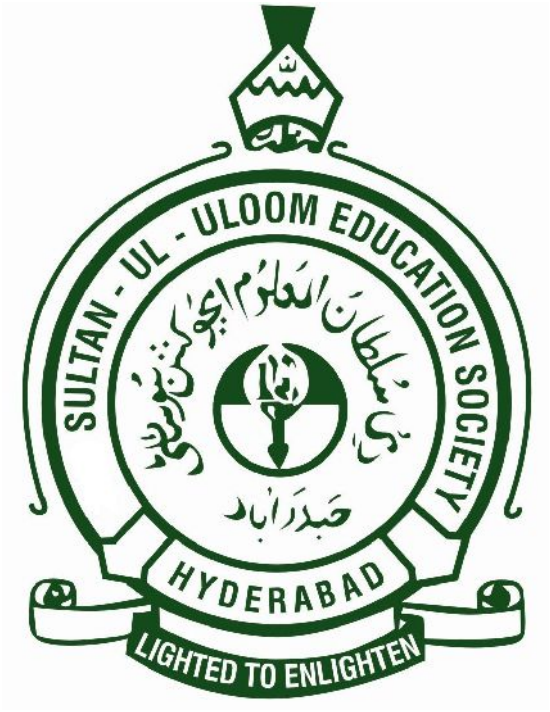
MICRO PROCESSORS & MICROCONTROLLERS LAB (EE 432)

LABORATORY MANUAL

IV/IV B.E I SEM EEE/EIE



DEPARTMENT OF ELECTRICAL ENGINEERING

MUFFAKHAM JAH COLLEGE OF ENGINEERING & TECHNOLOGY
Banjara Hills Road No 3, Hyderabad 34
www.mjcollege.ac.in

2014-15

Prepared By: G. RAVI KIRAN, Asst.Professor

MICROPROCESSORS & MICROCONTROLLERS LAB (EEE & EIE)

LIST OF EXPERIMENTS

Using MASM

Demo: (A) Addition of two 8 Bit/ 16 Bit Numbers.
(B) Subtraction of two 8 Bit/ 16 Bit Numbers.

1. (a) Programs for Signed/Unsigned Multiplication.
   (b) Program for Unsigned Division.

2. Program to find Average of 8 Bit/ 16-Bit Numbers in an Array.

3. (a) Program for finding the largest number in an Array.

   (b) Program for finding the smallest number in an Array.

4. (a)Programs for code conversion like BCD numbers to seven segment.

   (b) Program for searching a number in an array.

5. (a) Programs for computing factorial of a positive integer number.

   (b) Program to find number of one's in a given 8- bit number.

USING 8086 KIT

6. 8255 – PPI: ALP to generate Triangular wave using DAC

   (a) Program to generate Sawtooth wave form.
   (b) Program to generate Triangular wave form.
   (c) Program to generate Square wave form.

USING 8051 KIT

7. Arithmetic Instructions: Multibyte Operations
   (a) Program for addition/subtraction of two 16 bit numbers.
   (b) Program for multiplication/division of two 16 bit/32 bit numbers.

8. Data Transfer – block move, exchange, sorting, finding largest number in an array.
   (a) Program for finding maximum/minimum number in an array.
   (b) Program for exchange of data.

9. Boolean & Logical Instructions (Bit Manipulations)
   Program for reverse & logical 'OR' of a given number.

10. Traffic Light Controller.

USING 'C' Cross Compiler (KEIL Software)

11. Program for activating ports and generation of square wave.

12. (a) Program to find addition of two numbers.
    (b) Program of Multibyte Addition

13. (a) Program for ascending order/descending order of a given numbers
    (b) Program for data transfer.

INTRODUCTION TO MASM

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. The major different between compilers for a high level language like PASCAL and an Assembler is that the compiler usually emits several machine instructions for each PASCAL statement. The assembler generally emits a single machine instruction for each assembler language statement.

Attempting to write a program in machine language is not particularly bright. This process is very tedious, mistakes, and offers almost no advantages over programming in assembly language. The major disadvantages over programming in assembly language over pure machine code are that you must first assemble and link a program before you can execute it. However attempting to assemble the code by hand would take for longer than the small amount of time that the assembler takes the perform conversion for you. An assembler like Microsoft Macro Assembler (MASM) provides a large number of features for assembly language programmers. Although learning about these features take a fair amount of time. They are so useful that it is well worth the effort.

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. To assemble the file PROG.ASM use this command: (better to use DOS command line)

MASM PROG.ASM

The MASM program will assemble the PROG.ASM file. (To create PROG.OBJ from PROG.ASM)

To create PROG.EXE from PROG.OBJ, use this LINK command:

LINK PROG.OBJ

It converts the contents of PROG.OBJ into PROG.EXE.

To link more than one object file use + signs between their file names as in:

LINK PROGA+PROGB+PROGC

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	locate 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 only displays the 16-bit portion of the general purpose registers. Code view is capable of displaying the entire 32 bits. DEBUG is a very useful debugging tool. We will use DEBUG to step through a number of simple programs, gaining familiarity with Debug's commands as we do so. 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. If an .EXE or .COM file were specified, DEBUG would load the program according to accepted DOS conventions.

To execute the program file PROG.EXE use this command

DEBUG PROG.EXE

DEBUG uses a minus sign as its command prompt, so should see a "-" appear on display.

To get a list of some commands available with DEBUG is :

T ➔ trace (step by step execution)
U ➔ un assemble
D ➔ dump
G ➔ go (complete execution)
H ➔ Hex

To execute the program file PROG.ASM use the following procedure:

.MASM PROG.ASM
.LINK PROG.OBJ
.DEBUG PROG.EXE

ASSEMBLER DIRECTIVES: The limits are given to the assembler using some pre defined alphabetical strings called Assembler Directives which help assembler to correctly understand. The assembly language programs to prepare the codes.

DB	GROUP	EXTRN
DW	LABEL	TYPE
DQ	LENGTH	EVEN
DT	LOCAL	SEGMENT
ASSUME	NAME	
END	OFFSET	
ENDP	ORG	
ENDS	PROC	
EQU	PTR	

DB-Define Byte: The DB drive is used to reserve byte of memory locations in the available on memory.

DW-Define Word: The DW drive is used to reserve 16 byte of memory location available on memory.

DQ-Define Quad Word (4 words): The DB directives is used to reserve 8 bytes of memory locations in the memory available.

DT-Define Ten Byte: The DT directive is used to reserve 10 byte of memory locations in the available memory.

ASSUME: Assume local segment name the Assume directive is used to inform the assembler. The name of the logical segments to be assumed for different segment used in programs.

END: End of the program the END directive marks the end of an ALP.

ENDP: End of the procedure.
ENDS: End of the segment.
EQU: The directive is used to assign a label with a variable or symbol. The directive is just to reduce recurrence of the numerical values or constants in the program.

OFFSET: Specifies offset address.
SEGMENT: The segment directive marks the starting of the logical segment.

EXECUTION OF ASSEMBLY LANGUAGE PROGRAMMING IN MASM SOFTWARE:

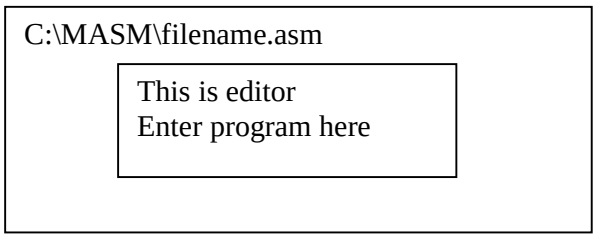
Assembly language programming has 4 steps.

1. Entering Program
2. Compile Program
3. Linking a Program
4. Debugging a Program

PROCEDURE:

1. Entering Program:-

Start Menu
Run ↵
Cmd ↵
C:\cd MASM ↵
C:\ MASM> edit filename.asm ↵



After entering program save & exit (ALT F & Press S or ALT F &Press X)
C:\MASM>

2. Compile the Program:-

C:\MASM> MASM filename.asm

Microsoft @macro assembler version 5.10
Copy rights reserved© Microsoft
Corp 1981 All rights reserved
Object filename [OBJ];
List filename [NUL, LIST];
Cross Reference [NUL, CRF];
Press enter the screen shows c>

3. Linking a Program:-

c> link filename.obj ↵
Microsoft @ overlay linker version 3.64
Copy rights reserved© Microsoft corp.
1983-88. All rights reserved
Object module [.OBJ];
Run file [.EXE];
List [NUL MAP];
Libraries [LIB];
Press enter till screen chows c>

4. Debug a Program:-
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.

OPERATION OF 8051 KIT

- Switch on power supply. Message"ANSHUMAN" will be displayed.
- Press"E" &then "ENTER" key.
- Select C=A & then press enter .default 6000 address will be displayed.
  - Note: for changing address select C=A address.
- Now enter the program. At the end press "ENTER" key twice.
- Then C= will be displayed. Press "Q".
- Press "S" & press enter.
- By pressing any key, select, EXT. memory, register. etc. &press "enter" key.
- For register, select general (AS, DPL, DPR etc),BANK etc. press enter.
- Now enter the inputs &press enter key
- Press "G" press "enter" key.
- BURST will be displayed. Press enter.
- ADDR will be displayed. Esc 6000&press enter.
- Wait, DONE message will be displayed.
- Now to view output, press "S"& press "ENTER".

OPERATION OF 8086 KIT

PROCEDURE RO OPERATE ANSHUMAN KIT 8086:

1. **ENTERING THE PROGRAM AND DATA**:
   - Switch on the power supply. "Anshuman" is displayed
   - Press 'E' & then ENETER
   - Lity will be displayed (for Utility commands) selecting A (Assembler) & D (Dissembler) press enter key. Entering mnemonics into kit, Press 'A' followed by starting address Enter simply Press "A" Default address 0100 will be selected.
   - Now enter the mnemonics of 8086 into kit type "INT A5" or "RET" for terminating the program. & Enter twice:
   - Press "Q" and then enter.
   - Press "S" & then enter
   - Memory will be displayed & press "enter key"
   - SRC-SEGM will be displayed.
   - Here type the address 0000, & Press enter
   - ADDR will be displayed

     Type the starting address, where data will be entered (0100) & Press enter key & enter the data. After entering data press ESC.

2. **EXECUTE THE PROGRAM**:
   - Press 'G' & enter.
   - BURST will be displayed then press enter key
   - SCR-SEGM will appear enter the default address 1000 & press enter key.
   - ADDR will appear enter the starting segment say 0100 & press enter key.
   - Message "Wait" command will appear.

3. **TO CHECK THE RESULT**:
   - Press 'S' & then press enter key.
   - Memory will be displayed & then press enter key.
   - SCR-SEGM will appear, here enter 0000 & press enter.
   - ADDR will appear. Now type the address of the output location/port to see the result.
   - Next .

PROCEDURE FOR TRAFFIC LIGHT CONTROLLER

To run the program for 8051 kit through EEPROM for traffic light control

- Connect 8051 kit to CPU through RS232 cable.
- Connect the Traffic Light Controller kit to 8051 kit through 24-pin FRC cable.
- In computer, Go to Programs, click on Accessories and select Communications and click on Hyper terminal, then a new window will be open.
- Give any name like “mjcet”, then another new window will be opened , click “OK”.
- In the COM1 properties window, select stop bits as 2 and click “OK”
- press reset in kit
- make Caps lock on from pc key board Press I> Enter
- PRTY > TYPE N
- NO PRTY> ENTER
- HEX>ENTER
- STRT 75B0
- END
- 77FF
- WAIT
- TRANSFER >SELECT TEXT FILE AND SHOW .HEX FILE ADDRESS (folder XPO51>XPO31ACC>TRAF2K3C.HEX)
- Eg Traf2 k3c.HEX
- DISPLAY COMMAND=
- Enter G ,Add will be displayed then type 75b0 (Starting address of program)
- wait will be displayed check results on the traffic kit

PROCEDURE FOR PROGRAMS ON KEIL SOFTWARE

- Click on Keil uvision3.
- Click on **'Project'**, create a new project and save it in a new folder choose target option for **Atmel** and **AT89C51**.
- Go to File, click on new file, and type the program.
- Go to File, click on **'save as'**, save the program with extension **.asm** on your particular folder where you saved your project.
- Add your program to **Source Group 1** which is at Target1 (Project workspace) which is created after selecting the target in step 2.
    - To do this right clicks on Source Group 1 and select 'Add files to Source Group 1'.
    - Search your code with .asm extension.
- Now Click on **Translate current file** tab present file toolbar and check for errors. If error present then rectify.
- Click on **Rebuild all target files** to add our program to the AT89C51 target.
- Go to **Debug**, click on S**tart/Stop debug session**.
- For giving input data: Go to **view**, click on **Memory window**.
    - Enter **inputs** for corresponding memory addresses.
        - For internal memory type: i:0x20 for example
        - For external memory type: x:0x2000 for example
- Now click on **"Run",** check the results.
- While in Debug don't make any changes in the program.
- After running again click S**tart/Stop debug session** to edit mode for changes in program.

Demo Program (A):

ADDITION OF TWO 16 BIT NUMBERS

AIM: To implement assembly language program for addition of two 16-bit numbers.

APPARTUS: MASM Software, P.C.

PROGRAM:

```
DATA SEGMENT
        N1 DW 1234H
        N2 DW 2134H
        RES DW ?
DATA ENDS

CODE SEGMENT
      ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA
        MOV DS, AX
        MOV AX, N1
        MOV BX, N2
        ADD AX, BX
        MOV RES, AX
        INT 21H

CODE ENDS

END START
```

RESULT:

AX = 3368h

Demo Program (B):

SUBTRATION OF TWO 16 BIT NUMBERS

AIM: To implement assembly language program for subtraction of two 16-bit numbers.

APPARTUS: MASM Software, P.C.

PROGRAM:

```
DATA SEGMENT
        N1 DW 4444H
        N2 DW 2121H
        RES DW ?
DATA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:DATA

START: MOV AX,DATA
        MOV DS,AX
        MOV AX,N1
        MOV BX,N2
        SUB AX,BX
        MOV RES,AX
        INT 21H

CODE ENDS


END START
```

RESULT:

AX = 2323h

Exp.No.01 (a)

ASSEMBLY LANGUAGE PROGRAM TO MULTIPLY TWO 16-BIT SIGNED/UNSIGNED NUMBERS

AIM: To implement assembly language program to multiply two 16-bit **signed numbers**.

APPARTUS: MASM Software and PC

ALGORITHM:

1. Start.
2. Initialize the data segment
3. Load the first number in AX Register.
4. Load the second number in BX Register.
5. Perform the multiplication of the two 16-bit numbers.
6. Store the result in AX Register.
7. Stop.

PROGRAM:

```
DATA SEGMENT

DATA ENDS

CODE SEGMENT
        ASSUME CS: CODE, DS: DATA

START:


          INT 21H

CODE ENDS
END START
```

RESULT:

AIM: To implement assembly language program to multiply two 16-bit **unsigned numbers**.

APPARTUS: MASM Software and PC

ALGORITHM:

1. Start.
2. Initialize the data segment
3. Load the first number in AX Register.
4. Load the second number in BX Register.
5. Perform the multiplication of the two 16-bit numbers.
6. Store the result in AX Register.
7. Stop.

PROGRAM:

```
DATA SEGMENT



DATA ENDS

CODE SEGMENT
        ASSUME CS: CODE, DS: DATA

START:




CODE ENDS
END START
```

RESULT:

Exp.No. 01(b)

ASSEMBLY LANGUAGE PROGRAM FOR UNSIGNED DIVISION OF TWO NUMBERS

AIM: To implement assembly language program to divide 32-bit with 16-bit numbers.

APPARTUS: MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize the data segment.
3. Load the higher 32-bit number to be divided in AX.
4. Load the 16-bit number in DX register.
5. Take the division in BX register.
6. Perform the unsigned division.
7. Store the result in AX register.
8. End.

PROGRAM:

```
DATA SEGMENT



DATA ENDS
CODE SEGMENT
        ASSUME CS: CODE, DS: DATA
START:




CODE ENDS
 END START
```

RESULT:

Demo:

ASSEMBLY LANGUAGE PROGRAM TO FIND SUM OF NUMBERS IN AN ARRAY

AIM: To implement ALP to find sum of numbers in the array.

APPARTUS:

MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize counter = 10.
3. Initialize array pointer.
4. Sum = 0.
5. Get the array element pointed by array pointer.
6. Add array element in the Sum.
7. Increment array pointer decrement counter.
8. Repeat steps 4, 5 & 6 until counter = 0.
9. Display Sum.
10. Stop.

PROGRAM:

```
DATA SEGMENT
      ARRAY DB 12H, 24H, 26H, 63H, 25H, 86H, 2FH, 33H, 10H, 35H
      SUM      DW 0
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA

START:        MOV AX, DATA
              MOV DS, AX
              MOV CL, 10
              XOR DI, DI
              LEA BX, ARRAY
BACK:         MOV AL, [BX+DI]
              MOV AH, 00H
              MOV SUM, AX
              INC DI
              DEC CL
              JNZ BACK
              INT 21
CODE ENDS
              END START
```

RESULT:

AX = 0211h

Exp.No. 02

ASSEMBLY LANGUAGE PROGRAM TO FIND AVERAGE OF 8-BIT NUMBERS IN AN ARRAY

AIM: To implement ALP to find average of 8-bit numbers in array.

APPARTUS:

MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize the data segment.
3. Initialize counter = 0.
4. Initialize pointer.
5. Initialize array base pointer.
6. Get the number.
7. Add sum to the number i.e. add array element.
8. Increment array pointer decrement counter.
9. Repeat steps 6, 7 & 8 under counter = 0.
10. Display average.
11. Stop.

PROGRAM:

```
DATA SEGMENT



DATA ENDS


CODE SEGMENT
ASSUME CS: CODE, DS; DATA


START:



        INT 03H
        CODE ENDS
END START
```

RESULT:

ASSEMBLY LANGUAGE PROGRAM TO FIND AVERAGE OF 16-BIT NUMBERS IN AN ARRAY

AIM: To implement ALP to find average of 16-bit numbers in array.

APPARTUS:

MASM Software, P.C.

ALGORITHM:

12. Start.
13. Initialize the data segment.
14. Initialize counter = 0.
15. Initialize pointer.
16. Initialize array base pointer.
17. Get the number.
18. Add sum to the number i.e. add array element.
19. Increment array pointer decrement counter.
20. Repeat steps 6, 7 & 8 under counter = 0.
21. Display average.
22. Stop.

PROGRAM:

```
DATA SEGMENT


DATA ENDS

CODE SEGMENT
ASSUME CS: CODE, DS; DATA

START:


        INT 21H
        CODE ENDS
END START
```

RESULT:

Exp.No. 3(A)

ASSEMBLY LANGUAGE PROGRAM TO FIND LARGEST NUMBER IN AN ARRAY

AIM: To implement ALP to find the maximum number in the array.

APPARTUS:

MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize data segment.
3. Initialize the pointer.
4. Initialize counter = 0.
5. Initialize the array base pointer.
6. Get the maximum number.
7. Compare the number with maximum number.
8. If num> MAX, Max = num & increment pointer.
9. Decrement the counter.
10. If count = 0 stop or else repeat steps 6, 7, 8, 9.
11. Store maximum number.
12. Stop.

PROGRAM:

```
DATA SEGMENT



DATA ENDS

CODE SEGMENT
        ASSUME CS:CODE,DS:DATA

START:



        HLT

CODE ENDS
END START
```

RESULT:

Exp.No. 03(B)

ASSEMBLY LANGUAGE PROGRAM TO FIND SMALLEST IN AN ARRAY

AIM: To implement ALP to find the minimum in the array.

APPARATUS:

MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize the data segment.
3. Initialize the pointer.
4. Initialize counter = 0.
5. Initialize base pointer for an array.
6. Get the minimum number.
7. Compare number with minimum number.
8. If number < MIN, MIN = NUM & increment pointer.
9. Decrement the counter.
10. If count = 0 Stop ; otherwise go to BACK.
11. Store the minimum number.
12. Stop.

PROGRAM:

```
DATA SEGMENT



DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:


    HLT

CODE ENDS

END START
```

RESULT:

Exp.No.04(A)

ASSEMBLY LANGUAGE PROGRAM TO CONVERT BCD TO SEVEN SEGMENT

AIM: To implement ALP to convert BCD to SEVEN SEGMENT.

APPARATUS:

MASM Software, P.C.

ALGORITHM:

1. Initialize the data segment.
2. Get the first number in AL.
3. Load BX with the starting address of lookup table.
4. Result is displayed.
5. Stop.

PROGRAM:

```
DATA SEGMENT


DATA ENDS
CODE SEGMENT
        ASSUME CS: CODE, DS: DATA

START:



        INT 21h
CODE ENDS
        END START
```

RESULT:

Exp.No. 4(B)

ASSEMBLY LANGUAGE PROGRAM TO SEARCH A NUMBER IN AN ARRAY

AIM: To implement ALP to search a number in an array.

APPARATUS: MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize the data segment.
3. Initialize the counter.
4. Initialize base pointer for array.
5. Get the number to be searched in AL.
6. Clear direction flag.
7. Scan & check CX = 0.
8. Result is displayed.
9. Stop.

PROGARM:

```
DATA SEGMENT



DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:


        INT 21H
CODE ENDS
        END START
```

RESULT:

Exp.No. 5(A)

ASSEMBLY LANGUAGE PROGRAM TO FIND FACTORIAL OF A GIVEN NUMBER

AIM: To implement ALP to find factorial of a number.

APPARATUS:

MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize data segment.
3. Get the number in AL.
4. Multiply the number with 8-bit number present in CL.
5. Increment the counter.
6. Compare with no.1
7. Display factorial of number.
8. Stop.

PROGRAM:

```
DATA SEGMENT


DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START


        INT 21H
CODE ENDS
        END START
```

RESULT:

Exp.No. 05(B)

ASSEMBLY LANGUAGE PROGRAM TO FIND NO. OF ONE'S IN A GIVEN 8-BIT NUMBER

AIM: To implement ALP to find number of ONE's in a given 8-bit number.

APPARATUS: MASM Software, P.C.

ALGORITHM:

1. Start.
2. Initialize the data segment.
3. Clear the base register.
4. Initialize the counter.
5. Rotate the number, check for '1'.
6. Result is displayed.
7. Stop.

PROGRAM:

```
DATA SEGMENT


DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:


INT 3H

CODE ENDS
END START
```

RESULT:

Exp.No. 06

ASSEMBLY LANGUAGE PROGRAM TO GENERATE TRIANGULAR, SQUARE & SAWTOOTH USING DAC

AIM: Write an 8086 program to interface 8255 PPI.

1. Generate saw tooth wave
2. triangular wave
3. Square wave using DAC interfacing

APPARATUS: 1) MP 8086 trainer kit
2) SMPS
3) DAC Interface module
4) Power Supply (5V)
5) 26 pin flat ribbon cable
6) 4/8 wire relamatic cable
7) Oscilloscope
8) CRO probes

6(A). GENERATION OF SAW TOOTH WAVE:-

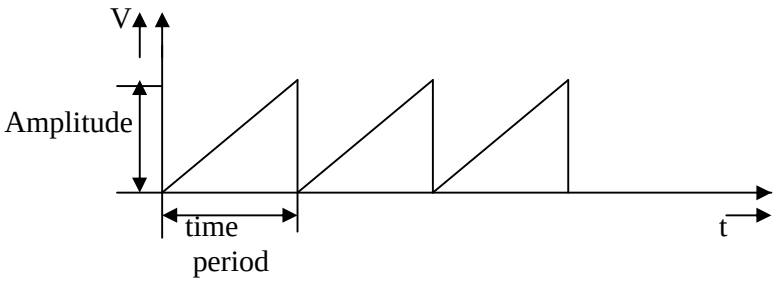
ALGORITHM:

1. Port B is selected.
2. Contents of accumulator are initialized to zero.
3. Data is send to port.
4. Contents of accumulator are increased.
5. Comparing immediate with FF.
6. Jump on no zero to step 3.
7. Sending data to the port.
8. Jump to step 6.

PROGRAM:

```
        MOV  DX, 8807H
        MOV  AL, 80H
        OUT  DX, AL
        MOV  AL, 00H
        MOV  DX, 8801H
L1:     OUT  DX, AL
        INC  AL
        JMP  L1
```

EXPECTED WAVEFORM:



EXPECTED RESULT:

Amplitude = Frequency =

Time Period =

6(B). TRIANGULAR WAVE GENERATION:

ALGORITHM:

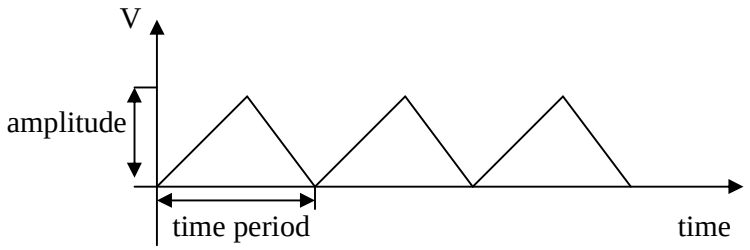
1. Port B is selected.
2. Contents of acc. are initialized to zero.
3. Data is sent to port.
4. Content of acc is incremented.
5. Comparing immediate with FF.
6. Jump on no zero to step1.
7. Send data to the port.
8. Contents of acc. are decremented.
9. Compare immediate with 00.
10. Jump on non zero to step7.
11. Jump to step3.
12. Output on CRO is obtained.

PROGRAM:

```
        MOV   DX, 8807H
        MOV   AL, 80H
        OUT   DX, AL
        MOV   AL, 00H
        MOV   DX, 8801H
L1:     OUT   DX, AL
        INC   AL
        CMP   AL, FFH
        JNZ   L1
```

```
L2:   OUT   DX, AL
      DEC   AL
      JNZ   L2
      JMP   L1
      INT   A5
```

EXPECTED WAVEFORM:



EXPECTED RESULT:

Amplitude = Frequency =

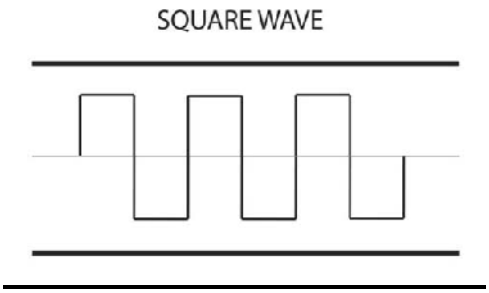
Time Period =

6(C). SQUARE WAVE FORM GENERATION

```
0100   MOV   DX, 8807H

       MOV   AL, 80H; Activation of port A

0105   OUT   DX, AL; Output to I/O Port

       MOV   AL, 00H; Initialize AL with 00

       MOV   DX, 8801H

       OUT   DX, AL; Output to I/O Port

       CALL  0117; program control to 0117 location

       MOV   AL, FFH ;Initialize AL with FF

       OUT   DX, AL; Output to I/O Port
```

```
CALL  0117;  program control to 0117 location

JMP  0105;  jump to location of 0105

0117 MOV CX,015D;setting the amplitude

NOP ; no operation

NOP ; no operation

RET
```

EXPECTED WAVEFORM:



EXPECTED RESULT:

Amplitude = Frequency =

Time Period =

Exp. No. 7(A)(i)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIBYTE ADDITION

AIM: Write 8051 program to implement multiple byte addition (addition of two 32-bit no's).

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

Generally 8 bits are called a byte, 16 bits are called as word, 32 bits are called as double word, and the data more than 4 byte is called as Multiple byte.

ALGORITHM:

1. Start.
2. Get the number 100.Get the first number.
3. Add result with second number.
4. Store in R_0 (or) in first number register.
5. Repeat the step for given no. of inputs.
6. Output is displayed in R_0, R_1, R_2, R_3.
7. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	A, R_0
6001	ADD	A, R_4
6002	MOV	R_0, A
6003	MOV	A, R_1
6004	ADDC	A, R_5
6005	MOV	R_1, A
6006	MOV	A, R_2
6007	ADDC	A, R_6
6008	MOV	R_2, A
6009	MOV	A, R_3
600A	ADDC	A, R_7
600B	MOV	R_3, A
600C	RET	

EXPECTED RESULTS:

Inputs: $R_0 = 11h$, $R_1 = 11h$, $R_2 = 11h$, $R_3 = 11h$
$R_4 = 22h$, $R_5 = 22h$, $R_6 = 22h$, $R_7 = 22h$

Outputs: $R_0 = 33h$, $R_1 = 33h$, $R_2 = 33h$, $R_3 = 33h$

Exp. No. 7(A)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIBYTE SUBTRACTION

AIM: Write 8051 program to implement subtraction of two 32 bit numbers.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

Generally 8 bits are called a byte, 16 bits are called as word, 32 bits are called as double word. Here we are subtracting two bytes, which are stored in the register. By using the instruction SUBB we can subtract byte by byte.

ALGORITHM:

1. Start.
2. Get the first number.
3. Subtract with the second number.
4. Store result in R_0.
5. Repeat the above steps for given no. of inputs.
6. Output is displayed in R_0, R_1, R_2, R_3.
7. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERAND
6000	CLR	C
6001	MOV	A, R_0
6002	SUBB	A, R_4
6003	MOV	R_0, A
6004	MOV	A, R_1
6005	SUBB	A, R_5
6006	MOV	R_1, A
6007	MOV	A, R_2
6008	SUBB	A, R_6
6009	MOV	R_2, A
600A	MOV	A, R_3
600B	SUBB	A, R_7
600C	MOV	R_3, A
600D	RET	

EXPECTED RESULT:

Inputs: $R_0 = 55h$, $R_1 = 55h$, $R_2 = 55h$, $R_3 = 55h$
$R_4 = 22h$, $R_5 = 22h$, $R_6 = 22h$, $R_7 = 22h$

Outputs: $R_0 = 33h$, $R_1 = 33h$, $R_2 = 33h$, $R_3 = 33h$

Exp. No. 7(B)(i)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIPLICATION OF 32-BIT NUMBERS

AIM: Write 8051 program to implement multiplication.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

After multiplication, if it is 16 bit multiplication the result will be stored in register A and register B. If it is 8 bit multiplication then the result will be store in register A.

ALGORITHM:

1. Start.
2. Get the first number.
3. Store the number.
4. Get the second number.
5. Multiply A & B.
6. Increment data pointer.
7. Get the higher byte & lower byte of result.
8. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	DPTR, #20A1
6003	MOVX	A, @DPTR
6004	MOV	F_0, A
6006	MOV	DPTR, #20A0
6009	MOVX	A, @DPTR
600A	MUL	AB
600B	MOV	DPTR, #20A2
600E	MOVX	@DPTR, A
600F	INC	DPTR
6010	MOV	A, F_0
6012	MOVX	@DPTR, A
6013	RET	

EXPECTED RESULT:

Inputs: 20A0 = 05h & 20A1 = 04h

Output: 20A2 = 14h

Exp. No. 7(B)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR DIVISION OF TWO 8 BIT NUMBERS

AIM: Write 8051 program to implement division operation.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

After division the quotient is stored in register 'A' and the reminder will be stored in register 'B'.

ALGORITHM:

1. Start.
2. Get the first number.
3. Store the number.
4. Get the second number.
5. Divide A & B.
6. Increment data pointer.
7. Get the quotient, reminder & display.
8. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	A, #00H
6003	MOV	DPTR, #20A0
6004	MOVX	A, @DPTR
6006	MOV	F_0, A
6009	MOV	A, #00H
600A	INC	DPTR
600B	MOVX	A, @DPTR
600C	DIV	A, B
600D	INC	DPTR
600E	MOVX	@DPTR, A
6011	MOV	A, F_0
6012	INC	DPTR
6013	MOVX	@DPTR, A
6014	RET	

EXPECTED RESULT:

Inputs: 20A0 = 15h & 20A1 = 03h

Output: 20A2 = 07h & 20A3 = 00h

Exp. No. 8(A)(i)

ASSEMBLY LANGUAGE PROGRAM FOR FINDING MAXIMUM NUMBER FROM 8-BIT TEN NUMBERS

AIM: Write a program for finding the minimum number from 8-bit ten numbers in 8051 kit.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

PROGRAM:

```
6000    MOV   DPTR, #7000        ; initialize the pointer to memory where
                                   numbers are stored
6003    MOV   R0, #0A            ; initialize the counter
6005    MOV   F0, #00            ; maximum = 0
6008    AGAIN: MOVX A, @DPTR     ; get the number from the memory
6009    CJNE A, F0, 02           ; NE = 600E – 600C=02, compare number with
                                   maximum
600C    AJMP 6012                ; address of SKIP = 6012, if equal go to SKIP
600E    NE: JC 02                ; SKIP = 6012- 6010, if not equal check for
                                   carry, if carry go to SKIP
6010    MOV F0,A                 ; otherwise maximum  = number
6012    SKIP: INC DPTR           ; increment memory pointer
6013    DJNZ R0,F3               ; AGAIN = FF – (6013-6007), decrement
                                   count, if count = 0 stop, otherwise go to AGAIN
6015    RET
```

EXPECTED RESULT:

Input:

```
7000    08
7001    02
7002    03
7003    05
7004    06
7005    01
```

7006 04
7007 07
7008 19
7009 00

OUTPUT

B=19h

Forward Jump:

For SKIP and NE label=

Address of location where to jump – address of location of next instruction after jump instruction => 600E-600C=02

Backward Jump:

For AGAIN label=

No. of bytes= (address of location of the count)-(address of location where to jump)

Count=FF- No. of bytes=FF-(6013-6007)=F3

Exp. No. 8(A)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR FINDING MINIMUM NUMBER FROM 8-BIT TEN NUMBERS

AIM: Write a program for finding the minimum number from 8-bit ten numbers in 8051 kit.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

PROGRAM:

```
6000    MOV   DPTR, #7000 ; initialize the pointer to memory where
                            numbers are stored

6003    MOV   R0, #0A     ; initialize the counter

6005    MOV   F0, #FF     ; minimum =FF

6008    AGAIN: MOVX A, @DPTR   ; get the number from the memory

6009    CJNE A, F0, 02    ; NE = 600E – 600C=02, compare number with
                            minimum

600C    AJMP 6012         ; address of SKIP = 6012, if equal go to SKIP

600E    NE: JNC 02        ; SKIP = 6012- 6010, if not equal check for
                            carry, if carry go to SKIP

6010    MOV F0,A          ; otherwise minimum  = number

6012    SKIP: INC DPTR    ; increment memory pointer

6013    DJNZ R0,F3        ; AGAIN = FF – (6013-6007), decrement

        count, if count = 0 stop, other wise go to
                            AGAIN
6015    RET
```

RESULT:

INPUT:

7000 08
7001 02
7002 03
7003 05
7004 06
7005 01
7006 04
7007 07
7008 19
7009 05

OUTPUT

B=01h

Exp. No. 8(B)

ASSEMBLY LANGUAGE PROGRAM FOR EXCHANGE OF DATA

AIM: Write a program for exchange of data in 8051.

APPARATUS:

3. MC 8051 trainer kit
4. SMPS

ALGORITHM:

1. Start.
2. Get the first number in Accumulator
3. Get the second number in R_0
4. Swap A, and exchange with R_0.
5. Display the result.
6. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	A, #C5H
6002	MOV	R_0, #C6H
6004	SWAP	A
6005	XCH	A, R_0
6006	RET	

EXPECTED RESULT:

'A' becomes 5Ch and moved to **R_0 = 5Ch**

R_0 = C6h is moved to **A = C6h**

Exp. No. 9

ASSEMBLY LANGUAGE PROGRAM FOR REVERSE AND LOGICAL 'OR'

AIM: Write a program for reverse the numbers and apply logic instruction OR gate to the given numbers using 8051kit.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

PROGRAM:

```
MOV DPL, #34     ; instead of dpl, type 82
MOV DPH, #12     ; instead of dph, type 83
MOV A, DPL
RL A
RL A
RL A
RL A
MOV DPL, A
MOV A, DPH
RL A
RL A
RL A
RL A
MOV DPH, A
ORL A, DPL
RET
```

EXPECTED RESULT:

Logical 'OR' result for given numbers 43h & 21h is A = 63h

DPL= 43h

DPH =21h

Exp. No. 10

ASSEMBLY LANGUAGE PROGRAM FOR TRAFFIC LIGHT CONTROLLER

AIM: Write an assembly language program for block traffic light control using keil software (AT89C51).

APPARATUS:

1. 8051 kit
2. P.C.
3. Traffic light control kit
4. 24 pin FRC cable

PROGRAM:

```
        MOV P2,#0H          ;PORT2
        MOV P1,#0H          ;PORT1
        MOV P3,#0H          ;PORT3
        MOV P0,#0H          ;PORT0


        MOV P0,#61H
        MOV P1,#68H
        MOV P3,#86H
        ACALL DLY4      ; DELAY OF 4 SEC FOR RED LED

STRT:
        MOV P0,#64H
        MOV P1,#58H
        MOV P3,#86H
        ACALL DLY4      ; 1 GREEN


        MOV P0,#62H
        MOV P1,#68H
        MOV P3,#86H
        ACALL DLY2      ;2 YELLOW

        MOV P0,#61H
        MOV P1,#68H
        MOV P3,#86H
        ACALL DLY4      ;3ARED


        MOV P0,#49H
        MOV P1,#68H
        MOV P3,#26H
        ACALL DLY4      ;3BGREEN

        MOV P0,#51H
        MOV P1,#68H
```

```
        MOV P3,#46H
        ACALL DLY2      ;4 YELLOW

        MOV P0,#61H
        MOV P1,#68H
        MOV P3,#86H
        ACALL DLY4      ;5A RED

        MOV P0,#61H
        MOV P1,#62H
        MOV P3,#92H
        ACALL DLY4      ;5B GREEN

        MOV P0,#61H
        MOV P1,#64H
        MOV P3,#8AH
        ACALL DLY2      ;6 YELLOW

        MOV P0,#61H
        MOV P1,#68H
        MOV P3,#86H
        ACALL DLY4      ;7A RED

        MOV P0,#21H
        MOV P1,#29H
        MOV P3,#87H
        ACALL DLY4      ;7B GREEN

        MOV P0,#0A1H
        MOV P1,#0A8H
        MOV P3,#86H
        ACALL DLY2      ;8A YELLOW

        MOV P0,#61H
        MOV P1,#68H
        MOV P3,#86H
        ACALL DLY4      ;8B RED

        LJMP STRT

DLY4:   LCALL DELAY     ; DELAY FOR 4 SEC
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        RET

DLY2:   LCALL DELAY     ; DELAY FOR 2 SEC
        LCALL DELAY
        RET

DELAY:                  ; DELAY FOR 1 SEC
```

```
        MOV R3,#0FH
D3:     MOV R1,#0FFH
D2:     MOV R2,#0FFH
D1:     DJNZ R2, D1
        DJNZ R1,D2
        DJNZ R3,D3
        RET
```

RESULT:

Traffic light control is executed successfully

Exp. No. 11

ASSEMBLY LANGUAGE PROGRAM FOR ACTIVATING PORTS & GENERATION OF SQUARE WAVE

AIM: Write an assembly language program for generating square waveform using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM(1):

```
MOV SP,#7H
CLR P1.0
BACK: MOV P1,#00H
ACALL DELAY
SETB P1.0
MOV P1,#0FFH
ACALL DELAY
SJMP BACK
DELAY:MOV R1,#0FFH
AGAIN:DJNZ R1,AGAIN
RET
END
```

PROGRAM(2):

```
MOV SP,#7H              ;initialize stack pointer
                        ;since we are using subroutine programe
BACK:MOV P1,#00H        ; send 00h on port 1 to generate
                        ;low level of square wave
ACALL DELAY             ; wait for some time
MOV P1,#0FFH            ;send ffh on port 1 to generate
                        ;high level of square wave
ACALL DELAY             ; wait for some time
SJMP BACK               ;repeat the sequence
DELAY:MOV R1,#0FFH      ;load count
AGAIN:DJNZ R1,AGAIN     ;decrement count and repeat the process
                        ;until count is zero
RET                     ;return to main programe
```

EXPECTED RESULTS:

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
✓	✓	✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓	✓	✓

Program (1) : Activating PORT1

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
							✓
							✓

Program (2) : Activating Individual PORT1 pin 0

Exp. No. 12(A)

ASSEMBLY LANGUAGE PROGRAM FOR ADDITION OF TWO NUMBERS

AIM: Write an assembly language program for adding two 8-bit numbers using keil Software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM:

```
MOV A, #05H
MOV B,#02H
ADD A,B
END
```

RESULT:

In accumulator, a= 7h

Exp. No. 12(B)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIBYTE ADDITION

AIM: Write an assembly language program for multibyte addition using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM:

```
MOV R0,#20H
MOV R1,#30H
MOV R3,#04H
CLR C
CLR A
AGAIN:   MOV A,@R0
         ADDC A,@R1
         MOV @R1,A
         INC R0
         INC R1
         DJNZ R3,AGAIN
      END
```

RESULT:

Inputs:

i: 0x20 -- 01h, 02h, 03h, 04h
i: 0x30 -- 05h, 06h, 07h, 08h

Output:

i: 0x30 -- 06h, 08h, 0Ah, 0Ch

Exp. No. 13(A)(i)

ASSEMBLY LANGUAGE PROGRAM FOR ASCENDING ORDER OF A GIVEN NUMBERS

AIM: Write an assembly language program for arranging in ascending/descending order using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM FOR ASCENDING ORDER:

```
        MOV R0,#5            ; INITIALIZE COUNTER 1
AGAIN:  MOV DPTR,#2000H      ; initialize memory pointer
        MOV R1,#4            ; initialize counter 2
BACK:   MOV R2,DPL           ; save lower byte of memory address
        MOVX A,@DPTR         ; Get the num ber
        MOV B,A              ; Save the number
        INC DPTR             ; Increment the memory pointer
        MOVX A,@DPTR         ; Get the next number
        CJNE A,B,n           ; If not equal check for greater or less
        AJMP SKIP            ; Otherwise go to skip
   n:   JNC SKIP             ;If
        MOV DPL,R2           ;Exchange
        MOVX @DPTR,A
        INC DPTR
        MOV A,B
        MOVX @dptr,A
SKIP:   DJNZ R1,BACK              ;If R1 not equal to 0 go to BACK
        DJNZ R0,AGAIN             ;If R0 not equal to 0 go to AGAIN
```

RESULT:

Inputs:

x: 0x2000 -- 05h, 02h, 01h, 04h

Output:

x: 0x2000 -- 01h, 02h, 04h, 05h

Exp. No. 13(A)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR DESCENDING ORDER OF A GIVEN NUMBERS

AIM: Write an assembly language program for arranging in ascending/descending order using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM FOR DESCENDING ORDER:

```
        MOV R0,#5           ; INITIALIZE COUNTER 1
AGAIN:  MOV DPTR,#2000H     ; initialize memory pointer
        MOV R1,#4           ; initialize counter 2
BACK:   MOV R2,DPL                  ; save lower byte of memory address
        MOVX A,@DPTR        ; Get the num ber
        MOV B,A             ; Save the number
        INC DPTR            ; Increment the memory pointer
        MOVX A,@DPTR        ; Get the next number
        CJNE A,B,n          ; If not equal check for greater or less
        AJMP SKIP           ; Otherwise go to skip
   n:   JC SKIP             ;If
        MOV DPL,R2          ;Exchange
        MOVX @DPTR,A
        INC DPTR
        MOV A,B
        MOVX @dptr,A
SKIP:   DJNZ R1,BACK            ;If R1 not equal to 0 go to BACK
        DJNZ R0,AGAIN           ;If R0 not equal to 0 go to AGAIN
```

RESULT:

Inputs:

x: 0x2000 -- 05h, 02h, 01h, 04h

Output:

x: 0x2000 -- 05h, 04h, 02h, 01h

Exp. No. 13(B)

ASSEMBLY LANGUAGE PROGRAM FOR DATA TRANSFER

AIM: Write an assembly language program for block move from one address to another address using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM:

```
MOV R0,#20H
MOV R1,#30H
MOV R3,#10H
CLR A
AGAIN:MOV A,@R0
MOV @R1,A
INC R0
INC R1
DJNZ R3,AGAIN
END
```

RESULT:

Inputs:

i: 0x20 -- 01h, 02h, 03h, 04h

Output:

i: 0x30 -- 01h, 02h, 03h, 04h