**MICROPROCESSORS**

**AND**

**MICROCONTROLLERS LAB**

**ETEE-358**

Mr. Rohit Uday Lakhane

(Faculty name)

Chander Jindal

06514802719

6th Semester

****

Maharaja Agrasen Institute of Technology, PSP area,

Sector – 22, Rohini, New Delhi – 110085

( Affiliated to Guru Gobind Singh Indraprastha University, New Delhi )

### 

### MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

### VISION

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

#### **MISSION**

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

**Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

**Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

**Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

**Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

**Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi-disciplinary approach.

**Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities



### MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

**VISION**

“To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

### MISSION

“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing innovative solutions to real life problems and fulfill societal needs.”

**MICROPROCESSORS AND MICROCONTROLLERS LAB**

**PRACTICAL RECORD**

**PAPER CODE** : ETEE-358

**NAME OF THE STUDENT** : Chander Jindal

**UNIVERSITY ROLL NO.** : 06514802719

**BRANCH** : CSE

**SECTION/ GROUP** : 6C4

**PRACTICAL DETAILS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **EXPERIMENT** | **DATE** | **SIGNATURE** |
| 1. | Addition of two 8-bit numbers in 8085 using   1. Immediate addressing mode 2. Direct addressing mode 3. Indirect addressing mode | 2/3/2022 |  |
| 2. | 1. Multiply two 8-bit numbers using 8085 2. Divide two 8-bit numbers using 8085 | 9/3/2022 |  |
| 3. | Sort a series of numbers saved at consecutive memory locations in:   1. Ascending Order 2. Descending Order | 16/3/2022 |  |
| 4. | Addition of two binary numbers of 8-byte length in 8086 | 23/3/2022 |  |
| 5. | To find the largest number in a string of data. | 30/3/2022 |  |
| 6. | To find the length of the string which is terminated by an identifier given in AL Register. | 13/4/2022 |  |
| 7. | To convert a given string of data into its equivalent 2’s complement form. | 5/5/2022 |  |
| 8. | To perform factorial of a given number. | 19/5/2022 |  |

**Experiment 1**

**Aim:** Addition of two 8-bit numbers in 8085 using

1. Immediate addressing mode
2. Direct addressing mode
3. Indirect addressing mode

**Apparatus:** Excel Technologies Kit 8085

**Theory:**

1. **Immediate Addressing Mode:** In immediate addressing mode there exists all the instruction that has an I at the end such as MVI, ADI, etc and all these commands acts on the immediate next value written. (Also the result of the addition is stored in the Register A itself unlike in Direct Addressing Mode and Indirect Addressing Mode). Here in this program first we copy the value of 57H in the register A using the MVI command (Move immediate). Then we using the ADI command (Add immediate) we will add 92H to 57H (ADI command by default adds the immediate next number with the register A) and the answer(E9H) is stored in the register A itself. Thus, the addition of two 8-bit numbers using Immediate addressing mode is done successfully.
2. **Direct Addressing Mode:** In direct addressing mode we assume that there are some values stored at particular memory locations and we load the value from a particular memory location into the accumulator. Here in this program first we load the value stored in 2050 location i.e. 57H into the accumulator using LDA (Load Accumulator with the contents from the memory location written along with it). Then this value is moved from register A to register B to vacate the space in register A to store the second number using LDA. After that ADD B command then adds the number stored in register B with the number stored in Register A (by default), thus resulting in our required sum (57H +92H). This result(E9H) is then stored in the memory location 2052 using the STA command (Store accumulator contents in the memory location written along with it). Thus, the addition of two 8-bit numbers using the Direct addressing mode is done successfully.
3. **Direct Addressing Mode:** In indirect addressing mode we assume that there are some values stored at particular memory locations. We first use the command LXI to load the 16-bit address 2050H into the register pair HL (using LXI H,2050) where H=20(Higher 8 bits) and L=50(Lower 8 bits). Now the command MOV A,M works only with the HL register pair by default and not with BC or DE register pair. So, by using this command the value stored at memory location 2050H represented by HL register pair i.e. 57H is moved to register A. Now to get the second number we first use the command INX H which increases the value of HL pair by 1(i.e. INX H=> HL=HL+1). Now the HL pair is pointing towards the 2051H memory location. After this the number stored in 2051H memory location is added to the number stored in register A using the ADD M command (that adds the number stored at a particular memory location with register A by default). This result of our addition (57H + 92H) is stored in the register A for now. Then we again use the INX H command so that HL register pair values becomes 2052H and then we use the MOV command to move the result(E9H) of the addition from register A to the memory location 2052H. Thus, the addition of two 8-bit numbers using the Indirect addressing mode is done successfully.

**Code:**

The two numbers to be added are 57H and 92H

57H +92H=E9H

1. **Using Immediate Addressing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Memory Address (MA)** | **Instruction** | **Opcode** | **Comments** |
| 2000H | MVI A,57H | 3E | Copy 57 in A |
| 2001H |  | 57 | A=57 |
| 2002H | ADI 92H | C6 | Add Immediate i.e. 92 to 57 |
| 2003H |  | 92 | A=E9 |
| 2004H | RST 5 | EF | Break Point Interrupt |

1. **Using Direct Addressing**

**Assuming:** 2050H=57H

2051H=92H

|  |  |  |  |
| --- | --- | --- | --- |
| **Memory Address (MA)** | **Instruction** | **Opcode** | **Comments** |
| 2000H | LDA 2050 | 3A |  |
| 2001H |  | 50 |  |
| 2002H |  | 20 | A = 57 |
| 2003H | MOV B,A | 47 | B = 57 |
| 2004H | LDA 2051 | 3A |  |
| 2005H |  | 51 |  |
| 2006H |  | 20 | A = 92 |
| 2007H | ADD B | 80 | A = E9 |
| 2008H | STA 2052 | 32 | Store result at 2052 |
| 2009H |  | 52 |  |
| 200AH |  | 20 | [2052]=E9 |
| 200BH | RST 5 | EF | Break Point Interrupt |

1. **Using Indirect Addressing**

**Assuming:** 2050H=57H

2051H=92H

|  |  |  |  |
| --- | --- | --- | --- |
| **Memory Address (MA)** | **Instruction** | **Opcode** | **Comments** |
| 2000 H | LXI H,2050H | 21 | HL=2050 |
| 2001 H |  | 50 | L=50 |
| 2002 H |  | 20 | H=20 |
| 2003 H | MOV A,M | 7E | A=57 |
| 2004 H | INX H | 23 | HL=2051 |
| 2005 H | ADD M | 86 | A=E9 |
| 2006 H | INX H | 23 | Hl=2052 |
| 2007 H | MOV M,A | 77 | [2052H]=E9 |
| 2008 H | RST 5 | EF | Break Point Interrupt |

**Result:**

1. **Immediate Addressing:**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| A=57H | A=E9H |

1. **Direct Addressing**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| [2050H]=57H | [2050H]=57H |
| [2051H]=92H | [2051H]=92H |
| A=57H | A=E9H |
| 2052H=? | 2052H=E9H |

1. **Indirect Addressing**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| [2050H]=57H | [2050H]=57H |
| [2051H]=57H | [2051H]=92H |
| A=57H | A=E9H |
| 2052H=? | [2052H]=E9H |

**Experiment 2**

**Aim:** i. Multiply two 8-bit numbers using 8085

ii. Divide two 8-bit numbers using 8085

**Apparatus:** Excel Technologies Kit 8085

**Theory:**

1. **Multiplication of two 8-bit numbers:** To multiply two numbers in 8085 say K\*L we use the fact that it is like adding the number K L times ( i.e. K+K+K+K+……L times). To do this first we store the first number i.e. 75H into register B and the second number i.e.93H in register C using the MVI command. We also initialise the registers D and A as 00 which will at the end of the program will store the higher and lower bits of our result respectively. Then we define a label AGAIN for multiple (i.e. 93 times) addition of number 75. First, we will add the number K stored in register B with the number stored in Accumulator using ADD command (i.e. 0+75) and the result is stored in A itself. Then we have the JNC NEXT (Jump if No Carry) command that will help to jump to a particular label written adjacent to it based on the fact that if there is no carry (i.e. Carry Flag=0) obtained in the arithmetic operation above JNC command then it will jump to the desired label otherwise the JNC command will be skipped and we move to the next command. There is another label NEXT in which first decrement the value stored in Register C by 1 using DCR C command i.e. the value of number 93H each time the addition of number 75H takes place. Then we have the JNZ AGAIN (Jump if not Zero) command for going back to the AGAIN label for addition of number 75H with the previously stored value in the Accumulator. Going back to the AGAIN label is based on the fact whether or not the Zero flag is 0 or not. If all the arithmetic operation above the JNZ command are not 0 then the Zero flag=0 and we jump to the label written adjacent to the JNZ command otherwise we skip the JNZ command and move to next command. There is also a command called INR D which will increases the value of Register D by 1(i.e. D=D+1). It is will be executed if there is a carry obtained above the JNC NEXT command and will add all the carry values. This all will happed till the time values in Register C is 01 before it reaches the NEXT label. There the value in register C becomes zero and the Zero flag=1 and then the instruction JNZ AGAIN will be skipped and we will reach the end of our program at RST 5. Thus, the multiplication of two 8-bit numbers using the Indirect addressing mode is done successfully.
2. **Division of two 8-bit numbers:** To divide two numbers in 8085 say K/L we use the fact that it is like subtracting the number K by number L until it is greater than 0 i.e. K-L-L-L-L-…….. >0. The number of times we subtract the number L will gives us the Quotient and the remaining part of number K will be the Remainder. To do this first we store the first number i.e. 0DH into register B and the second number i.e.03H in register C using the MVI command. We also initialise the registers D as FFH which will at the end of the program will store the Quotient of the division being done. The arithmetic calculations will be done in Accumulator A which in the end will have the Remainder of the division. Then we move the value(0DH) stored in register B to the Accumulator to carry out the subtraction of number 03H. Then we define the BACK label in which the first statement is INR D which will increase the value of Register D by 1(i.e. D=D+1) and then there is the SUB C statement after which we have JNC BACK instruction that checks the Carry flag before doing INR D. If there is no carry (i.e. Carry Flag=0) obtained in the arithmetic operation above JNC command then it will jump to the desired label otherwise the JNC command will be skipped and we move to the next command. So, this will happen until K-L-L-L>0 i.e. 0DH-03H-03H >0. When this expression becomes < 0 the Carry flag (CY)=1 so we will not go back to the BACK label, but the value in A had become negative and to restore it to get the actual remainder we will need to ADD C after the JNC BACK command. Now initially D=FFH because even though what we have done before we will get D=5, so we initialize D=FFH, thus getting the Quotient(D=4) as 4 at the end(as FFH+01H=00H). Thus, the multiplication of two 8-bit numbers using the Indirect addressing mode is done successfully.

**Code:**

1. **Multiplication of two 8 bit numbers**

The two numbers to be multiplied are 75H and 93H

75H\*93H=432FH

(**In Decimal:** 117\*147=17199)

|  |  |  |  |
| --- | --- | --- | --- |
| **Memory Address (MA)** | **Instruction** | **Opcode** | **Comments** |
| 2000 H | MVI B,75H | 06 |  |
| 2001 H |  | 75 | B=75 |
| 2002 H | MVI C,93H | 0E |  |
| 2003 H |  | 93 | C=93 |
| 2004 H | MVI D,00H | 16 |  |
| 2005 H |  | 00 | D=0 |
| 2006 H | MVI A,00H | 3E |  |
| 2007 H |  | 00 | A=0 |
| 2008 H | **AGAIN:** ADD B | 80 |  |
| 2009 H | JNC NEXT | D2 |  |
| 200A H |  | 0D | [Address of |
| 200B H |  | 20 | NEXT Label] |
| 200C H | INR D | 14 | To Add the Carry Values |
| 200D H | **NEXT:** DCR C | 0D |  |
| 200E H | JNZ AGAIN | C2 |  |
| 200F H |  | 08 | [Address of |
| 2010 H |  | 20 | AGAIN Label] |
| 2011 H | RST 5 | EF | Break Point Interrupt |

1. **Division of two 8 bit numbers**

The two numbers to be divided are 0DH and 03H

0DH/03H = Remainder:4 & Quotient:1

**(In Decimal:** 13/3 = Remainder:4 & Quotient:1)

|  |  |  |  |
| --- | --- | --- | --- |
| **Memory Address (MA)** | **Instruction** | **Opcode** | **Comments** |
| 2000 H | MVI B,0DH | 06 |  |
| 2001 H |  | 0D | B=0D |
| 2002 H | MVI C,03H | 0E |  |
| 2003 H |  | 03 | C=03 |
| 2004 H | MVI D,FFH | 16 |  |
| 2005 H |  | FF | D=FF |
| 2006 H | MOV A,B | 78 | A=0D |
| 2007 H | **BACK:** INR D | 14 | To get the Quotient |
| 2008 H | SUB C | 91 |  |
| 2009 H | JNC BACK | D2 |  |
| 200A H |  | 07 | [Address of |
| 200B H |  | 20 | BACK Label] |
| 200C H | ADD C | 81 |  |
| 200D H | RST 5 | EF | Break Point Interrupt |
|  |  |  |  |

**Result:**

1. **Multiplication of two 8-bit numbers**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| A=00H | A=2FH |
| B=75H | B=75H |
| C=93H | C=00H |
| D=00H | D=43H |

1. **Division of two 8-bit numbers**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| A=0DH(Remainder) | A=01H or 1(In Decimal) |
| B=0DH | B=0DH |
| C=03H | C=03H |
| D=FFH(Quotient) | D=04H or 4(In Decimal) |

Experiment 3

**Aim:** Sort a series of numbers saved at consecutive memory locations in:

1. Ascending Order
2. Descending Order

**Apparatus:** Excel Technologies Kit 8085

**Theory:** To sort a series of number saved at consecutive memory locations in ascending or descending order we need to know how to swap two numbers and find the largest number in a series of numbers. We start by assuming by that the numbers to be sorted are already stored in consecutive memory locations. Then we initialize a Register D with value 05 that specifies the number of iterations we need to perform to sort the code in either ascending or descending order. Then we define an AGAIN label in which we initialize the Register E with value 05 that specifies the number of comparisons to be done for each iteration. Then the memory location 2050H which holds the first number is loaded into the HL Register pair. Then we define a back Label for comparing the numbers stored in first two memory locations i.e. 2050H and 2051H for the first time. It will include moving the number from the memory location 2050H into A, then increasing the value of HL Register pair by 1 and then comparing the number stored in 2051H location with the number stored in Accumulator. This is done with the help of CMP M command which by defaults compare the number with the number stored in Accumulator. The CMP command has a following table:

|  |  |  |
| --- | --- | --- |
|  | CY (Carry Flag) | Z |
| A<M | 1 | 0 |
| A=M | 0 | 1 |
| A>M | 0 | 0 |

Now depending upon the following condition we will have the value of CY and Z for each comparison. If we get CY=1 we will jump to the SKIP label using the JC SKIP statement. If CY=0 we skip the JC SKIP statement and move to the next executable statement.

1. If we move to the SKIP label it means that 1 comparison is done and we should do the next comparison for that particular iteration. Thus we decrease the value of E by 1 using DCR E and then if we have value of Z=0 we go back to the BACK label otherwise we skip this statement which will happen only when E=00H which will mean Z=1.
2. If we go back to the BACK label we will again perform the second comparison between two numbers for a particular iteration.
3. If we skip the JNZ BACK statement it will mean that 1 iteration is over and we will decrease the number of iterations by 1 using DCR D command. After this we will check if D is 00 or not.
4. If D=00 we will skip the JNZ AGAIN statement because then the Z=1 and we will reach the end of our program.
5. If D !=00, Z=0 and we will jump to the AGAIN label for our next iteration as we need to start again from the 1st comparison, so we need to restart the condition again.
6. If we skip the JC SKIP statement it means that either the two numbers compared are equal to each other or the first number is greater than the second i.e. A>M. In either case we need to swap the two numbers. For this we first move the number represented by M to Register B and then move the number stored in Accumulator to M(a particular memory location). Then we decrease the value of register pair HL by that store the number represented by M using DCX H (i.e. DCX H =>HL=HL-1) and move the smaller number stored in Register B to the memory location HL-1. Then again we increase the value of HL Register pair by 1 to carry out the next comparison for a particular iteration.

Thus, a series of numbers saved at consecutive memory locations is sorted in **Ascending order.**

**To sort the series in Descending order we just need to change the Statement JC SKIP => JNC SKIP(**i.e. Jump if CY=0 that is the number in Accumulator is greater than the number represented by M; If CY=1, it means that i the number in Accumulator is less than the number represented by M and we need to swap the two number which would be done after skipping the JNC SKIP statement)**.** Rest of the code remains the same

**Code:**

We assume that the numbers to be sorted are saved from memory location 2050H to 2055H.

1. **Ascending Order**

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Address (MA) | Instruction | Opcode | Comments |
| 2000 H | MVI D,05H | 16 | D=05 |
| 2001 H |  | 05 |  |
| 2002 H | **AGAIN:** MVI E,05H | 1E |  |
| 2003 H |  | 05 | E=05 |
| 2004 H | LXI H,2050H | 21 | HL=2050 |
| 2005 H |  | 50 | L=50 |
| 2006 H |  | 20 | H=20 |
| 2007 H | **BACK:** MOV A,M | 7E |  |
| 2008 H | INX H | 23 |  |
| 2009 H | CMP M | BD |  |
| 200A H | JC SKIP | DA |  |
| 200B H |  | 12 | [Address of |
| 200C H |  | 20 | SKIP Label] |
| 200D H | MOV B,M | 46 |  |
| 200E H | MOV M,A | 77 |  |
| 200F H | DCX H | 2B |  |
| 2010 H | MOV M,B | 70 |  |
| 2011 H | INX H | 23 |  |
| 2012 H | **SKIP:** DCR E | 1D |  |
| 2013 H | JNZ BACK | C2 |  |
| 2014 H |  | 07 | [Address of |
| 2015 H |  | 20 | BACK Label] |
| 2016 H | DCR D | 15 |  |
| 2017 H | JNZ AGAIN | C2 |  |
| 2018 H |  | 02 | [Address of |
| 2019 H |  | 20 | AGAIN Label] |
| 201A H | RST 5 | EF | Break Point Interrupt |

1. **Descending Order**

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Address (MA) | Instruction | Opcode | Comments |
| 2000 H | MVI D,05H | 16 | D=05 |
| 2001 H |  | 05 |  |
| 2002 H | **AGAIN:** MVI E,05H | 1E |  |
| 2003 H |  | 05 | E=05 |
| 2004 H | LXI H,2050H | 21 | HL=2050 |
| 2005 H |  | 50 | L=50 |
| 2006 H |  | 20 | H=20 |
| 2007 H | **BACK:** MOV A,M | 7E |  |
| 2008 H | INX H | 23 |  |
| 2009 H | CMP M | BD |  |
| 200A H | JNC SKIP | DA |  |
| 200B H |  | 12 | [Address of |
| 200C H |  | 20 | SKIP Label] |
| 200D H | MOV B,M | 46 |  |
| 200E H | MOV M,A | 77 |  |
| 200F H | DCX H | 2B |  |
| 2010 H | MOV M,B | 70 |  |
| 2011 H | INX H | 23 |  |
| 2012 H | **SKIP:** DCR E | 1D |  |
| 2013 H | JNZ BACK | C2 |  |
| 2014 H |  | 07 | [Address of |
| 2015 H |  | 20 | BACK Label] |
| 2016 H | DCR D | 15 |  |
| 2017 H | JNZ AGAIN | C2 |  |
| 2018 H |  | 02 | [Address of |
| 2019 H |  | 20 | AGAIN Label] |
| 201A H | RST 5 | EF | Break Point Interrupt |

**Result:**

1. **Ascending Order**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| [2050H] =78H | [2050H] =12H |
| [2051H] =72H | [2051H] =30H |
| [2052H] =56H | [2052H] =44H |
| [2053H ] =44H | [2053H] =56H |
| [2054H ] =12H | [2054H ] =72H |
| [2055H] =30H | [2055H] =78H |

1. **Descending Order**

|  |  |
| --- | --- |
| **Before Execution** | **After Execution** |
| [2050H] =78H | [2050H] =78H |
| [2051H] =72H | [2051H] =72H |
| [2052H] =56H | [2052H] =56H |
| [2053H ] =44H | [2053H] =44H |
| [2054H ] =12H | [2054H ] =30H |
| [2055H] =30H | [2055H] =12H |

**Experiment 4**

**Aim:** Addition of two binary numbers of 8-byte length in 8086

**Apparatus:** MASM 8086

**Theory:**

* First the Data Segment is defined by segment defining statement. In the Data Segment two 8-byte numbers are created that is assigned to D1 and D2 respectively using DB data type (directive) which is used to declare a byte type variable or we can say we created two array naming D1 and D2 using DB which 8 byte or 64 bits long. They we create an array RESULT of length 08H and having values as 0,0,0,0,0,0,0,0 due to the DUP(0) operator. We also created a variable CARRY which is 1 byte long using DB which is initially assigned value as 00H. After this the Data Segment ends.
* Then the Code Segment is defined and the labels of the Code and the Data Segments are assigned to the CS and DS segments by ASSUME directive. Then we define a START label. Earlier we have assumed that Chander is the Data Segment but now we are initializing Chander SEGMENT actually as DS. This is done using the MOV command where Chander is loaded into the DS segment Register through AX Register (Accumulator: 16-bit Register). The AX Register here is used because segment Registers cannot be loaded with the immediate data. We also initialize a 16-bit counter Register CX with value 0008H because we need to add two 8-byte numbers.
* Firstly, effective address of the first element of D1 array is loaded into SI (a 16-bit source index Register) using LEA (Load Effective Address) command and effective address of the first element D2 array is loaded into DI (a 16-bit destination index Register) and effective address of the RESULT is loaded into BX( a 16-bit base Register).Then we clear the carry flag(sets it to 0) using the CLC instruction(and not affecting any flags).Then in the NEXT label we move the content of SI to AL(a 8-bit Register) Register and add the content of DI with AL Register along with the carry using the ADC(Add with Carry) command. The result is stored in the AL Register itself. After this we move the content of AL Register to BX Register. Then increment the content of SI Register, DI Register and BX Register by one and then we LOOP back to the NEXT label again for the next iteration(the label to loop back to is written adjacent to the LOOP command in the LOOP statement). The LOOP statement does two things:

1. It decreases the value of the CX Register (CX by default) by 1 i.e. CX=CX-1
2. It checks whether or not the value of CX Register is 0
3. If CX==0 then we move to the instruction after the LOOP statement
4. If CX!=0 then we jump to the label specified in the LOOP command.

Here the LOOP NEXT statement will be executed 8 times as the value of CX=0008H(initially).

After this we add CARRY variable value with the Carry Flag value and 00H whose result will be stored in the CARRY variable itself, thus representing the Carry obtained in the addition of two 8-byte numbers. After this the CODE SEGMENT ends and hence our code.

**Code:**

Chander SEGMENT

D1 DB 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H

D2 DB 01H, 01H, 01H, 01H, 01H, 01H, 01H, 01H

RESULT DB 08H DUP(0)

CARRY DB 00H

Chander ENDS

CODE SEGMENT

ASSUME CS:CODE , DS:Chander

START: MOV AX, Chander

MOV DS, AX

MOV CX, 0008H

LEA SI, D1

LEA DI, D2

LEA BX, RESULT

CLC

NEXT: MOV AL, [SI]

ADC AL, [DI]

MOV [BX], AL

INC SI

INC DI

INC BX

LOOP NEXT

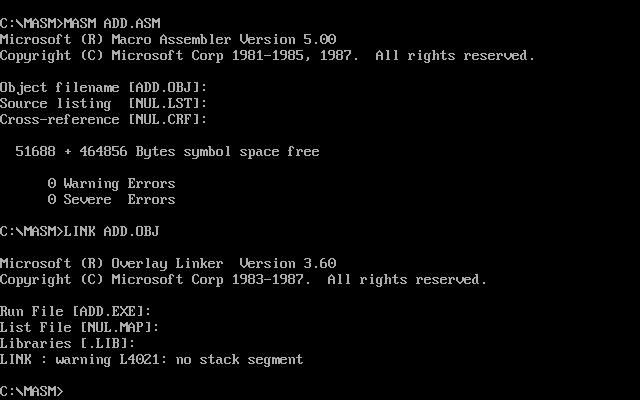
ADC CARRY, 00H

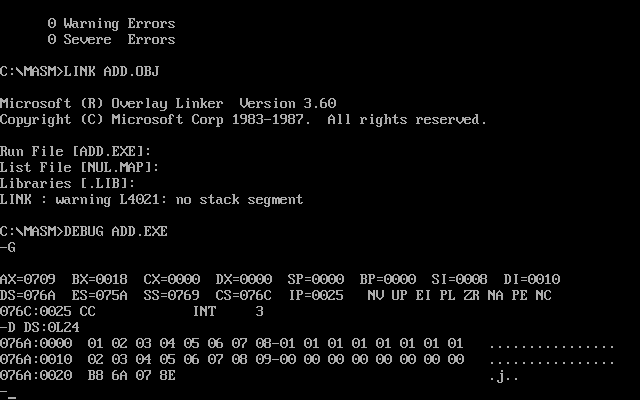
INT 3

CODE ENDS

END START

**Result:**





**Experiment 5**

**Aim:** To find the largest number in a string of data.

**Apparatus:** MASM 8086

**Theory:**

* First the Data Segment is defined by segment defining statement. In the Data Segment a string of data is entered that is assigned to ARRAY using DB data type (directive) which is used to declare a byte type variable or we can say we created an array or string of data naming ARRAY using DB. We also created a variable MAX which 1 byte long using DB which initially assigned value as 01H. After this the Data Segment ends.
* Then the Code Segment is defined and the labels of the Code and the Data Segments are assigned to the CS and DS segments by ASSUME directive. Then we define a START label. Earlier we have assumed that Chander is the Data Segment but now we are initializing Chander SEGMENT actually as DS. This is done using the MOV command where Chander is loaded into the DS segment Register through AX Register (Accumulator: 16-bit Register). The AX Register here is used because segment Registers cannot be loaded with the immediate data. Then we initialize the Registers DI and CL with 00H and 10d respectively (done so to keep a check on the string of numbers to go through to find the largest number). Now effective address of the first element of ARRAY is loaded into BX (a 16-bit base Register) using LEA (Load Effective Address) command. To find the largest number in any given array, the contents of the array must be compared with an arbitrary biggest number (MAX) which is taken to be 01H and is moved to the AL Register.
* In the BACK label we compare the first number of the array([BX+DI]=[Address of first element of ARRAY+00H]=06H) with the number in the Register AL using the CMP command.

The CMP command has a following table(CMP A,M):

|  |  |  |
| --- | --- | --- |
|  | CY (Carry Flag) | Z |
| A<M | 1 | 0 |
| A=M | 0 | 1 |
| A>M | 0 | 0 |

Now depending upon the following condition we will have the value of CY and Z for each comparison. If we get CY=0 we will jump to the SKIP label using the JNC SKIP statement. If CY=1 we skip the JNC SKIP statement and move to the next executable statement.

1. If we jump to the SKIP label we will increment DI by 1 using INC DI and decrease CL by 1 using DEC CL which is done because INC DI will help to point to the next number in the ARRAY and DEC CL will conclude that the current number before decrementing the value of CL is not the largest number. After decreasing the value of CL by 1 and then if we have value of Z=0 we go back to the BACK label otherwise we skip this statement which will happen only when CL=00H which will mean Z=1.
2. If we go back to the BACK label we will again perform the second comparison between two numbers i.e. between the previous value of [BX+DI] which is in Register AL and the next number which the current [BX+DI] points too.
3. If we skip the JNZ BACK statement it will mean that we have found the largest number in the given array on numbers and it is moved(copied) from AL Register to MAX variable. After this the CODE SEGMENT ends and hence our code.
4. If we skip the JNC SKIP statement it means that the first number is less than the second i.e. A<M. In this case we need to move the second number [BX+DI] to the Register AL as it is the bigger number and then we reach the SKIP label and continue further as stated above.

**Code:**

Chander SEGMENT

ARRAY DB 06H, 09H, 45H, 75H, 12H, 42H, 09H, 14H, 56H, 38H

MAX DB 01H

Chander ENDS

CODE SEGMENT

ASSUME CS:CODE , DS:Chander

START: MOV AX, Chander

MOV DS, AX

MOV DI, 00H

MOV CL, 10d

LEA BX, ARRAY

MOV AL, MAX

BACK: CMP AL, [BX+DI]

JNC SKIP

MOV DL, [BX+DI]

MOV AL, DL

SKIP: INC DI

DEC CL

JNZ BACK

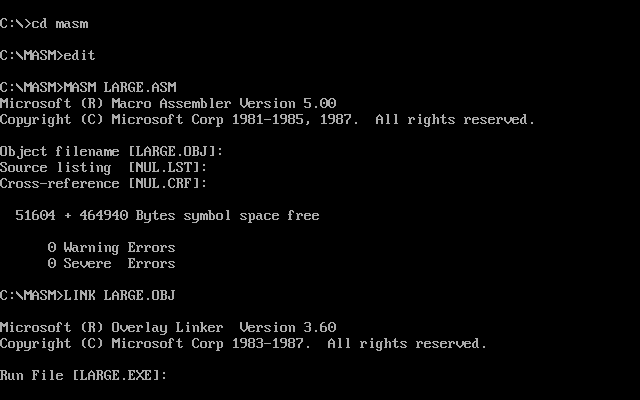
MOV MAX, AL

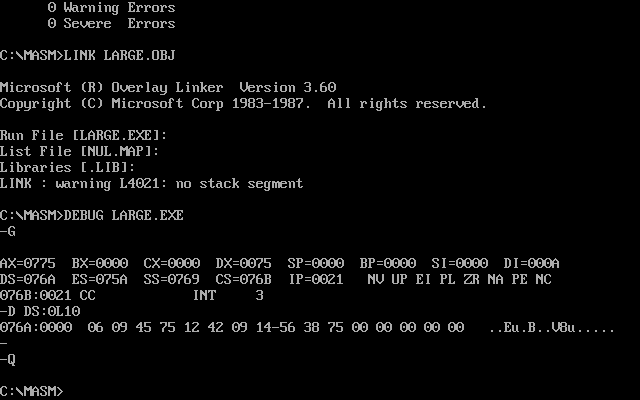
INT 3

CODE ENDS

END START

**Result:**





**Experiment 6**

**Aim:** To find the length of the string which is terminated by an identifier given in AL Register.

**Apparatus:** MASM 8086

**Theory:**

* First the Data Segment is defined by segment defining statement. In the Data Segment a string of data is entered that is assigned to NUM using DB data type (directive) which is used to declare a byte type variable or we can say we created an array or string of data naming NUM using DB. The last number 97H is actually is an identifier which upon reaching marks the end of the array or string of data. After this the Data Segment ends.
* Then the Code Segment is defined and the labels of the Code and the Data Segments are assigned to the CS and DS segments by ASSUME directive. Then we define a START label. Earlier we have assumed that Chander is the Data Segment but now we are initializing Chander SEGMENT actually as DS. This is done using the MOV command where Chander is loaded into the DS segment Register through AX Register. The AX Register here is used because segment Registers cannot be loaded with the immediate data.
* Firstly, effective address of the first element of NUM array is loaded into SI (a 16-bit source index Register) using LEA (Load Effective Address) command. Then we initialize the Register AL (a 8-bit Register) with the identifier i.e. 97H. This is done because 97H marks the end of the array and to find the length of the array we can compare each element of the array with 97H till it becomes equal to 97H thus giving us the length of the array. We also initialize an 8-bit Register DL with value 00H to count the number of elements in the array (i.e. the length of the array).
* In the BACK label, we compare the element represented by the [SI] with the value of the Register AL i.e. our identifier using the CMP command.

The CMP command has a following table(CMP A,M):

|  |  |  |
| --- | --- | --- |
|  | CY (Carry Flag) | Z |
| A<M | 1 | 0 |
| A=M | 0 | 1 |
| A>M | 0 | 0 |

Now depending upon the following condition we will have the value of CY and Z for each comparison. If we get Z=1 we will jump to the NEXT label using the JE NEXT statement (where JE is Jump if equal). If Z=0 we skip the JE NEXT statement and move to the next executable statement (the label to loop back to is written adjacent to the JE command in the JE statement). Thus JE NEXT statement works if and only if A==M (i.e. AL==[SI])

1. If AL==[SI] we jump to the NEXT label, thus ending the CODE SEGMENT ends and hence our code.
2. If AL!=[SI] we skip to the next statement which increments the value of DL Register by 1 using INC DL statement which means that we have counted an element of the array. Then using the INC SI statement we increment the value of SI Register by 1 so that it will now points to the next element in the array. The JMP BACK statement jumps to the BACK label no matter what the value of CY and Z is for comparison of the next element with the identifier(97H) using CMP command.

**Code:**

Chander SEGMENT

NUM DB 01H,02H,03H,04H,05H,0FH,97H

Chander ENDS

CODE SEGMENT

ASSUME CS:CODE , DS:Chander

START: MOV AX, Chander

MOV DS, AX

LEA SI, NUM

MOV AL, 97H

MOV DL, 00H

BACK: CMP AL, [SI]

JE NEXT

INC DL

INC SI

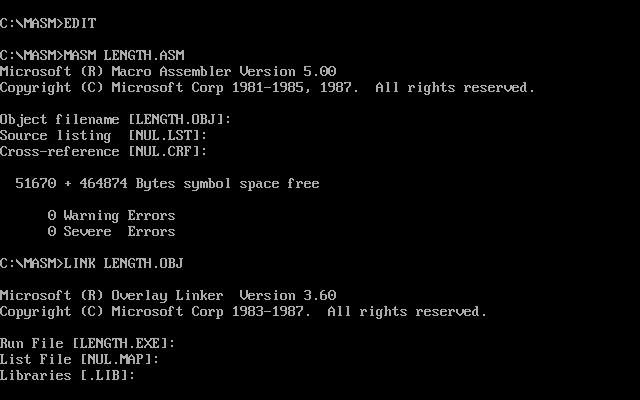
JMP BACK

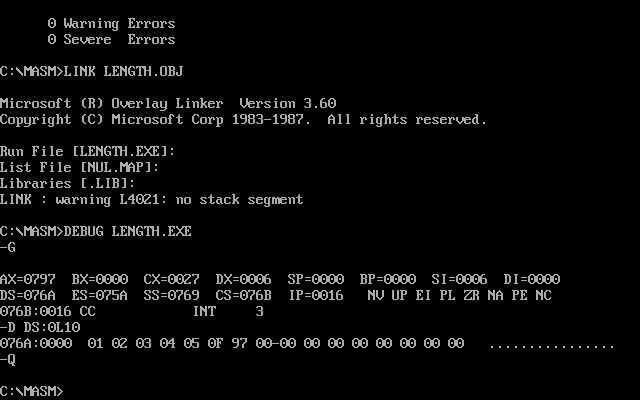
NEXT: INT 3

CODE ENDS

END START

**Result:**





The number of elements in the array are 6 which is shown by the Register DL that stores the value 06. In the output we have DX=0006, where DX=DH+DL; DH=00 and DL=06(the last two digits of Register DX)

**Experiment 7**

**Aim:** To convert a given string of data into its equivalent 2’s complement form.

**Apparatus:** MASM 8086

**Theory:**

* First the Data Segment is defined by segment defining statement. In the Data Segment a string of data is entered that is assigned to STRING1 using DB data type (directive) which is used to declare a byte type variable or we can say we created an array or string of data naming STRING1 using DB. After this the Data Segment ends.
* Then the Code Segment is defined and the labels of the Code and the Data Segments are assigned to the CS and DS segments by ASSUME directive. Then we define a START label. Earlier we have assumed that Chander is the Data Segment but now we are initializing Chander SEGMENT actually as DS. This is done using the MOV command where Chander is loaded into the DS segment Register through AX Register (Accumulator: 16-bit Register). The AX Register here is used because segment Registers cannot be loaded with the immediate data.
* Firstly, effective address of the first element of STRING 1 array is loaded into SI (a 16-bit source index Register) using LEA (Load Effective Address) command. Then a 16-bit counter Register CX is initialized with value 05H as we have to find 2’s compliment of 5 numbers. Then in the BACK label we move the content of SI Register to BH Register using the MOV command. Now carry out the two’s complement operation using NEG BH instruction which two’s complements the data in the BH Register and stores the result back into the BH Register. Then, we move the data which has been complemented and is presently stored in the BH Register back to the memory location being pointed to by SI and simultaneously increment the SI pointer by 1 to point to the next memory location using INC instruction. Then we LOOP back to the BACK label again for the next iteration (the label to loop back to is written adjacent to the LOOP command in the LOOP statement). The LOOP statement does two things:

1. It decreases the value of the CX Register (CX by default) by 1 i.e. CX=CX-1
2. It checks whether or not the value of CX Register is 0
3. If CX==0 then we move to the instruction after the LOOP statement
4. If CX!=0 then we jump to the label specified in the LOOP command.

Here the LOOP BACK statement will be executed 5 times as the value of CX=05H (initially). After this the CODE SEGMENT ends and hence our code.

**Code:**

Chander SEGMENT

STRING1 DB 01H,02H,03H,04H,05H

Chander ENDS

CODE SEGMENT

ASSUME CS:CODE , DS:Chander

START: MOV AX, Chander

MOV DS, AX

LEA SI, STRING1

MOV CX, 05H

BACK: MOV BH, [SI]

NEG BH

MOV [SI], BH

INC SI

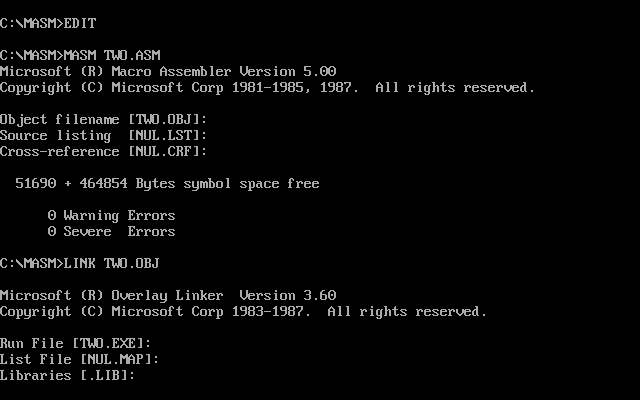
LOOP BACK

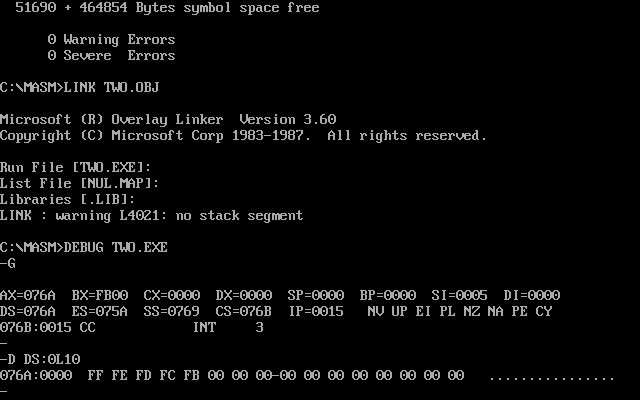
INT 3

CODE ENDS

END START

**Result:**





As we are operating on the string of data STRING1 itself to find the 2’s compliment of the respective elements, thus in the output when we print the array we get the 2’s compliment values of the elements of the array.

**Experiment 8**

**Aim:** To perform factorial of a given number.

**Apparatus:** MASM 8086

**Theory:**

* First the Data Segment is defined by segment defining statement. In the Data Segment a number is entered whose factorial is to be found that is assigned to VAR1 using DW data type (directive) which is used to declare a word type variable. After this the Data Segment ends.
* Then the Code Segment is defined and the labels of the Code and the Data Segments are assigned to the CS and DS segments by ASSUME directive. Then we define a START label. Earlier we have assumed that Chander is the Data Segment but now we are initializing Chander SEGMENT actually as DS. This is done using the MOV command where Chander is loaded into the DS segment Register through AX Register (Accumulator: 16-bit Register). The AX Register here is used because segment Registers cannot be loaded with the immediate data.
* Then a 16-bit counter Register CX is initialized with value stored in word type variable VAR1 using the MOV command. We also initialize the Register AX (Accumulator) with value 0001H. In the label L1 we use the statement MUL CX that multiplies the value in CX Register with the value in AX Register by default and stores the result of the multiplication the AX Register itself. Then we LOOP back to the L1 label again for the next iteration(the label to loop back to is written adjacent to the LOOP command in the LOOP statement). The LOOP statement does two things:

1. It decreases the value of the CX Register (CX by default) by 1 i.e. CX=CX-1
2. It checks whether or not the value of CX Register is 0
3. If CX==0 then we move to the instruction after the LOOP statement
4. If CX!=0 then we jump to the label specified in the LOOP command.

Here the LOOP BACK statement will be executed 7 times as the value of CX=07H (initially) to find the factorial of 07H (the value stored in VAR1). After this the CODE SEGMENT ends and hence our code.

**Code:**

Chander SEGMENT

VAR1 DW 07H

Chander ENDS

CODE SEGMENT

ASSUME CS:CODE , DS:Chander

START: MOV AX, Chander

MOV DS, AX

MOV CX, var1

MOV AX, 0001H

L1: MUL CX

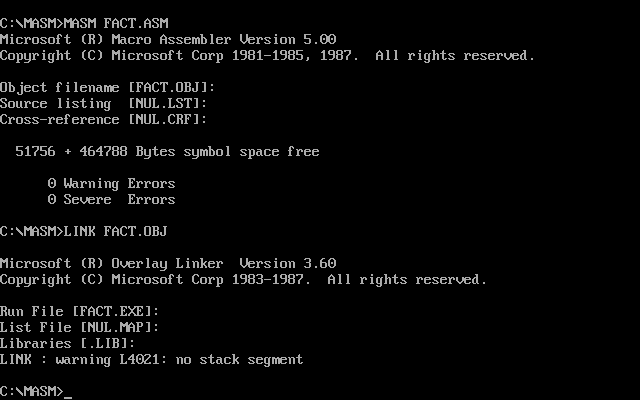
LOOP L1

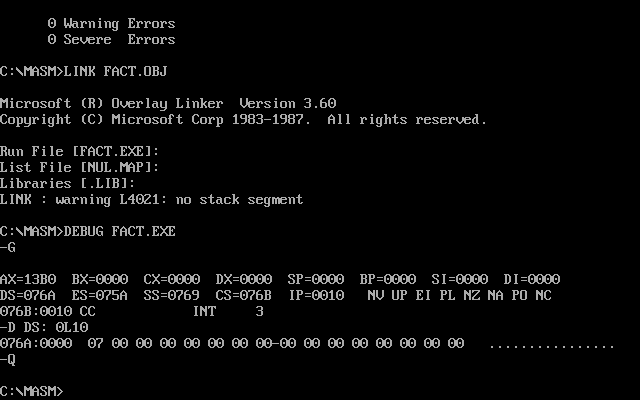
INT 3

CODE ENDS

END START

**Result:**





When we multiply two 8-bit numbers using MUL command the result is stored in the AX Register itself. For ex: MUL BL; using MUL command BL Register that stores an 8-bit value is multiplied by the AL Register by default and the result which 16-bit is stored in AX Register. But for multiplication of two 16-bit numbers the result is stored in the combination of DX and AX i.e. AX\*CX=DX (Higher 16-bits) AX(Lower 16-bits). In the given program on doing MUL CX the factorial value of number 07H obtained is small which is why it is stored completely in AX Register and the DX Register is 0. This is shown in the output as AX=13B0 (Factorial of number 07H); DX=0000.