two of the bits to flip in ax register. Thus, changing Ali’s vote count to 983 (1111010111) and giving lead to his opponent. Will parity bit detect error here? Justify your answer.**

**Answer:**

**1. Initial Vote Count (Before Error):**

* Ali’s original vote count is 1023, which is 1111111111 in binary.
* This binary number has 10 ones (1s). Since 10 is even, the parity bit for even parity would be 0.

**2. Vote Count After Error:**

* After the error, Ali’s vote count changes to 983, which is 1111010111 in binary.
* This binary number has 8 ones (1s). Since 8 is even, the parity bit for even parity would still be 0.

**3. Parity Bit Check for Errors:**

* Single-bit Error: Parity bits are great for detecting single-bit errors. If only one bit changes, the total number of 1s will change from even to odd or from odd to even. The parity bit will detect this change.
* Two-bit Error: Parity bits are not effective for detecting errors where two bits flip. When two bits flip, the total number of 1s can still be even (or odd), so the parity bit won’t be able to detect that an error occurred.

**4. Applying to Ali’s Case:**

* The error in Ali’s vote count involved flipping two bits: 1111111111 changed to 1111010111.
* Both numbers have an even number of 1s (10 ones and 8 ones), so if we use even parity, the parity bit will be the same in both cases (0 for even parity).

🡺 **Conclusion**: In this case, the parity bit will not detect the error because the parity (number of 1s) remains even despite the two bits flipping. Parity bits are not designed to catch errors where two bits change simultaneously.

**Question 11:**

**What were the reasons behind the introduction of segmentation in the 8088-microprocessor architecture? How did segmentation address the limitations of earlier memory management approaches?**

**Ans:**

***Reasons for Segmentation in the 8088 Microprocessor***

1. **Address Space Expansion:**
   * The 8088 could directly address only 64 KB with its 16-bit address bus. Segmentation allowed access to up to 1 MB of memory.
2. **Efficient Memory Management:**
   * Segmentation divides memory into segments (code, data, stack), making it easier to manage and organize large programs.
3. **Improved Organization:**
   * Segments help separate and organize different types of data and instructions, facilitating better memory usage.
4. **Protection and Isolation:**
   * Segmentation provides a way to protect and isolate different segments, like having separate segments for code and data.

**Question 12:**

**What is the effective address generated by the following combinations if they are valid. If not give reason. Initially BX=0x0100, SI=0x0010, DI=0x0001, BP=0x0200, and SP=0xFFFF**

**a. bx-si**

**b. bx-bp**

**c. bx+10**

**e. bx+sp**

**f. bx+di**

**Ans:**

* **BX = 0x0100**
* **SI = 0x0010**
* **DI = 0x0001**
* **BP = 0x0200**
* **SP = 0xFFFF**

**Calculations:**

1. **bx - si:**
   * BX = 0x0100
   * SI = 0x0010
   * Effective Address = 0x0100 - 0x0010 = 0x00F0

***Result: 0x00F0***

1. **bx - bp:**
   * BX = 0x0100
   * BP = 0x0200
   * Effective Address = 0x0100 - 0x0200 = -0x0100 (in decimal: -256)

***Result: -0x0100 (or, if considering unsigned, it could be seen as 0xFFFF in a 16-bit register)***

1. **bx + 10:**
   * BX = 0x0100
   * Adding 10 (0x0A in hexadecimal)
   * Effective Address = 0x0100 + 0x0A = 0x010A

***Result: 0x010A***

1. **bx + sp:**
   * BX = 0x0100
   * SP = 0xFFFF
   * Effective Address = 0x0100 + 0xFFFF = 0x0100 - 0x0001 (in two's complement arithmetic) = 0x00FF

***Result: 0x00FF***

1. **bx + di:**
   * BX = 0x0100
   * DI = 0x0001
   * Effective Address = 0x0100 + 0x0001 = 0x0101

***Result: 0x0101***

**Question 13: Fibonacci Series for 5 terms**

**org 0x100**

**;Initialize the first two Fibonacci numbers**

**mov ax, 0 ; F0 = 0**

**mov bx, 1 ; F1 = 1**

**;Store F0 and F1 in memory locations**

**mov [fib0], ax**

**mov [fib1], bx**

**;Calculate the third Fibonacci number (F2 = F0 + F1)**

**add ax, bx ; AX = F0 + F1**

**mov [fib2], ax ; Store F2**

**;Calculate the fourth Fibonacci number (F3 = F1 + F2)**

**mov ax, bx ; AX = F1**

**add ax, [fib2] ; AX = F1 + F2**

**mov [fib3], ax ; Store F3**

**;Calculate the fifth Fibonacci number (F4 = F2 + F3)**

**mov bx, [fib2] ; BX = F2**

**add bx, [fib3] ; BX = F2 + F3**

**mov [fib4], bx ; Store F4**

**mov ax, 0x4c00 ; Exit program**

**int 0x21**

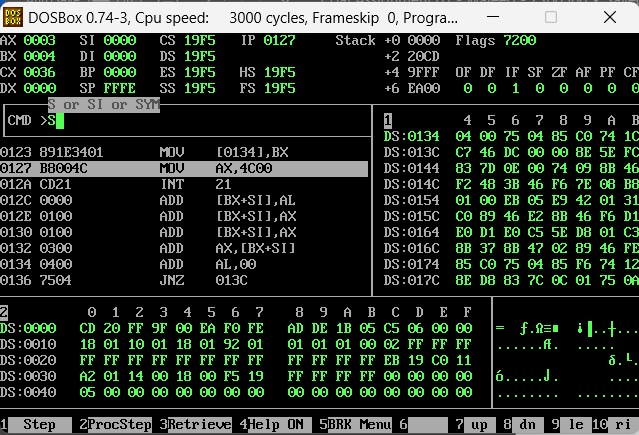
**fib0 dw 0 ; Memory location to store F0**

**fib1 dw 0 ; Memory location to store F1**

**fib2 dw 0 ; Memory location to store F2**

**fib3 dw 0 ; Memory location to store F3**

**fib4 dw 0 ; Memory location to store F4**

**OUTPUT:  
**

**Question 14: Sum of 5 Numbers in Array:**

**Code:  
org 0x100**

**mov ax, [arr]**

**mov bx,2**

**mov cx,5**

**l1:**

**add ax,[arr+bx]**

**add bx,2**

**sub cx,1**

**jnz l1**

**mov [sum], ax**

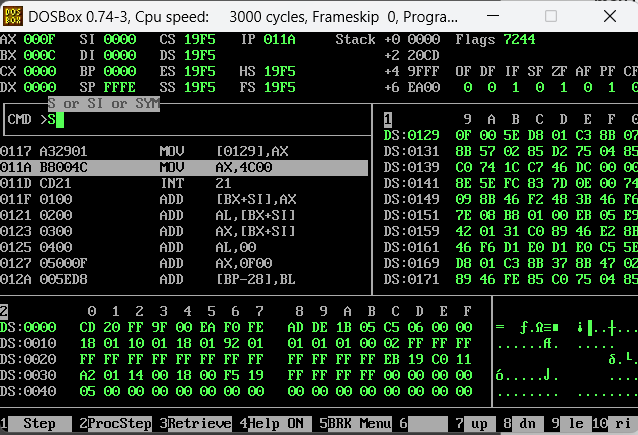
**mov ax, 0x4c00**

**int 0x21**

**arr: dw 1,2,3,4,5**

**sum: dw 0**

**Output:**

****

**Question 15: Multiplying a Num by 5 through Repeated Addition:**

**Code:  
org 0x100**

**mov ax, [num]**

**mov cx,4**

**l1:**

**add ax,[num]**

**sub cx,1**

**jnz l1**

**mov [sumfive], ax**

**mov ax, 0x4c00**

**int 0x21**

**num: dw 2**

**sumfive: dw 0  
  
Output:**