Experiments

**(Minimum 10 Exercises (at least 2 questions from each part I, II, III & IV) ) : 2 Hrs/week**

1. **Assembly Language Programming Exercises/Experiments using 8086 Trainer kit**
   1. Implementation of simple decimal arithmetic and bit manipulation operations.
   2. Implementation of code conversion between BCD, Binary, Hexadecimal and ASCII.
   3. Implementation of searching and sorting of 16-bit numbers.

### Exercises/Experiments using MASM (PC Required)

1. Study of Assembler and Debugging commands.
2. Implementation of decimal arithmetic (16 and 32 bit) operations.
3. Implementation of String manipulations.
4. Implementation of searching and sorting of 16-bit numbers.

### Interfacing Exercises/Experiments with 8086 trainer kit through Assembly Language Programming

1. Interfacing with stepper motor - Rotate through any given sequence.
2. Interfacing with 8255 (mode0 and mode1 only).
3. Interfacing with 8279 (Rolling message, 2 key lockout and N-key rollover implementation).
4. Interfacing with Digital-to-Analog Converter.

### Exercises/Experiments using 8051 trainer kit

1. Familiarization of 8051 trainer kit by executing simple Assembly Language programs such as decimal arithmetic and bit manipulation.
2. Implementation of Timer programming (in mode1).

# LIST OF LAB EXPERIMENTS

### Assembly Language Programming Exercises/Experiments using 8086 Trainer kit

* 1. ADDITION OF TWO 8 - BIT NUMBERS (WITH CARRY)
  2. SUBTRACTION OF TWO SIXTEEN BIT NUMBERS
  3. MULTIPLICATION OF TWO 16 - BIT NUMBERS
  4. CONVERSION OF HEXADECIMAL NUMBER TO BCD
  5. CONVERSION OF BCD NUMBER TO HEXADECIMAL
  6. SEARCHING OF SMALLEST DATA FROM AN ARRAY
  7. EVALUATE THE EXPRESSION USING STACK
  8. SEARCHING OF SMALLEST DATA FROM AN ARRAY USING SUBROUTINE

### Exercises/Experiments using MASM (PC Required)

* 1. PROGRAM FOR 16 BIT ADDITION.
  2. PROGRAM FOR 16 BIT SUBTRACTION
  3. PROGRAM FOR 32 BIT ADDITION
  4. PROGRAM FOR 32 BIT SUBTRACTION
  5. PROGRAM TO READ AND PRINT A STRING
  6. PROGRAM TO REVERSE A STRING
  7. PROGRAM TO CHECK WHETHER A GIVEN STRING IS PALINDROME OR NOT
  8. PROGRAM TO CONCATENATE TWO STRINGS
  9. PROGRAM TO REPLACE A CHARACTER
  10. PROGRAM TO REPLACE ALL VOWELS
  11. PROGRAM TO SEARCH A 16 BIT NUMBER

### Interfacing Exercises/Experiments with 8086 trainer kit through Assembly Language Programming

* 1. STEPPER MOTOR IN FORWARD DIRECTION
  2. INTERFACING WITH 8255 (MODE0 AND MODE1 ONLY).

### Exercises/Experiments using 8051 trainer kit

* 1. MULTI BYTE ADDITION
  2. MULTI BYTE SUBTRACTION
  3. IMPLEMENTATION OF TIMER PROGRAMMING

# Assembly Language Programming Exercises/Experiments using 8086 Trainer kit

* 1. **ADDITION OF TWO 8 - BIT NUMBERS (WITH CARRY)**

## AIM :

To add two 8 bit hexadecimal numbers residing in memory and store the result & carry in different memory locations.

**PROGRAM**

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1004**  **1008**  **100B**  **100D**  **100F**  **1011**  **1015** | **8A 26 00 11**  **8A 1E 01 11**  **C6 C0 00**  **00 DC**  **73 02**  **FE C0**  **88 06 02 11**  **88 26 03 11** | **MOV AH,[1100]**  **MOV BL,[1101] MOV AL,00 ADD AH,BL JNC 1011**  **INC AL**  **MOV [1102],AL MOV [1103],AH** | **First input byte in AH Register**  **Second input byte in BL Register**  **Jump if no carry**  **Carry in location 1102 Result in location 1103** |

# RESULT

**INPUT OUTPUT**

[1100]=FF [1102]=01

[1101]=FF [1103]=FE

# SUBTRACTION OF TWO SIXTEEN BIT NUMBERS

**AIM :**

To subtract two 16 bit hexadecimal numbers residing in memory and store the difference in memory location.

# PROGRAM

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | | **MNEMONICS** | | **REMARKS** | |
| **1000**  **1001**  **1002**  **1003**  **1004**  **1005**  **1006**  **1007**  **1008**  **1009**  **100A**  **100B**  **100C** | **8A**  **06**  **00**  **11**  **2B**  **06**  **02**  **11**  **89**  **06**  **00**  **12**  **F4** | | **MOV AX,[1100]**  **SUB AX,[1102]**  **MOV [1200],AX**  **HLT** | | **First input word in AX register**  **Second input word in location 1102**  **Result in location 1200** | |
| **RESULT**  **INPUT** | |  | | **OUTPUT** | |
| Minuend : [1100] =99 | | [1101]=99 | | [1200]=62 [1201]=FD | |
| Subtrahend: [1102] =36 | | [1103]=9C | |  | |

**AIM :**

# MULTIPLICATION OF TWO 16 - BIT NUMBERS

To multiply two 16 bit hexadecimal numbers and store the result in memory. The input data is also to be fetched from memory.

**PROGRAM**

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1004**  **1008**  **100A**  **100E**  **1012** | **8B 06 00 11**  **8B 06 02 11**  **F7 E3**  **89 06 00 12**  **89 16 02 12**  **F4** | **MOV AX,[1100] MOV BX,[1102] MUL BX**  **MOV [1200],AX**  **MOV [1202],DX HLT** | **Move the first number to AX**  **Move second number to BX**  **Multiply the contents of BX and AX**  **Lower word of result in 1200**  **Higher order word of result in 1202**  **End of the program** |

|  |  |  |  |
| --- | --- | --- | --- |
| **RESULT INPUT** |  | **OUTPUT** |  |
| 1100 = 03 | 1101= 04 | 1200= 18 | 1201= 35 |
| 1102= 08 | 1103= 07 | 1202= 1C | 1203= 00 |

**AIM :**

# CONVERSION OF HEXADECIMAL NUMBER TO BCD

To convert hexadecimal number in to its corresponding BCD and store the result in memory. The input data is also to be fetched from memory.

**PROGRAM**

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1004**  **1007**  **1009**  **100B**  **100D**  **1010**  **1012**  **1014**  **1016**  **1017**  **1019**  **101B**  **101D**  **101F**  **1023** | **8A 26 00 11**  **80 E4 F0**  **B1 04**  **B0 00 D2 EC 80 FC 00**  **74 0D**  **B7 00**  **04 16**  **27**  **73 02**  **FE C7 FE CC 75 F5**  **8A 26 00 11**  **80 E4 0F** | **MOV AH,[1100] AND AH,F0**  **MOV CL,04 MOV AL,00 SHR AH,CL CMP AH,00 JE 101F MOV BH,00 ADD AL,016 DAA**  **JNC 101B INC BH DEC AH JNZ 1014**  **MOV AH,[1100]**  **AND AH,0F** | **Input byte brings to AH register**  **Masking of bits with F0**  **Bits shift right 4times**  **Decimal adjust for addition**  **Masking of bits with 0F** |

|  |  |  |  |
| --- | --- | --- | --- |
| **1026**  **1029**  **102B**  **102E**  **1030**  **1031**  **1033**  **1035**  **1039**  **103C** | **80 FC 0A**  **72 03**  **80 C4 06**  **00 E0**  **27**  **73 02**  **FE C7**  **88 3E 01 11**  **A2 02 11 F4** | **CMP AH, 0A JC 102E ADD AH,06 ADD AL,AH DAA**  **JNC 1035 INC BH**  **MOV[1101],BH**  **MOV[1102],AL HLT** | **Converting lower nibble to decimal**  **Higher bit of result in 1101**  **Lower bit of result in 1102 End of the program** |

**RESULT**

**(I) INPUT**

|  |
| --- |
| ARITHMATIC OPERATIONS |
| **99=90=09 (masking & shifting) 09\*16 =ADD 16, 9 TIMES=144(repeated addition equivalent to multiplication) ADD 09+144=153(result is 153)** |

[1100] = 99

## OUTPUT

[1101] = 01

[1102] = 53

## (II) INPUT

|  |
| --- |
| ARITHMATIC OPERATIONS |
| **FF=F0=0F (masking & shifting) 0F\*16 =ADD 16**  **,15 TIMES=240(repeated addition equivalent to multiplication) ADD 15+240=255(result is 255)** |

[1100]=FF

## OUTPUT

[1101]=02

[1102]=55

**AIM :**

# CONVERSION OF BCD NUMBER TO HEXADECIMAL

To convert BCD number in to its corresponding hexadecimal number and store the result in memory. The input data is also to be fetched from memory. (Hex limit up to 63, BCD limit up to 99.)

## PROGRAM

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1003**  **1005**  **1007**  **1009**  **100B**  **100C**  **100E**  **1010**  **1012**  **1014**  **1016**  **1018**  **101A**  **101C** | **A0 01 11**  **B1 00**  **3C 16**  **72 07**  **2C 16**  **2F**  **FE C1 EB F5 3C 0A**  **72 02**  **2C 06**  **88 C3**  **B0 10 F6 E1 00 D8** | **MOV AL,[1101] MOV CL,00 CMP AL,016 JB 1010**  **SUB AL,016 DAS**  **INC CL JMP 1005 CMP AL,0A JB 1016 SUB AL,06**  **MOV BL,AL MOV AL,10 MUL CL ADD AL,BL** | **Byte in AL register**  **Compare with 16 Jump on below 16**  **Decimal adjust for subtract**  **Compare with 9 Jump on below 9 Subtract 06 from**  **number to get hex digits A to F**  **Multiply with 10** |

|  |  |  |  |
| --- | --- | --- | --- |
| **101E**  **1021** | **A2 02 11 F4** | **MOV [1102],AL HLT** | **Result in location 1102** |

**RESULT**

|  |
| --- |
| ARITHMATIC OPERATIONS |
| **If 63>16 then 63-16=47 ; CL =1**  **again 47>16 then 47-16=31 ; CL =2**  **again 31>16 then 31-16=15 ; CL =3**  **now 15<16 then compare with 10 15>10 then 15-06=0F= gives F CL \*10=3\*10=30**  **30+0F=3F** |

**(I) INPUT** [1101]=63 **OUTPUT** [1102]=3F

**(II) INPUT** [1101]=20 **OUTPUT** [1102]=14

|  |
| --- |
| ARITHMATIC OPERATIONS |
| **If 20>16 then 20-16=04; CL cl=1**  **Then compare with 10 04<10 then 10\* CL =10\*1=10 then 10+04=14** |

**AIM :**

# SEARCHING OF SMALLEST DATA FROM AN ARRAY

To search the smallest data from an array and store the result in memory. The input data is also to be fetched from memory.

## PROGRAM

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1004**  **1007**  **1009**  **100B**  **100E**  **1010**  **1013**  **1014**  **1016**  **1018**  **101C** | **8B 0E 00 11**  **BE 01 11**  **FE C9 8A 24**  **3A 64 01**  **72 03**  **8A 64 01**  **46**  **FE C9 75 F3**  **88 26 00 12**  **F4** | **MOV CL,[1100] MOV SI,1101**  **DEC CL MOV AH,[SI]**  **CMP AH,[SI+1] JC 1013**  **MOV AH,[SI+1] INC SI**  **DEC CL JNZ 100B**  **MOV [1200],AH**  **HLT** | **Store count in 1100**  **Pointed first element of array**  **Compare first & second elements If AH contains the smallest then**  **jump to 1013**  **Checking for the end of the array**  **Result in location 1200** |

|  |  |  |  |
| --- | --- | --- | --- |
| **RESULT** |  | | |
| **(I) INPUT** | **OUTPUT** | **(II) INPUT** | **OUTPUT** |
| [1100]= 07 | [1200]=59 | [1100]=05 | [1200]=01 |
| [1101]=AB [1102]=DE  [1103]=EC |  | [1101]=01  [1102]=09  [1103]=07 |  |
| [1104]=59 [1105]=FB  [1106]=AD |  | [1104]=06  [1105]=05 |  |
| [1107]=69 |  |  |  |

**EVALUATE THE EXPRESSION USING STACK**

**AIM :**

To evaluate the expression (a+b)\*(c+d) using stack operations

**PROGRAM**

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1003**  **1007**  **1009**  **100A**  **100D**  **1011**  **1013**  **1015**  **1016**  **1018**  **101B** | **A1 00 11**  **8B 1E 01 11**  **01 D8**  **50**  **A1 02 11**  **8B IE 03 11**  **01 D8**  **89 C3**  **58**  **F7 E3 A3 00 12 F4** | **MOV AX,[1100] MOV BX,[1101] ADD AX,BX PUSH AX**  **MOV AX,[1102] MOV BX,[1103] ADD AX,BX MOV BX,AX POP AX**  **MUL BX**  **MOV [1200],AX HLT** | **Move 1st number to AX Move 2nd number to BX Add 1st and 2nd number Push result to stack Move 3rd number to AX Move 4th number to BX**  **Move the result to BX**  **Pop previous result from stack**  **Multiply the results Store final result to 1200** |

|  |  |  |
| --- | --- | --- |
| **RESULT**  **(I) INPUT** |  | **OUTPUT** |
| 1100 =06 | 1102=08 | 1200=DD |
| 1101=07 | 1103=09 |  |

**SEARCHING OF SMALLEST DATA FROM AN ARRAY USING SUBROUTINE**

**AIM :**

To search the smallest data from an array and store the result in memory using subroutine. The input data is also to be fetched from memory.

**PROGRAM**

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** | **REMARKS** |
| **1000**  **1004**  **1007**  **1009**  **100C**  **1010**  **1500**  **1502**  **1505**  **1507**  **150A**  **150B**  **150D**  **150F** | **8B 0E 00 11**  **BE 01 11 FE C9 E8 F4 04**  **88 26 00 12**  **F4**  **8A 24**  **3A 64 01**  **72 03**  **8A 64 01**  **46**  **FE C9 75 F3**  **C3** | **MOV CL,[1100] MOV SI,1101 DEC CL**  **CALL 1500**  **MOV [1200],AH HLT**  **MOV AH,[SI] CMP AH,[SI+1] JC 150A**  **MOV AH,[SI+1] INC SI**  **DEC CL JNZ 1502**  **RET** | **Store count in 1100**  **Pointed first element of array**  **Call subroutine in the location 1500**  **Result in location 1200**  **Compare first & second elements If AH contains the smallest then jump to 150A**  **Checking for the end of the array return to location 100C** |

|  |  |  |  |
| --- | --- | --- | --- |
| **RESULT**  **(I) INPUT** | **OUTPUT** | **(II) INPUT** | **OUTPUT** |
| [1100]= 07 | [1200]=59 | [1100]=05 | [1200]=01 |
| [1101]=AB [1102]=DE  [1103]=EC |  | [1101]=01  [1102]=09  [1103]=07 |  |
| [1104]=59 [1105]=FB  [1106]=AD |  | [1104]=06  [1105]=05 |  |
| [1107]=69 |  |  |  |

# Exercises/Experiments using MASM (PC Required)

The Microsoft macro assembler is an x86 high level assembler for DOS and Microsoft windows. It supports wide varieties of macro facilities and structured programming idioms including high level functions for looping and procedures A program called assembler used to convert the mnemonics of instructions along with the data into the equivalent object code modules, these object code may further converted into executable code using linked and loader programs. This type of program is called as ASSEMBLY LANGUAGE PROGRAMMING. The assembler converts and Assembly language source file to machine code the binary equivalent of the assembly language program. In this respect, the assembler reads an ASCII source file from the disk and program as output. An assembler like Microsoft Macro Assembler (MASM) provides a large number of features for assembly language programmers. Microsoft MASM version 6.11 contains updated software capable of processing printing instructions. Machine codes and instruction cycle counts are generated by MASM for all instructions on each processor beginning with 8086.

# REGISTER TYPE AND USE

General Use Registers (described above):

**(E)AX:** Increment Register (for loops, etc.) **(E)BX:** Base Register (addresses and offsets) **(E)CX:** Free Register (any value)

**(E)DX:** Free Register (any value)

# POINTERS USED DURING CODE EXCECUTION AND JUMPS

**(E)IP:** Instruction Pointer (points to the next instruction as a relative offset of the Code Segment)

**(E)BP:** Base Pointer (shadows (E)SP and can be set manually)

**(E)SP:** Stack Pointer (points to the ‘end’ of the stack)

# SEGMENTS:

**CS:** code segment (holds the base pointer for the program)

**SS:** stack segment (points to the top item in the stack)

**DS:** data segment (points to the data elements)

**ES:** extra segment (can be added to any of the first three)

**FS:** extra segment (can be added to any of the first three)

**GS:** extra segment (can be added to any of the first three)

# INT 21H

The DOS interrupt 21h provides the user facility to interrupt with input and output devices directly but conditioning has to be done on the data while translating through the peripheral devices. It performs several functions related to input output devices. Each function has a function call to perform a specific function which depends on the code that written in the AH register.

# PROCEDURE

A procedure is a collection of instructions to which we can direct the flow of our program, and once the execution of these instructions is over control is given back to the next line to process of the code which called on the procedure. At the time of invoking a procedure the address of the next instruction of the program is kept on the stack so that, once the flow of the program has been transferred and the procedure is done, one can return to the next line of the original program, the one which called the procedure.

# MACRO

A macro is a group of repetitive instructions in a program which are codified only once and can be used as many times as necessary. A macro is like a procedure that inserts a block of statements at various points in your program during assembly. The main difference between a macro and a procedure is that in the macro the passage of parameters is possible and in the procedure it is not. Other difference is that, when a procedure invoked, then the control goes to the sub program and after completing it return back to main program. But when a macro is called, then the sub-program part is inserted in this position. So it takes more memory space than procedure. But, less time consuming.

# STUDY OF ASSEMBLER AND DEBUGGING COMMANDS.

Masm programming can done in four steps

1. Program writing using a notepad and save with .ASM extension
2. Assembling using Masm command and creating the object file
3. Linking using the link command and creating the executable file
4. Executing ie running the exe file

# MASM ASSEMBLER COMMANDS

### C :/>cd foldername C:/foldername>edit filename.asm

After this command executed in command prompt an editor window will open. Program should be typed in this window and saved. The program structure is given below.

### Structure of Program:

.model tiny/small/medium/large

.Stack <some number>

.data

; Initialize data

; which is used in program.

.code

; Program logic goes here.

;

End

### To run the program, the following steps have to be followed: C:/foldername>masm filename.asm

After this command is executed in command prompt if there are no errors in

program regarding to syntax the assembler will generates an object module as discuss above.

### C:/foldername>link filename.obj

After verifying the program for correct syntax and the generated object files should be linked together. For this the above link command should be executed and it will give an EXE file if the model directive is small as discuss above.

The following is a list of MASM reserved words:

|  |  |
| --- | --- |
| ASSUME | assume definition |
| CODE | begin code segment |
| DATA | begin data segment DB define byte |
| DD | define double word |
| DQ | define quad word |
| DS | define storage |
| DUP | duplicate |
| DW | define word |
| ELSE | else statement |
| END | end program |
| ENDM | end macro |
| ENDIF | end if statement |
| ENDP | end procedure |
| ENDS | end segment |
| EQU | equate |
| IF | if statement |
| FAR | far reference |
| MACRO | define macro |
| .MODEL | model type |
| NEAR | near reference |
| OFFSET | offset |
| ORQ | origin |
| PARA | paragraph |
| PROC | define procedure |
| .EXIT | generate exit code |

PUBLIC public reference

SEG l ocate segment

SEGMENT define segment

PTR pointer

### USING DEBUG TO EXECUTE THE 80x86 PROGRAM:

DEBUG is a utility program that allows a user to load an 80x 86 programs into memory and execute it step by step. DEBUG displays the contents of all processor registers after each instruction execute, allowing the user to determine if the code is performing the desired task. DEBUG contains commands that can display and modify memory, assemble instructions, disassemble code already placed into memory, trace single or multiple instructions, load registers with data and do much more.

DEBUG loads into memory like any other program, in the first available slot. The memory space used by DEBUG for the user program begins after the end of Debug’s code.

To execute the program file PROG.EXE use this command C> debug filename.exe

* (Screen shows only dash)
* t

‘t’ for trace the program execution by single stepping starting from the address

SEG.OFFSET. ‘q’ for Quit from Debug & return to DOS

DEBUG uses a minus sign as its command prompt, so should see a “-“ appear on display. A list of some commands available with DEBUG is :

? - To see command list

A - Assemble instructions in assembler and convert them to machine code directly.

1. - Compare two memory blocks
2. - Dumb the control of memory location
3. - Enter bytes of data in memory.
4. - go (complete execution)
5. - Hexarithmetic for adding and subtracting hexadecimal numbers I - Input to read a byte of data from any of the I/O ports in the PC
6. - load a program or absolute sectors from the disk into memory
7. - Move moves data
8. - Name, specifying the name of a file which DEBUG is going to Load or Write
9. - Output to send a single byte of data to a port
10. - Proceed To execute the next instruction
11. - Quit the program
12. - To see the present content of all registers
13. - Search occurrence of a specific byte or series of bytes within a segment.
14. - trace (step by step execution)
15. - Unassemble the program

W - Write

# PROGRAM FOR 16 BIT ADDITION

**ALGORITHM:**

Step 1 : Initialize the data segment.

Step 2 : Get the first number in AX register. Step 3 : Get the second number in BX register. Step 4 : Add the two numbers.

Step 5 : Display the result. Step 6 : Stop **PROGRAM**

.MODEL SMALL

.DATA

A DW 1234H

B DW 0100H

.CODE

MOV AX, @DATA MOV DS, AX

MOV AX, A

MOV BX, B

ADD AX, BX MOV CH, 04H MOV CL, 04H

MOV BX, AX L2: ROL BX, CL

MOV DL, BL

AND DL, 0FH

CMP DL, 09

JBE L4

ADD DL, 07 L4: ADD DL, 30H

MOV AH, 02

INT 21H

DEC CH

JNZ L2

MOV AH, 4CH INT 21H

END **OUTPUT** 1334

# PROGRAM FOR 16 BIT SUBTRACTION

**ALGORITHM:**

Step 1 : Initialize the data segment.

Step 2 : Get the first number in AX register. Step 3 : Get the second number in BX register. Step 4 : Subtract the two numbers.

Step 5 : Display the result. Step 6 : Stop **PROGRAM**

.MODEL SMALL

.DATA

A DW 1234H

B DW 0100H

.CODE

MOV AX, @DATA MOV DS, AX

MOV AX, A

MOV BX, B

SUB AX, BX MOV CH, 04H MOV CL, 04H

MOV BX, AX L2: ROL BX, CL

MOV DL, BL

AND DL, 0FH

CMP DL, 09

JBE L4

ADD DL, 07 L4: ADD DL, 30H

MOV AH, 02

INT 21H

DEC CH

JNZ L2

MOV AH, 4CH INT 21H

END **OUTPUT** 1134

# PROGRAM FOR 32 BIT ADDITION

**ALGORITHM**

Step 1 **:** Initialize the data segment.

Step 2 **:** Load the LSB of first number into AX register. Step 3 **:** Load the MSB of first number into BX register.

Step 4 **:** Load the LSB of the second number into CX register. Step 5 **:** Load the MSB of the second number into DX register. Step 6 **:** Add the LSBs of two number.

Step 7 **:** Add the MSBs of two numbers along with carry. Step 8 **:** Display the result.

Step 9 **:** Stop.

# PROGRAM

.MODEL SMALL

.DATA

NUM1 DD 12345678H NUM2 DD 11111112H RES DD ?

.CODE

START: MOV AX,@DATA MOV DS,AX

MOV AX,WORD PTR NUM1 MOV BX,WORD PTR NUM1+2 MOV CX,WORD PTR NUM2 MOV DX,WORD PTR NUM2+2 ADD AX,CX

ADD BX,DX

MOV WORD PTR RES,AX MOV WORD PTR RES+2,BX MOV DH,02

LABEL3:MOV CH,04 LABEL2:MOV CL,04 ROL BX,CL

MOV DL,BL AND DL,0FH CMP DL,09 JBE LABEL1 ADD DL,07H

LABEL1:ADD DL,30H MOV AH,02H

INT 21H DEC CH

JNZ LABEL2

MOV BX,WORD PTR RES DEC DH

JNZ LABEL3 MOV AH,4CH INT 21H

END START

## OUTPUT

2345678A

# PROGRAM FOR 32 BIT SUBTRACTION

**ALGORITHM**

Step 1 : Initialize the data segment.

Step 2 : Load the LSB of first number into AX register. Step 3 : Load the MSB of first number into BX register.

Step 4 : Load the LSB of the second number into CX register. Step 5 : Load the MSB of the second number into DX register. Step 6 : Subtract the two LSBs.

Step 7 : Subtract the two MSBs along with borrow. Step 8 : Display the result.

Step 9 : Stop.

# PROGRAM

.MODEL SMALL

.DATA

NUM1 DD 12345678H NUM2 DD 11110000H RES DD ?

.CODE

START: MOV AX,@DATA MOV DS,AX

MOV AX,WORD PTR NUM1 MOV BX,WORD PTR NUM1+2 MOV CX,WORD PTR NUM2 MOV DX,WORD PTR NUM2+2 SUB AX,CX

SUB BX,DX

MOV WORD PTR RES,AX MOV WORD PTR RES+2,BX MOV DH,02

LABEL3:MOV CH,04 LABEL2:MOV CL,04 ROL BX,CL

MOV DL,BL AND DL,0FH CMP DL,09 JBE LABEL1 ADD DL,07H

LABEL1:ADD DL,30H MOV AH,02H

INT 21H DEC CH

JNZ LABEL2

MOV BX,WORD PTR RES DEC DH

JNZ LABEL3 MOV AH,4CH INT 21H

END START

## OUTPUT

01235678

# PROGRAM TO READ AND PRINT A STRING

**ALGORITHM**

Step 1 : Start

Step 2 : Define the model in which the program is written Step 3 : In the data segment define the string to be printed Step 4 : Move the contents in data to AX

Step 5 : Move the contents in AX to DS register

Step 6 : Move the address of MSG1 to DX and call the interrupt

Step 7 : Read the string by moving array offset to DX and call the interrupt by using 3FH

Step 8 : Move the address of MSG2 to DX and call the interrupt

Step 9 : Print the entered string by moving the address of the array into DX and call interrupt

Step 10 : Stop

# PROGRAM

.MODEL SMALL

.STACK

.DATA

MSG1 DB 10,13,"ENTER THE STRING $" MSG2 DB 10,13,"ENTERED STRING IS $" STR1 DB 20 DUP('$')

.CODE START:

MOV AX,@DATA MOV DS,AX

LEA DX,MSG1

MOV AH,09H INT 21H MOV AH,3FH

MOV DX,OFFSET STR1 INT 21H

LEA DX,MSG2 MOV AH,09H INT 21H

LEA DX,STR1 MOV AH,09H INT 21H

END

## OUTPUT

ENTER THE STRING COMPUTER ENTERED STRING IS COMPUTER

# PROGRAM TO REVERSE A STRING

**ALGORITHM**

Step 1 : Start

Step 2 : Define macro to display string which is stored in AX register Step 3 : Declare the string array and message variable in the data segment

Step 4 : Call the macro by passing arguments which are already declared in the data segment

Step 5 : Read the string to register using interrupt

Step 6 : Print the first location of the string to the register

Step 7 : Compare the first character with the ‘$’ increment the pointer till the end of the string reached in the ‘$’

Step 8 : If a ‘$’ encounter, end of the string is reached ten decrement the pointer and display each character in reverse order till the pointer index zero

Step 9 : Stop

# PROGRAM

.MODEL SMALL

.STACK

.DATA

MSG1 DB 0AH,0DH,"ENTER THE STRING: $" MSG2 DB 0AH,0DH,"REVERSED STRING IS: $" STR1 DB 100 DUP('$')

.CODE START:

MOV AX,@DATA MOV DS,AX

LEA DX,MSG1 MOV AH,09H INT 21H

LEA DX,STR1 MOV AH,3FH INT 21H

LEA DX,MSG2 MOV AH,09H INT 21H

MOV SI,00H

L2: CMP STR1[SI],"$" JE L1

INC SI JMP L2 L1: DEC SI

MOV DL,STR1[SI] MOV AH,02H

INT 21H JNZ L1 END

## OUTPUT

ENTER THE STRING: COMPUTER REVERSED STRING IS: RETUPMOC

# PROGRAM TO CHECK WHETHER A GIVEN STRING IS PALINDROME OR NOT

**ALGORITHM**

Step 1 : Start

Step 2 : Define macro to display a message

Step 3 : Declare string array and message in data segment Step 4 : Call macro to display message to read string

Step 5 : Read the string using macro

Step 6 : Traverse the string to the last character by comparing each character with ‘$’

Step 7 : Point another pointer to the first character and compare that with the last character by the help of the pointer which is pointing to the last

Step 8 : If equal then increment the second pointer and decrement the first pointer to get the next character and compare

Step 9 : Repeat the step 9 until this first pointer reaches to the first character and index to the zero then the string is palindrome

Step 10 : If any mismatch found goto step 12

Step 11 : Print message as the string is not palindrome Step 12 : Stop

# PROGRAM

.MODEL SMALL

.STACK

.DATA

D1 DB 100 DUP("$")

D2 DB 0AH,0DH,"ENTER THE STRING: $"

D3 DB 0AH,0DH,"STRING IS PALINDROME $"

D4 DB 0AH,0DH,"NOT PALINDROME $"

.CODE START:

MOV AX,@DATA MOV DS,AX

LEA DX,D2 MOV AH,09H INT 21H

LEA DX,D1 MOV AH,3FH INT 21H

SUB AX,02H MOV SI,AX DEC SI MOV CX,AX MOV DI,00H L1:

MOV AH,D1[DI] CMP AH,D1[SI] JNE L2

INC DI DEC SI LOOP L1 LEA DX,D3

MOV AH,09H INT 21H

JMP L3 L2:

LEA DX,D4 MOV AH,09H INT 21H

L3:

END START

## OUTPUT

ENTER THE STRING: AMMA STRING IS PALINDROME ENTER THE STRING: AMMU NOT PALINDROME

# PROGRAM TO CONCATENATE TWO STRINGS

**ALGORITHM**

Step 1 : Start

Step 2 : Enter the model in which program is writing Step 3 : In data segment define the data to print

Step 4 : Define two array variable of size 100 Step 5 : In code segment move data to AX register

Step 6 : Move all contents of AX into data segment Step 7 : Enter the string by calling interrupt

Step 8 : Enter the second string and write it into the second array variable Step 9 : Move 00H to both SI and DI

Step 10 : Compare the 1st element of the 1st string with ‘$’ if they are not equal, increment SI. This is continued until ‘$’ if it is reached then we have to compare the 1st element of 2nd string with ‘$’

Step 11 : If they are not equal, put the element to the BL register, then it is moved to STR1.When the ‘$’ is reached in 2nd string we print the concatenated string in STR2

Step 12 : Exit from the program Step 13 : Stop

**PROGRAM** READ MACRO A MOV AH,3FH LEA DX,A

INT 21H ENDM

DISP MACRO B

MOV AH,09H LEA DX,B INT 21H ENDM

.MODEL SMALL

.STACK

.DATA

MSG1 DB 0AH,0DH,"ENTER THE FIRST STRING : $" MSG2 DB 0AH,0DH,"ENTER THE SECOND STRING : $" MSG3 DB 0AH,0DH,"NEW STRING : $"

STR1 DB 100 DUP("$") STR2 DB 100 DUP("$")

.CODE START:

MOV AX,@DATA MOV DS,AX MOV SI,00H MOV DI,00H DISP MSG1 READ STR1

SUB AX,02H MOV SI,AX DISP MSG2 READ STR2 L1:

CMP STR2[DI],'$' JNE L2

DISP MSG3

DISP STR1 JMP L3 L2:

MOV AH,STR2[DI] MOV STR1[SI],AH INC SI

INC DI LOOP L1 L3:

END

## OUTPUT

ENTER THE FIRST STRING : KIT ENTER THE SECOND STRING :KAT NEW STRING : KITKAT

# PROGRAM TO REPLACE A CHARACTER

**ALGORITHM**

Step 1 : Start

Step 2 : Define the string variable message in DS Step 3 : Read string from keyboard

Step 4 : Read character to be replaced and it be replaced with this character Step 5 : Initialize source index to zero

Step 6 : Increment the value of SI until the string reached its ends and checks each element with the character be replaced

Step 7 : If they are equal, then replace it with the other element using registers Step 8 : Print the strings after replacement

Step 9 : Stop

**PROGRAM** READ MACRO X MOV DX,OFFSET X MOV AH,3FH

INT 21H ENDM

DISPLAY MACRO X MOV DX,OFFSET X MOV AH,09H

INT 21H ENDM

.MODEL SMALL

.STACK

.DATA

MSG1 DB 0AH,0DH,"ENTER THE STRING :$"

MSG2 DB 0AH,0DH,"ENTER THE CHARACTER TO BE REPLACED :$"

MSG3 DB 0AH,0DH,"ENTER THE CHARACTER TO BE REPLACED WITH : $" MSG4 DB 0AH,0DH," NEW STRING IS : $"

STR1 DB 100 DUP("$")

.CODE START:

MOV AX,@DATA MOV DS,AX DISPLAY MSG1 READ STR1 DISPLAY MSG2 MOV AH,01H

INT 21H MOV BL,AL

DISPLAY MSG3 MOV AH,01H INT 21H

MOV CL,AL MOV SI,00H L0:

CMP STR1[SI],'$' JE L3

L1:

CMP STR1[SI],BL JE L2

INC SI JMP L0 L2:

MOV STR1[SI],CL INC SI

JMP L0 L3:

DISPLAY MSG4 DISPLAY STR1 END

## OUTPUT

ENTER THE STRING : AMMU

ENTER THE CHARACTER TO BE REPLACED : M ENTER THE CHARACTER TO BE REPLACED WITH : P NEW STRING IS : APPU

# PROGRAM TO REPLACE ALL VOWELS

**ALGORITHM**

Step 1 : Start

Step 2 : Define the data segment with statement to enter the string and the string of vowels ‘AEIOUaeiou’

Step 3 : Start the code segment by starting the address of data part to accumulator register

Step 4 : Let the message to the data register Step 5 : Read the string using 3FH

Step 6 : Initialize SI,DI as 00H

Step 7 : Compare each element with each element of the strings ‘AEIOUaeiou’ Step 8 : If not equal, then print the character

Step 9 : Repeats these steps till the ends of string reached Step 10 : Stop

**PROGRAM** READ MACRO S LEA DX,S

MOV AH,3FH INT 21H ENDM

DISPLAY MACRO MSG LEA DX,MSG

MOV AH,09H INT 21H ENDM

.MODEL SMALL

.STACK

.DATA

MSG1 DB 0AH,0DH,"ENTER A STRING:$"

MSG2 DB 0AH,0DH,"THE STRING WITHOUT VOWEL IS:$" MSG3 DB 0AH,0DH,"AEIOUaeiou$"

STR1 DB 100 DUP("$")

.CODE START:

MOV AX,@DATA MOV DS,AX DISPLAY MSG1 READ STR1 DISPLAY MSG2 MOV DI,00H MOV SI,00H

L1:

MOV AH,STR1[SI] CMP MSG3[DI],AH JE L2

INC DI

CMP MSG3[DI],'$' JE V

LOOP L1 V:

MOV AH,02H MOV DL,STR1[SI] INT 21H

INC SI

MOV DI,00H CMP STR1[SI],'$' JE STOP

JMP L1

L2:

INC SI

MOV DI,00H CMP STR1[SI],'$' JE STOP

JMP L1 STOP:

END START

## OUTPUT

ENTER A STRING:ELEPHANT

THE STRING WITHOUT VOWEL IS:LPHNT

# PROGRAM TO SEARCH A 16 BIT NUMBER

**ALGORITHM**

Step 1 : Start

Step 2 : Define macro to display message Step 3 : Initialize the data segment

Step 4 : Initialize the length

Step 5 : Compare each numbers in the array to be searched Step 6 : If the number is encountered, print found

Step 7 : If the number is not in the array, print not found Step 8 : Stop

# PROGRAM

.MODEL SMALL

.DATA

ARR DW 0111H,0112H,0113H,0114H,0115H LEN DW ($-ARR)/2

KEY EQU 0116H MSG1 DB "FOUND$"

MSG2 DB "NOT FOUND$"

.CODE

MOV AX,@DATA MOV DS,AX MOV BX,00

MOV DX,LEN MOV CX,KEY

AGAIN: CMP BX,DX JA NOTFND

MOV AX,BX ADD AX,DX

SHR AX,1 MOV SI,AX ADD SI,SI

CMP CX,ARR[SI] JAE BIG

DEC AX MOV DX,AX JMP AGAIN

BIG: JE FOUND INC AX

MOV BX,AX JMP AGAIN

FOUND: LEA DX,MSG1 JMP DISPL

NOTFND: LEA DX,MSG2 DISPL : MOV AH,09H INT 21H

INT 3 END

# OUTPUT

NOT FOUND

# STEPPER MOTOR INTERFACING

## AIM:-

To interface a stepper motor with 8086 microprocessor kit.

## THEORY:-

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotary motion occurs in a stepwise manner from one equilibrium position to the next.

Stepper motor control is a very popular application of microprocessor in control area. The are widely used in ( simple position control systems in the open and closed loop mode ) a variety of applications such as computer peripherals and in the areas of process control machine tools, medicine, numerically controlled machines and robotics.

## WAVE SCHEME

The stepper motor windings A1, B1, A2, B2 can be cyclically excited with a DC current to run the motor in the clockwise direction. By reversing the phase sequences as A1, B2, A2, B1. we can obtain anticlockwise stepping.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Anti clock Wise** |  |  | **Clock Wise** |  |
| **STEP1 A1** | **ON** | **STEP1** | **A1** | **ON** |
| **STEP2 B2** | **ON** | **STEP2** | **B1** | **ON** |
| **STEP3 A2** | **ON** | **STEP3** | **A2** | **ON** |
| **STEP4 B1** | **ON** | **STEP4** | **B2** | **ON** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ANTICLOCK WISE** | | | | | **CLOCK WISE** | | | | |
| **STEP** | **A1** | **A2** | **A3** | **A4** | **STEP** | **A1** | **A2** | **A3** | **A4** |
| **1** | **1** | **0** | **0** | **0** | **1** | **1** | **0** | **0** | **0** |
| **2** | **0** | **0** | **0** | **1** | **2** | **0** | **0** | **1** | **0** |
| **3** | **0** | **1** | **0** | **0** | **3** | **0** | **1** | **0** | **0** |
| **4** | **0** | **0** | **1** | **0** | **4** | **0** | **0** | **0** | **1** |

## 2 – PHASE SCHEME

In this scheme, any two adjacent stator windings are energized. There are two magnetic fields active in the quadrature and none of the rotor pole faces can be in direct alignment with the stator poles. A partial but symmetric alignment of the rotor poles is possible. The step angle is 30 degree and the rotor is offset by 15 degree with respect to the wave scheme.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Anti clock Wise** |  |  | **Clock Wise** |  |
| **STEP1 A1 ON, A4** | **ON** | **STEP1** | **A1 ON, A3** | **ON** |
| **STEP2 A2 ON, A4** | **ON** | **STEP2** | **A2 ON, A3** | **ON** |
| **STEP3 A2 ON, A3** | **ON** | **STEP3** | **A2 ON, A4** | **ON** |
| **STEP4 A1 ON, A3** | **ON** | **STEP4** | **A1 ON, A4** | **ON** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ANTICLOCK WISE** | | | | | **CLOCK WISE** | | | | |
| **STEP** | **A1** | **A2** | **A3** | **A4** | **STEP** | **A1** | **A2** | **A3** | **A4** |
| **1** | **1** | **0** | **0** | **1** | **1** | **1** | **0** | **1** | **0** |
| **2** | **0** | **1** | **0** | **1** | **2** | **0** | **1** | **1** | **0** |
| **3** | **0** | **1** | **1** | **0** | **3** | **0** | **1** | **0** | **1** |
| **4** | **1** | **0** | **1** | **0** | **4** | **1** | **0** | **0** | **1** |

## HALF STEPPING

This wave scheme and 2 phase scheme give step size of 30 degree. There is an offset of 15 degree in between the two states. By interleaving the 2 schemes the step size can be reduced to 15 degree thereby improving the accuracy of the stepper motor.

The switching sequence is:

1. A1 (ON)
2. A1 and B1 (ON)
3. B1 (ON)
4. B1 and A2 (ON)
5. A2 (ON)
6. A2 and B2 (ON)
7. B2 (ON)
8. B2 and A1 (ON)
9. A1 (ON), etc...

**GROUP B**

**MODE SET FLAG**

**ACTIVE (1)**

**GROUP A**

**PORT C (LOW) 1-I/P**

**0-O/P**

**PORT C (UPPER) 1-I/P**

**0-O/P**

**PORT B 1-I/P**

**0-O/P**

**PORT A 1-I/P**

**0-O/P**

**MODE SELECT**

1. **– MODE 0**
2. **– MODE 1**

**MODE SELECT 00 – MODE 0**

**01 – MODE 1**

**1X – MODE 2**

Eight steps are required to move the shaft by 120 degree and 24 steps for one complete revolution. By reversing the switching sequence, we can reverse the direction of rotation.

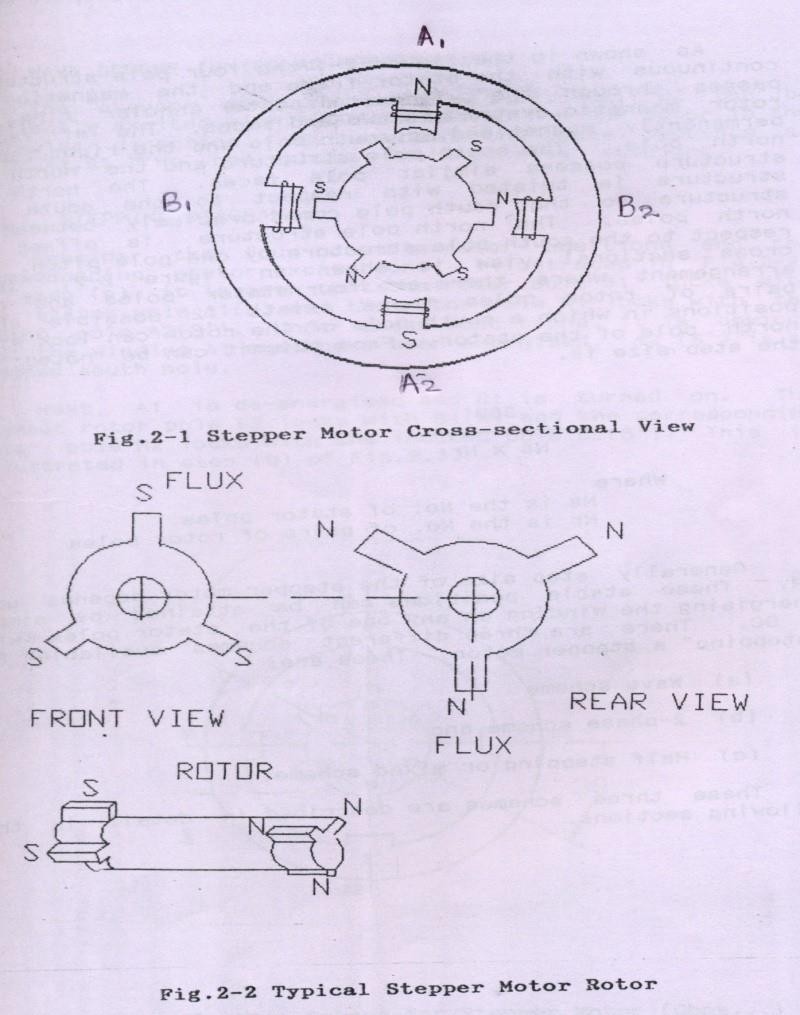
## WAVE SWITCHING SCHEME

Here only one pole is energized by the microprocessor. When A2 alone is energized, the nearest rotor pole locks with it. When B2 alone is energized the nearest rotor pole is locked with it.

Totally 12 such steps cause a displacement of 360 degree. Thus the step angle is 30 degree.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STEP** | **A1** | **A2** | **B1** | **B2** |
| **1** | **1** | **0** | **0** | **0** |
| **2** | **1** | **0** | **1** | **0** |
| **3** | **0** | **0** | **1** | **0** |
| **4** | **0** | **1** | **1** | **0** |
| **5** | **0** | **1** | **0** | **0** |
| **6** | **0** | **1** | **0** | **1** |
| **7** | **0** | **0** | **0** | **1** |
| **8** | **1** | **0** | **0** | **1** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |



**TO RUN THE STEPPER MOTOR IN FORWARD DIRECTION**

**PROGRAM**

|  |  |  |
| --- | --- | --- |
| **MEMORY ADDRESS** | **OBJECT CODES** | **MNEMONICS** |
| **1000**  **1003**  **1007**  **100A**  **100C**  **100E**  **1010**  **1012**  **1016**  **1017**  **1019**  **101A**  **101C**  **101F** | **C6 C3 45**  **C7 C7 20 10**  **C6 C1 04**  **8A 05 E6 C0 FE CB 74 0D**  **C7 C2 10 10 4A**  **75 FD**  **47**  **E2 EE**  **E9 E4 FF F4** | **MOV BL,45 MOV DI,1020**  **MOV CL,04 MOV AL,[DI] OUT C0,AL DEC BL**  **JZ 101F**  **MOV DX,1010 DEC BL**  **JNZ 1016 INC DI LOOP 100A JMP 1003**  **HLT** |

**RESULT**

**INPUT**

1020 =09 1022=06

1021=05 1023=0A

The stepper motor was interfaced with 8086 microprocessor and related programming experiments have been carried out.

# INTERFACING WITH 8255

**AIM**

Program to sense switch positions sw0‐sw7. The sensed pattern to be displayed on port, to which 8 LED’s are connected. Use 8086 Processor.

# HARDWARE REQUIREMENTS

The 8086 Microprocessor kit, 8255 Programmable Peripheral Interface add on card, Interface cable and Power Supply.

# THEORY

The 8255 is a widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O.

Features

* Three 8‐bit IO ports PA, PB, PC
* PA can be set for Modes 0, 1, 2. PB for 0,1 and PC for mode 0 and for BSR.
* Modes 1
* and 2 are interrupt driven.
* PC has two 4‐bit ports: PC upper (PCU) and PC lower (PCL), each can be set independently for Input or Output. Each PC bit can be set/reset individually in BSR mode.
* PA and PCU are Group A (GA) and PB and PCL are Group B (GB)
* Address/data bus must be externally demultiplexed.

8255 can be operated in two modes

**BSR (Bit Set Reset ) mode or**

1. I/O mode
2. BSR mode

Bit set/reset, applicable to PC only. One bit is S/R at a time.

Command word

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 0 (0=BSR) | X | X | X | B2 | B1 | B0 | S/R (1=S,0=R) |

Bit select: (Taking Don't care's as 0)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B2 | B1 | B0 | PC bit | Control word (Set) | Control word (reset) |
| 0 | 0 | 0 | 0 | 0000 0001 = 01h | 0000 0000 = 00h |
| 0 | 0 | 1 | 1 | 0000 0011 = 03h | 0000 0010 = 02h |
| 0 | 1 | 0 | 2 | 0000 0101 = 05h | 0000 0100 = 04h |
| 0 | 1 | 1 | 3 | 0000 0111 = 07h | 0000 0110 = 06h |
| 1 | 0 | 0 | 4 | 0000 1001 = 09h | 0000 1000 = 08h |
| 1 | 0 | 1 | 5 | 0000 1011 = 0Bh | 0000 1010 = 0Ah |
| 1 | 1 | 0 | 6 | 0000 1101 = 0Dh | 0000 1100 = 0Ch |
| 1 | 1 | 1 | 7 | 0000 1111 = 0Fh | 0000 1110 = 0Eh |

### I/O mode

The I/O mode is further divided into three modes:

Mode 0 : all ports function as simple I/O ports

Mode 1 : Hand shake mode whereby Ports A and/or B use bits from port C as handshake signals

Mode 2. : Port A can be set up for bidirectional data transfer using handshaking signals from Port C,

and Port B can be set up either in Mode 0 or Mode 1.

### Command word

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| 1 (1=I/O) | GA mode select | | PA | PCU | GB mode select | PB | PCL |

* + D6, D5: GA mode select:
    - 00 = mode0
    - 01 = mode1
    - 1X = mode2
  + D4(PA), D3(PCU): 1=input 0=output
  + D2: GB mode select: 0=mode0, 1=mode1
  + D1(PB), D0(PCL): 1=input 0=output Port Address

|  |  |
| --- | --- |
| Register | Address |
| Control word register | C6 |
| Port A | C0 |
| Port B | C2 |
| Port C | C4 |

# PROGRAM

* 1. Initialize port A as input port and port B as output port in mode 0 , to input the data at port A as set by the SPDT switches and to output the same data to port B to glow the LED accordingly.
  2. Initialize port A as input port and port c as output port in mode 0 , to input the data at port A as set by the SPDT switches and to output the same data to port B to glow the LED accordingly.
  3. Initialize port C as input port and port B as output port in mode 0 , to input the data at port A as set by the SPDT switches and to output the same data to port B to glow the LED accordingly.

### Mode 0, Port A input port, Port B output port

|  |  |
| --- | --- |
| **LABEL** | **MNEMONICS** |
|  | MOV AL,90H OUT C6,AL IN AL,C0 OUT C2,AL  MOV [1200],AL  HLT |

**Mode 0, Port A input port, Port C output port**

|  |  |
| --- | --- |
| **LABEL** | **MNEMONICS** |
|  | MOV AL,90H OUT C6,AL IN AL,C0 OUT C4,AL  MOV [1200],AL HLT |

**Mode 0, Port C input port, Port B output port**

|  |  |
| --- | --- |
| **LABEL** | **MNEMONICS** |
|  | MOV AL,90H OUT C6,AL IN AL,C4 OUT C2,AL  MOV [1200],AL HLT |

# RESULT

The 8255 PPI was interfaced with 8086 microprocessor and related programming experiments have been carried out.

# MULTI BYTE ADDITION

**AIM**

To add two 64 bit numbers residing in memory location 4200 and 4210 and the result is stored in 4500.

**PROGRAM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MEMORY** | **OBJECT** | **LABEL** | **MNEMONICS** | **COMMENTS** |
| **ADDRESS** | **CODES** |
| **4100** | **C3** | **L1** | **CLR C** | **Clear carry flag** |
| **4101** | **7D 00** | **MOV R5,#00** | **Register for carry** |
| **4103** | **7C 08** | **MOV R4,#08** | **Count** |
| **4105** | **90 42 00** | **MOV DPTR,#4200** | **Pointer for first no.** |
| **4108** | **A8 82** | **MOV R0,DPL** |  |
| **410A** | **A9 83** | **MOV R1,DPH** |  |
| **410C** | **90 42 10** | **MOV DPTR,#4210** | **Pointer for second no.** |
| **410F** | **AA 82** | **MOV R2,DPL** |  |
| **4111** | **AB 83** | **MOV R3,DPH** |  |
| **4113** | **90 45 00** | **MOV DPTR,#4500** | **Pointer for result** |
| **4116** | **C0 82** | **PUSH DPL** | **Address for result stored** |
| **4118** | **CO 83** | **PUSH DPH** | **in stack.** |
| **411A** | **88 82** | **MOV DPL,R0** |  |
| **411C** | **89 83** | **MOV DPH,R1** |  |
| **411E** | **E0** | **MOVX A,@DPTR** |  |
| **411F** | **F5 F0** | **MOV B,A** | **Move first no. from** |
| **4121** | **A3** | **INC DPTR** | **4200 to A** |
| **4122** | **A8 82** | **MOV R0,DPL** | **Move A to B** |
| **4124** | **A9 83** | **MOV R1,DPH** | **Increment DPTR** |
| **4126** | **8A 82** | **MOV DPL,R2** |  |
| **4128** | **8B 83** | **MOV DPH,R3** |  |
| **412A** | **E0** | **MOVX A,@DPTR** |  |
| **412B** | **35 F0** | **ADDC A,B** |  |
| **412D** | **A3** | **INC DPTR** | **Move second no. from** |
| **412E** | **AA 82** | **MOV R2,DPL** | **4210 to A** |
| **4130** | **AB 83** | **MOV R3,DPH** | **Add 1st and 2nd nos. with** |
| **4132** | **D0 83** | **POP DPH** | **carry** |
| **4134** | **D0 82** | **POP DPL** | **Increment DPTR** |
| **4136** | **F0** | **MOVX @DPTR,A** |  |
| **4137** | **A3** | **INC DPTR** |  |
| **4138** | **DC DC** | **DJNZ R4, L1** | **Retrieve the data from** |
| **413A** | **50 01** | **JNC L2** | **stack** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **413C** | **0D** |  | **INC R5** | **Store the result at 4500 Increment DPTR Decrement and jump to 4116 if R4 not zero**  **If no carry Jump to 413D**  **Increment carry Move carry to A Store carry**  **End of the program** |
| **413D** | **ED** | **L2** | **MOV A,R5** |
| **413E** | **F0** |  | **MOVX@DPTR,A** |
| **413F** | **80 FE** | **L3** | **SJMP L3** |

|  |
| --- |
| **RESULT:-** |
| **DATA DATA RESULT** |
| 4200- 3C 4210-15 4500 - 51 |
| 4201- 42 4211- 6A 4501- AC |
| 4202- 9A 4212- 84 4502- 1E |
| 4203- 86 4213- 3B 4503- C2 |
| 4204- 35 4214- 92 4504- C7 |
| 4205- 40 4215- 85 4505- C5 |
| 4206- 3C 4216- 15 4506- 51 |
| 4207- FF 4217- 85 4507- 84 |
| 4508- 01 |
| **CONCLUSION:-** |
| Two 64 bit numbers residing in memory location 4200 and 4210 are added, and the result |
| is stored in 4500**.** |

**MULTIBYTE SUBTRACTION**

**AIM**

To subtract two 64 bit numbers residing in memory location 4200 and 4210 and the resultis stored in 4500.

# PROGRAM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MEMORY** | **OBJECT** | **LABEL** | **MNEMONICS** | **COMMENTS** |
| **ADDRESS** | **CODES** |
| **4100** | **C3** |  | **CLR C** | **Clear carry flag** |
| **4101** | **7C 08** |  | **MOV R4,#08** | **Count** |
| **4103** | **90 42 00** |  | **MOV DPTR,#4200** | **Pointer for 2nd no.** |
| **4106** | **A8 82** |  | **MOV R0,DPL** |  |
| **4108** | **A9 83** |  | **MOV R1,DPH** |  |
| **410A** | **90 42 10** |  | **MOV DPTR,#4210** | **Pointer for 1st no.** |
| **410D** | **AA 82** |  | **MOV R2,DPL** |  |
| **410F** | **AB 83** |  | **MOV R3,DPH** |  |
| **4111** | **90 45 00** |  | **MOV DPTR,#4500** | **Pointer for result** |
| **4114** | **C0 82** | **L1** | **PUSH DPL** | **Address for result stored in** |
| **4116** | **C0 83** |  | **PUSH DPH** | **stack.** |
| **4118** | **88 82** |  | **MOV DPL,R0** |  |
| **411A** | **89 83** |  | **MOV DPH,R1** |  |
| **411C** | **E0** |  | **MOVX A,@DPTR** | **Move first no. from 4200 to A** |
| **411D** | **F5 F0** |  | **MOV B,A** | **Move A to B** |
| **411F** | **A3** |  | **INC DPTR** | **Increment DPTR** |
| **4120** | **A8 82** |  | **MOV R0,DPL** |  |
| **4122** | **A9 83** |  | **MOV R1,DPH** |  |
| **4124** | **8A 82** |  | **MOV DPL,R2** |  |
| **4126** | **8B 83** |  | **MOV DPH,R3** |  |
| **4128** | **E0** |  | **MOVX A,@DPTR** | **Move 2nd no. from 4210 to A** |
| **4129** | **95 F0** |  | **SUBB A,B** | **Subtract 1st and 2nd nos. with** |
|  |  |  |  | **borrow** |
| **412B** | **A3** |  | **INC DPTR** | **Increment DPTR** |
| **412C** | **AA 82** |  | **MOV R2,DPL** |  |
| **412E** | **AB 83** |  | **MOV R3,DPH** |  |
| **4130** | **D0 83** |  | **POP DPH** | **Retrieve the data from stack** |
| **4132** | **D0 82** |  | **POP DPL** |  |
| **4134** | **F0** |  | **MOVX @DPTR,A** | **Store the result at 4500** |
| **4135**  **4136**  **4138** | **A3**  **DC DC**  **80 FE** | **L2** | **INC DPTR DJNZ R4, L1**  **SJMP L2** | **Increment DPTR**  **Decrement and jump to 4114 if R4 not zero**  **End of the program** |

|  |
| --- |
| **RESULT:-** |
| **DATA DATA RESULT** |
| 4210- 3C 4200- 15 4500- 27 |
| 4211- 6A 4201- 42 4501- 28 |
| 4212- 9A 4202- 84 4502- 16 |
| 4213- 86 4203- 3B 4503- 4B |
| 4214- 92 4204- 35 4504- 5D |
| 4215- 85 4205- 40 4505- 45 |
| 4216- 3C 4206- 15 4506- 27 |
| 4217- 6A 4207- 42 4507- 28 |
| **CONCLUSION:-** |
| Two 64 bit numbers residing in memory location 4200 and 4210 are subtracted, and the |
| result is stored in 4500. |

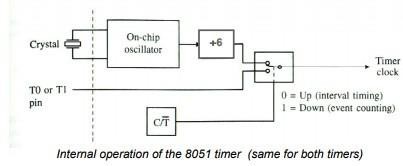
**IMPLEMENTATION OF TIMER PROGRAMMING**

**8051 Timers**

* 2 timers (Timer 0 and Timer 1)
* 16-bit timers (65,535) max
* Flag is set when the timer overflows
* Timers can be based on internal clock (OSC/6) or from externalsource (counter mode).

**Timer Registers**

* TCON Timer Control
* TMOD Timer Mode
* TH0/TL0 Timer 0 16 bit register (byte addressable only)
* TH1/TL1 Timer 1 16 bit register (byte addressable only)



Program to generate 100 µs delay using timer 0 mode 1. Procedure is:

* Initialise TMOD register
* Initialise TL0 and TH0
* Start the Timer
* Monitor TF0 until it is set

Delay: MOV TMOD,#01H ; initialise TMOD MOV TL0,#47H ; initialise TL0 MOV TL0,#FFH ; initialise TH0 SETB TR0 ; start timer

Wait: JNB TF0,Wait ; wait for TF0 CLR TR0 ; stop timer

CLR TF0 ; clear TF0 RET

100 ∗10 *−* 3

*DelayValue*= 6

=184

11.0592 ∗106

*TimerReloadValue*=65535 *–* 184=65351=0 *xFF* 47so Timer 0 is loaded with: TH0 = 0x6F; TL0 = 0xFF;

# RESULT

The TIMER was interfaced with 8051 microcontroller and related to generate 100 µs delay using timer 0 mode 1 have been carried out.