**CENTRAL UNIVERSITY OF HARYANA**

**Department of Computer Science & Engineering under SOET**



**Micro Processors and Interfacing Lab**

**Practical 1-4**

Submitted by:-

# Ponnaganti pavan kumar

Roll no :- 202102

B.Tech(CSE, 5th Sem)

Submitted to

Dr.nitin goyal

AssociateProfessor

Central University of Haryana (SOET)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Practical 1.(i)- Write an ALP to move block of data without overlap.** | | | | |  |
| **Aim:** | | | | |
| To Write an ALP to Move a Block of Data without Overlap | | | | |
| **Software Required**: | | | | |
| Masm 16 Bit | | | | |
| **Algorithm**: | | | | |
| **1.** Define block of data | | | | |
| **2.** Save memory for block transfer as block2 | | | | |
| **3.** Load block1 into SI | | | | |
| **4.** Load block2 into DI | | | | |
| **5.** Initialize counter | | | | |
| **6.** Move first data into DI | | | | |
| **7.** Repeat step 6 until counter is zero | | | | |
| **8.** End | | | | |
| **Program**: | | | | |
| **.**MODEL SMALL .DATA | | | | |
| BLK1 DB 01,02,03,04,05,06,07,08,09,0AH BLK2 DB 10 DUP (?) | | | | |
| COUNT DW 0AH | | | | |
| .CODE | | | | |
| MOV AX, @DATA MOV DS, AX MOV ES, AX | | | | |
| MOV SI, OFFSET BLK1; MOV DI, OFFSET BLK2 MOV CX, COUNT | | | | |
| AGAIN: CLD | | | | |
| REP MOVSB MOV | | | | |
| OUTPUT: |  |  | | |  |
| BEFORE EXECUTION |  | | |  |
| ================ |  | | |  |
|  | 320 | | | 0 |
|  | ZF AF PF CF | | | 0 |
| 0 0 |  | 0 | |  |  |
| + |  | |  |
| ¦CMD >¦ 1 | 0 | 1 | 2 3 4 5 6 7 |  |  |
| + ¦ DS:0000 |  | 00 FC F3 A4 B4 4C CD 21 | |  |  |
| 000 8ED8 MO DS,AX | DS:0008 | 0 0 0 0 0 0 | | 0 |  |
| 000 8EC0 MO ES,AX | DS:0010 | 0 0 0 0 0 0 | | 0 |  |
| 000 BE0800 MO SI,0008 | DS:0018 | 0 0 0 0 0 0 | | 0 |  |
| 000 BF1200 MO DI,0012 | DS:0020 | 0 0 0 0 0 0 | | 0 |  |
| 000 8B0E1C00 MO CX,[001C] | DS:0028 | 0 0 0 0 0 0 | | 0 |  |
| 001 FC CL | DS:0030 | 0 0 0 0 0 0 | | 0 |  |
| 001 F3A4 REP MOVSB | DS:0038 | 0 0 0 0 0 0 | | 0 |  |
| 001 B44C MO AH,4C | DS:0040 | 0 0 0 0 0 0 | | 0 |  |
| 001 CD21 INT 21 | DS:0048 | 0 0 0 0 0 0 | | 0 |  |



|  |  |  |  |
| --- | --- | --- | --- |
| AFTER EXECUTION  ================  3200  ZF AF PF CF 0 | | | |
| 0 0  + |  | 0 |  |
| ¦CMD > | ¦ 1 | 0 1 2 3 4 5 6 | 7 |
| + ¦ DS:0000 |  | 00 FC F3 A4 B4 4C CD 21 |  |
| 001 B44C MO AH,4C | DS:0008 | 0 0 0 0 0 0 0 |  |
| 001 CD21 INT21 | DS:0010 | 0 0 0 0 0 0 0 |  |
| 001 0102 AD [BP+SI],AX | DS:0018 | 0 0 0 0 0 0 0 |  |
| 001 0304 AD AX,[SI] | DS:0020 | 0 0 0 0 0 0 0 |  |
| 001 050607 AD AX,0706 | DS:0028 | 0 0 0 0 0 0 0 |  |
| 001 0809 OR [BX+DI],CL | DS:0030 | 0 0 0 0 0 0 0 |  |
| 002 0A01 OR AL,[BX+DI] | DS:0038 | 0 0 0 0 0 0 0 |  |
| 002 0203 AD AL,[BP+DI] | DS:0040 | 0 0 0 0 0 0 0 |  |
| 002 0405 AD AL,05  ===============================================  **Result:**  The Block Of Data Defined In The Program Is Moved From  **Verification And Validation**:  Output Is Verified For Different Bytes Of Data And Is Succ Destination Address Without Overlap.  **Conclusion**:  The Block Of Data Defined In The Program Is Moved To D | DS:0048    ===============  Source To Destina  essfully Moved From  estination Without | 0 0 0 0 0 0 0  ===================  tion Without Overlap Successful  Default Source Address To  Overlap And Output Is Verified. | ly. |

**Practical 1.(ii) – Write an ALP to move a block with overlap**

**Aim -** To Write An Alp To Move Block Of Data With Overlap

**Software Required:**

Masm 16 Bit

### Algorithm:

1. Define block of data
2. Reserve memory for block transfer as block2
3. Move block1 address to SI
4. Move block2 address to DI
5. Initialize counter
6. Point DI to block+ n
7. Move block1 data to block2
8. Repeat step 7 until counter is zero
9. End

### Program:

.MODEL SMALL

.DATA

BLK1 DB 01,02,03,04,05,06,07,08,09,0AH BLK2 DB 10 DUP (?) ; SET POINTER

.CODE

MOV AX, @DATA ; MOV THE STARTING ADDRESS MOV DS, AX MOV ES, AX

MOV SI, OFFSET BLK1 ; SET POINTER REG TO BLK1 MOV DI, OFFSET BLK2 REG TO BLK2 MOV CX, 0AH ; SET COUNTER

ADD SI, 0009H ADD DI, 0004H

AGAIN:

MOV AL, [SI] MOV [DI], AL DEC SI DEC DI

DEC CL ; DECREMENT COUNTER

JNZ AGAIN INT 21H END

; TO END PROGRAM MOV AH, 4CH

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OUTPUT:  BEFORE EXECUTION |  | | | | | | | | | | | | |
| ================ |
|  | 0 | 1 | 2 | 3 4 | 5 6 | 7 | 8 | 9 A | B | C | D | E | F |
| DS:0000 |  | B 4 | C | 2 0 | 0 0 | 04 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 |  | 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 |  | 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DS:0030 0 | 0 0 0 0 0 00 0 0 0 0 | 0 | 0 | 0 | 0 |
| DS:0040 0 | 0 0 0 0 0 00 0 0 0 0 | 0 | 0 | 0 | 0 |

================================================================================= AFTER EXECUTION ===============

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 |  | 1 | 2 | 3 4 | 5 6 | 7 | 8 | 9 A | B | C | D | E | F |
| DS:0000 |  | B | 4 | C | 2 0 | 0 0 | 04 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 |  | 0 | 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 |  | 0 | 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 |  | 0 | 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 |  | 0 | 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |

# Result:

The Block Of Data Defined In The Program Is Moved From Source To Destination With Overlap Successfully.

**Verification And Validation**:

Output Is Verified For Different Bytes Of Data And Is Successfully Moved From Default Source Address To Destination Address With Overlap.

### Conclusion:

The Block Of Data Defined In The Program Is Moved To Destination With Overlap And Output Is Verified.

**Practical 1.(iii) - Write an ALP to interchange a block of data.**

### Aim:

To Program To Interchange A Block Of Data

### Software Required:

Masm 16 Bit

### Algorithm:

* 1. Define two sets of data.
  2. Load address of src to SI
  3. Load address of dst to DI
  4. Initialize counter
  5. Interchange data in src and dst **6.** Repeat step 5 until counter = 0.

**7.** End

### Program:

.MODEL SMALL

.DATA

|  |  |  |
| --- | --- | --- |
| SRC DB 10H,20H,30H,40H,50h | | |
|  | DST DB 06,07,08,09,0AH  COUNT EQU 05 |  |
| .CODE |  |  |
|  | MOV AX, @DATA MOV DS, AX  LEA SI,  SRC LEA DI, DST | ; INITIALIZE THE DATA REGISTER |
| BACK: | MOV CL, COUNT | ; INITIALIZE THE COUNTER |
|  | MOVAL,[I] MOV BL, [DI] |  |
|  | MOV [SI], BL MOV [DI], AL INC  SI INC DI  DEC CL | ; INTERCHANGE THE DATA |
|  | JNZ BACK MOV AH, 4CH  INT 21H END | ; REPEAT UNTIL COUNTER BECOMES ZERO |

================================================================================= OUTPUT:

BEFORE EXECUTION

================

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 | 1 | 2 | 3 | 4 | 5 | 0 | 0 | 08 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DS:0030 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 0  AFTER EXECUTION  =============== | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 0 | 0 | 0 | 0 | 0 | 1 | 2 | 30 | 4 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Result:**

Program Is Executed Without Errors And The Output Is Verified

**Verification And Validation**:

Output Is Verified And Is Found Correct

#### Conclusion:

The Blocks Of Data Defined In The Program Is Interchanged And Output Is Verified.

## Practical 2.(i) - Write an ALP to add 2 Multibyte numbers.

**Aim**: To Write An Alp To Add 2 Multibyte No.s

**Software Required**:

Masm 16 Bit

### Algorithm :

1. Initialize the MSBs of sum to 0
2. Get the first number.
3. Add the second number to the first number.
4. If there is any carry, increment MSBs of sum by 1.
5. Store LSBs of sum. 6. Store MSBs of sum.

**Program**:

.MODEL SMALL

.DATA

|  |  |  |
| --- | --- | --- |
| N1 DQ 122334455667788H ; FIRST NUMBER N2 DQ  122334455667788H ; SECOND NUMBER | | |
| .CODE | SUM DT ? | ; INITIALIZE THE DATA REGISTER |
| BACK  : | MOV AX, @DATA MOV DS, AX  LEA SI, N1  LEA DI, N2 LEA BX, SUM  MOV CL, 04H CLC  MOV AX, [SI] ADC AX, [DI]  MOV [BX], AX INC SI INCSI INC DI  INC  DI | ; POINTER TO FIRST NUMBER  ; POINTER TO SECOND NUMBER  ; COUNTER FOUR WORD  ;MOVE FIRST WORD |
| OVER: |  |  |
|  | JNZ BACK JNC OVER  MOV AX,  0001H MOV [BX], AX | ; REPEAT UNTIL COUNTER BECOMES ZERO |
| MOV AH, 4CH INT 21H  END |  |

================================================================================= OUTPUT:

BEFORE EXECUTION

================

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 01 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

AFTER EXECUTION

===============

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 8 | 7 | 6 | 5 | 4 | 3 | 2 | 01 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 |
| DS:0010 1 | E | C | A | 8 | 6 | 4 | 02 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Result:*

Program Is Executed Without Errors And The Output Is Verified

**Verification And Validation**:

Output Is Verified And Is Found Correct

#### Conclusion:

The Addition Of Two Multibye Data Is Done And The Output Is Verified

## Practical 2.(ii) - Write an ALP to subtract two Multibyte numbers

### Aim:

To Write An Alp To Subtract Two Multibyte Numbers

**Software Required**: MASM 16 BIT

# Algorithm :

1. Initialize the MSBs of difference to 0
2. Get the first number
3. Subtract the second number from the first number.
4. If there is any borrow, increment MSBs of difference by 1.
5. Store LSBs of difference
6. Store MSBs of difference

**Program**:

.MODEL SMALL

.DATA

.CODE

BACK

:

N1 DQ 122334455667788H ; FIRST NUMBER N2 DQ 111111111111111H ; SECOND NUMBER RESULT DT ?

MOV AX, @DATA ; INITIALIZE THE DATA REGISTER MOV DS, AX

LEA SI, N1 ; POINTER TO FIRST NUMBER

LEA DI, N2 ; POINTER TO SECOND NUMBER

LEA BX,MOV CX, RESULT 04H ; COUNTER FOUR WORD CLC

MOV AX, [SI] ; MOVE FIRST WORD SBB AX, [DI]

MOV [BX], AX INC SI

INC SI ; MOVE SI, DI CONTENTS INC DI

INC DI

INC BX ; INCREMENT BX TO STORE RESULTS INC BX

LOOP BACK MOV AH, 4CH INT 21H

END

================================================================================= OUTPUT: BEFORE EXECUTION ================

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 2 | 3 | 4 5 | 6 7 | 8 | 9 A | B C | | D E | F |
| DS:0000 |  | 4 E F | B | 4 C 2 | 00 | 8 | 7 6 5 | 4 3 | | 2 | 0 |
| DS:0010 1 1 1 1 1 1 1 01 0 0 0 0 0 0 0 0 DS:0020 0 0 0 0 0 0 0 00 0 0 0 0 0 0 0 0 | | | | | | | | | | | |
| DS:0030 | 0 0 0 | | 0 | 0 0 0 | 00 | 0 | 0 0 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 0 0 | | 0 | 0 0 0 | 00 | 0 | 0 0 0 | 0 | 0 | 0 | 0 |

AFTER EXECUTION

===============

0 1 2 3 4 5 6 7 8 9 A B C D E F

DS:0000 4 E F B 4 C 2 00 8 7 6 5 4 3 2 0 DS:0010 1 1 1 1 1 1 1 01 7 6 5 4 3 2 1 0 DS:0020

0 0 0 0 0 0 0 00 0 0 0 0 0 0 0 0

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DS:0030 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |

=================================================================================

**Result:**

Program Is Executed Without Errors And The Output Is Verified

**Verification And Validation**:

Output Is Verified And Is Found Correct

#### Conclusion:

The Subtraction Of Two Multibye Data Is Done And The Output Is Verified

**Practical 2.(iii)- Write An Alp To Multiply Two 16-Bit Numbers.**

### Aim:

To Write An Alp To Multiply Two 16-Bit Numbers

### Software Required:

Masm 16 Bit

### Algorithm:

1. Get The Multiplier.
2. Get The Multiplicand 3. Initialize The Product To 0.
3. Product = Product + Multiplicand
4. Decrement The Multiplier By 1
5. If Multiplicand Is Not Equal To 0,Repeat From Step (D) Otherwise Store The Product.

### Program:

.MODEL SMALL

.STACK .DATA

MULTIPLICAND DW 00FFH; FIRST WORD HERE MULTIPLIER DW 00FFH; SECOND WORD HERE

.CODE START: PRODUCT DW 2 DUP(0); RESULT OF MULIPLICATION HERE

MOV AX, @DATA MOV DS, AX

MOV AX, MULTIPLICAND MUL MULTIPLIER

MOV PRODUCT, AX MOV PRODUCT+2,

DX MOV AH, 4CH INT 21H END

START

================================================================================= OUTPUT: BEFORE EXECUTION

================

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 1 | 2 | 3 4 | 5 6 | 7 | 8 | 9 A | B | C | D | E | F |
| DS:0000 | 1 0 | 0 | B 4 | C 2 | 00 | F | 0 F | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040  AFTER EXECUTION  =============== | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 1 | 2 | 3 4 | 5 6 | 7 | 8 | 9 A | B | C | D | E | F |
| DS:0000 | 1 0 | 0 | B 4 | C 2 | 00 | F | 0 F | 0 | 0 | F | 0 | 0 |
| DS:0010 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 0 | 0 | 0 0 | 0 0 | 00 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

=================================================================================

**Result:**

Program Is Executed Without Errors And The Output Is Verified

**Verification And Validation**:

Output Is Verified And Is Found Correct

#### Conclusion:

The Multiplication Of Two 16 Bit Data Is Done And The Output Is Verified

Practical 2.(iv) - Write an ALP to perform the conversion from BCD to binary.

AIM: TO DEVELOP AND EXECUTE AND ASSEMBLY LANGUAGE PROGRAM TO PERFORM THE CONVERSION FROM BCD TO BINARY

SOFTWARE REQUIRED: MASM 16 BIT PROGRAM:

.MODEL SMALL

.DATA

|  |  |
| --- | --- |
| . CODE |  |
| BCD\_INPUT DB 61H  MOV AX, @DATA | ; BCD NUMBER IN\_VALUE DB (?) |
| MOV DS, AX | ; INITIALIZE DATA SEGMENT MOV AL, BCD\_INPUT |
| MOV BL, AL | ; MOVE NUMBER TO AL REGISTER AND BL, 0FH |

AND AL, 0F0H MOV CL, 04H ROR AL, CL MOV BH, 0AH MUL BH ADD AL, BL

MOV IN\_VALUE, AL ; STORE THE BINARY EQUIVALENT NUMBER MOV AH, 4CH INT 21H

END ; END PROGRAM

================================================================================= OUTPUT:

BEFORE EXECUTION

================

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 6 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 0  AFTER EXECUTION  =============== | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 6 | 3 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

=================================================================================

**RESULT**: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

**VERIFICATION AND VALIDATION:** OUTPUT IS VERIFIED AND IS FOUND CORRECT

**CONCLUSION**: THE CONVERSION OF NUMBER FROM BCD TO BINARY IS DONE AND THE OUTPUT IS VERIFIED

## Program 3.(i) - Write an ALP to separate odd and even numbers.

**AIM:** TO **WRITE AN ALP TO SEPARATE ODD AND EVEN NUMBERS**

**SOFTWARE REQUIRED**: MASM 16 BIT

**PROGRAM**:

.MODEL SMALL

.DATA

ARRAY DB 12H, 98H, 45H, 83H, 28H, 67H, 92H, 54H, 63H, 76H ARR\_EVEN DB 10 DUP (?)

ARR\_ODD DB 10 DUP (?)

. CODE

MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT MOV DS, AX

MOV CL, 0AH ; INITIALIZE THE COUNTER

XOR DI, DI ; INITIALIZE THE ODD POINTER

XOR SI, SI ; INITIALIZE THE EVEN POINTER BACK LEA BP, ARRAY

:

MOV AL, DS:[BP] ; GET THE NUMBER

TEST AL, 01H ; MASK ALL BITS EXCEPT LSB

JZ NEXT ; IF LSB = 0 GOT TO NEXT

LEA BX, ARR\_ODD MOV [BX+DI], AL

INC DI ; INCREMENT THE ODD NEXT POINTER JMP SKIP

:

LEA BX, ARR\_EVEN

MOV [BX+SI], AL

SKIP:

INC SI ; INCREMENT THE EVEN POINTER

INC BP ; INCREMENT ARRAY BASE POINTER

LOOP BACK ; DECREMENT THE

COUNTER MOV AH, 4CH

INT 21H

END ; END PROGRAM

**================================================================================= OUTPUT:**

**BEFORE EXECUTION**

**================**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | |  | **1** | **2** | | **3** | **4** | | **5** | **6** | | **7** |  | **8** | **9** |  | **A** | **B C** | | | **D E** | | | **F** |  |
| **DS:0000** |  | | **1** |  | **9** | |  | **4** | |  | **8** | |  | **2** |  |  | **6** |  | **9** | | | **54** | | |  | **6** |
| **7** | **0** | **0** | | **0** | | | **0** | | | **0** | | | **0 DS:0010** | | | | | **0** | | | | **0** | | | **0** | |
| **0** | **0** |  | **0** |  |  | **0** |  |  | **00** |  |  | **0** |  | **0** |  |  | **0** |  |  | **0** |  |  | **0** |  |  | **0** |
| **0** | **0** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **DS:0020** |  | **0** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  | **00** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  |
| **DS:0030** |  | **0** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  | **00** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  |
| **DS:0040** |  | **0** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  | **00** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  | **0** | **0** |  |

**AFTER EXECUTION**

**===============**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **DS:0000** | **1** | **9** | **4** | **8** | **2** | **6** | **9** | **54** | **6** | **7** | **1** | **9** | **2** | **9** | **5** | **7** |
| **DS:0010** | **0** | **0** | **0** | **0** | **4** | **8** | **6** | **63** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0020** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0030** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0040** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**=================================================================================**

**RESULT**: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED. **VERIFICATION AND VALIDATION**: OUTPUT IS VERIFIED AND IS FOUND CORRECT **CONCLUSION: THE ODD AND EVEN NUMBERS ARE SEPERATED AND OUTPUT IS VERIFIED**

## Program 3.(ii) - Write an ALP to separate positive and negative numbers.

**AIM**: TO WRITE AN ALP TO SEPARATE POSITIVE AND NEGATIVE NUMBERS

**SOFTWARE REQUIRED**: MASM 16 BIT

**PROGRAM**:

**.MODEL SMALL**

.**DATA**

|  |  |
| --- | --- |
| . **CODE**  **BACK**:  **NEXT**:  SKIP: | ARRAY DB 12H, -98H,-45H,83H,-28H, 67H, 92H, -54H, -63H, 76H NEGI DB 10 DUP (?)  POSI DB 10 DUP (?)  MOV AX, @DATA ; INITIALIZE THE DATA SEGMENT MOV DS, AX  MOV CL, 0AH ; INITIALIZE THE COUNTER  XOR DI, DI ; INITIALIZE THE POINTER FOR NEGATIVE NUMBER  XOR SI, SI ; INITIALIZE THE POINTER FOR POSITIVE NUMBER LEA BP, ARRAY  MOV AL, DS:[BP] ; GET THE NUMBER  TEST AL, 80H ; MASK ALL BITS EXCEPT MSB JZ NEXT ; IF LSB = 0 GOT TO NEXT  LEA BX, NEGI MOV [BX+DI], AL  INC DI ; INCREMENT THE NEGATIVE POINTER JMP SKIP  LEA BX, POSI MOV  [BX+SI], AL  INC SI ; INCREMENT THE POSITIVE POINTER  INC BP ; INCREMENT ARRAY BASE POINTER LOOP BACK ; DECREMENT THE  COUNTER MOV AH, 4CH INT  21H  END ; END PROGRAM |

**=================================================================================**

**OUTPUT:**

BEFORE EXECUTION

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 | 1 | 6 | B | 8 | D | 6 | 9 | A | 9 | 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AFTER EXECUTION |  | | | | | | | | | | | | | | | |
| =============== | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 | 1 | 6 | B | 8 | D | 6 | 9 | A | 9 | 7 | B | 8 | D | 9 | A | 9 |
| DS:0010 | 0 | 0 | 0 | 0 | 1 | 6 | 6 | 76 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

===================================================================

**RESULT**: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

**VERIFICATION AND VALIDATION**: OUTPUT IS VERIFIED AND IS FOUND CORRECT

**CONCLUSION**: THE POSITIVE AND NEGATIVE NUMBERS ARE SEPERATED AND OUTPUT IS VERIFIED

## Program 3.(iii) - Write an ALP to check bitwise palindrome or not.

**AIM:** TO **WRITE AN ALP TO CHECK BITWISE PALINDROME OR NOT**

**SOFTWARE REQUIRED**: MASM 16 BIT

**PROGRAM**:

**.MODEL SMALL**

**.STACK 100**

**PRINTSTRING MACRO MSG**

**MOV AH, 09H ; MACRO TO DISPLAY THE MESSAGE MOV DX, OFFSET MSG**

**INT 21H ENDM**

|  |  |  |
| --- | --- | --- |
| **.DATA** | **NUM DB 0FFH**  **TABLE DB 81H, 42H, 24H, 18H**  **MSG1 DB 'THE NUMBER EXHIBITS BITWISE PALINDROME:$'**  **MSG2 DB 'THE NUMBER DOESNOT EXHIBITS BITWISE PALINDROM:$'** | |
| **. CODE**  **L1:**  **NEXT:**  **SKIP:** | **MOV AX, @DATA MOV DS, AX**  **LEA SI, TABLE MOV CX, 0004H XOR AX, CX**  **MOV AL, NUM AND AL, [SI] JPE NEXT**  **PRINTSTRING MSG2 JMP SKIP**  **INC SI DEC CX**  **COUNTER JNZ L1 PRINTSTRING MSG1**  **MOV AH, 4CH INT 21H** | **; INITIALIZE THE DATA SEGMENT**  **; SET COUNTER**  **; CLEAR AX REGISTER**  **; DISPLAY MESSAGE 2**  **; INCREMENT POINTER**  **; DECREMENT**  **; DISPLAY MESSAGE 1** |
|  | **END** | **; END PROGRAM** |

**OUTPUT:**

**;C:\8086> ENTER THE FILE NAME**

**; THE NUMBER EXHIBITS BITWISE PALINDROME**

**=================================================================================**

**RESULT**: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED.

**VERIFICATION AND VALIDATION**: OUTPUT IS VERIFIED AND IS FOUND CORRECT.

**CONCLUSION: THE GIVEN NUMBER EXHIBITS BITWISE PALINDROME**

## Program 4.(i) - Write an ALP to find largest number from a given array.

**AIM:** TO **WRITE AN ALP TO FIND LARGEST NO FROM THE GIVEN ARRAY.**

**SOFTWARE REQUIRED**: MASM 16 BIT

**PROGRAM**:

**.MODEL SMALL**

**.STACK 100**

**.DATA**

|  |  |  |
| --- | --- | --- |
| **NUM DB 12H, 37H, 01H, 36H, 76H ; INITIALISE DATA** | | |
| **.CODE**  **LOOP1:** | **SMALL DB (?)**  **MOV AX, @DATA MOV DS, AX MOV CL, 05H MOV AL, 00H LEA SI, NUM**  **CMP AL, [SI] JNC LOOP2 MOV AL, [SI]** | **; TO STORE LARGEST NUM**  **; INITIALIZE THE DATA SEGMENT**  **; SET COUNTER**  **; POINTER TO NUMBER**  **; COMPARE 1ST AND 2ND NUMBER** |
|  | ==========================================================================  =======  **OUTPUT**:  **BEFORE EXECUTION** | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 | 1 | 3 | 0 | 3 | 7 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**AFTER EXECUTION**

===============

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| DS:0000 | 1 | 3 | 0 | 3 | 7 | 7 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DS:0040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

============================================================================

=====

**RESULT**: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED. **VERIFICATION AND VALIDATION**: OUTPUT IS VERIFIED AND IS FOUND CORRECT **CONCLUSION**: THE LARGEST NUMBER IN THE GIVEN ARRAY IS 76 AND OUTPUT IS VERIFIED.

## Practical 4.(ii) - WRITE AN ALP TO FIND SMALLEST NO FROM THE GIVEN ARRAY

**AIM**: TO WRITE AN ALP TO FIND SMALLEST NO FROM THE GIVEN ARRAY.

**SOFTWARE REQUIRED**: MASM 16 BIT

**PROGRAM**:

.**MODEL** SMALL

.**STACK** 100

.**DATA**

NUM DB 12H, 37H, 01H, 36H, 76H ; INITIALISE DATA

|  |  |
| --- | --- |
| SMALL DB (?)  .**CODE**  MOV AX, @DATA MOV DS, AX MOV CL, 05H  MOV AL, 0FFH  LEA SI, NUM  **LOOP1**  : CMP AL, [SI] JC LOOP2 MOV AL, [SI] | ; TO STORE SMALLEST NUM  ; INITIALIZE THE DATA SEGMENT  ; SET COUNTER  ; POINTER TO NUMBER  ; COMPARE 1ST AND 2ND NUMBER |

**LOOP2 INC SI**

**: DEC**

**CL JNZ LOOP1**

**MOV SMALL, AL MOV AH, 4CH INT 21H**

**END ; END PROGRAM**

**================================================================================= OUTPUT:**

**BEFORE EXECUTION**

**================**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **DS:0000** | **1** | **3** | **0** | **3** | **7** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0010** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0020** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0030** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0040** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**AFTER EXECUTION**

**===============**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **DS:0000** | **1** | **3** | **0** | **3** | **7** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0010** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0020** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0030** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0040** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**=================================================================================**

**RESULT:** PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED.

**VERIFICATION AND VALIDATION:** OUTPUT IS VERIFIED AND IS FOUND CORRECT.

**CONCLUSION:** THE SMALLEST IN THE GIVEN NUMBER IS 01 AND OUTPUT IS VERIFIED

## Practical 4.(iii) - WRITE AN ALP TO SORT A GIVEN SET OF 16BIT UNSIGNED INTEGERS INTO ASCENDING ORDER USING BUBBLE SORT ALGORITHM

**AIM:** TO WRITE AN ALP TO SORT A GIVEN SET OF 16BIT UNSIGNED INTEGERS INTO ASCENDING ORDER USING BUBBLE SORT ALGORITHM **SOFTWARE REQUIRED**: MASM 16 BIT

**PROGRAM**:

.**MODEL** SMALL

.**DATA**

A DB 23H, 45H, 55H, 22H, 64H ; INITIALISE DATA SIZE1 DW ($-A)

|  |  |
| --- | --- |
| .  **CODE** MOV AX, @DATA MOV DS, AX | ; CALCULATE SIZE OF NUMBERS  ; INITIALIZE THE DATA SEGMENT |
| MOV BX, SIZE1  DEC BX **OUTLOOP**: | ; THE NO. OF DATA BYTES IS INITIALIZE IN BX |
| MOV CX, BX | ; SAVE COUNTER IN CX REGISTER |
| MOV SI, 00  **INLOOP**: | ; INITIALISE POINTER |
| MOV AL, A[SI] | ; LOAD THE DATA INTO AL POINTED BY SI |
| INC SI | ; INCREMENT THE POINTER |
| CMP AL, A[SI] | ; IS CONTENT OF AL<SI POINTED |
| JB NEXT | ; YES, GO NEXT |
| XCHG AL, A[SI] | ; NO, EXCHANGE TWO DATA |
| MOV A[SI-1], AL MEMORY  **NEXT** LOOP INLOOP  : DEC BX  JNZ OUTLOOP MOV AH, 4CH INT 21H | ; MOVE TILL END OF |
| END | ; END PROGRAM |

**================================================================================= OUTPUT:**

**BEFORE EXECUTION**

**================**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **DS:0000** | **B** | **4** | **C** | **2** | **2** | **4** | **5** | **22** | **6** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0010** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0020** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0030** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0040** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**AFTER EXECUTION**

**===============**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **DS:0000** | **B** | **4** | **C** | **2** | **2** | **2** | **4** | **55** | **6** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0010** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0020** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **DS:0030** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **00** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**DS:0040 0 0 0 0 0 0 0 00 0 0 0 0 0 0 0 0**

**============================================================**

**RESULT**: PROGRAM IS EXECUTED WITHOUT ERRORS AND THE OUTPUT IS VERIFIED

**VERIFICATION AND VALIDATION**: OUTPUT IS VERIFIED AND IS FOUND CORRECT

**CONCLUSION:** THE GIVEN NUMBERS ARE ARRANGED IN ASCENDING ORDER AND THE OUTPUT IS VERIFIED

**THE END**