Terna Engineering College
Computer Engineering Department

Program: Sem V

Course: Microprocessor Lab

Faculty: ARATHI BOYANAPALLI

LAB Manual

PART A

(PART A: TO BE REFERRED BY STUDENTS)

Experiment No. 6

A.1 Aim:
Write an assembly language program to display System time using DOS/BIOS Interrupt.

A.2 Prerequisite:
Basic knowledge of 8086 instruction set and interrupts.

A.3 Outcome:
After successful completion of this experiment, students will be able to -

1. Use appropriate instructions to program microprocessors to perform various tasks.
2. Develop the program in assembly/ mixed language for Intel 8086 processor.
3. Demonstrate the execution and debugging of assembly/ mixed language programs.

A.4 Theory:
Following are two popular dos functions used along with int 21h to get and set the system time -

1. **INT 21/2C - Get Time**

   AH = 2C
   on return:
   CH = hour (0-23)
   CL = minutes (0-59)
   DH = seconds (0-59)
   DL = hundredths (0-99)

- retrieves DOS maintained clock time

2. **INT 21/2D - Set Time**
   AH = 2D
   CH = hour (0-23)
   CL = minutes (0-59)
   DH = seconds (0-59)
   DL = hundredths (0-99)
   on return:
   AL = 00 if time change successful
   = FF if time invalid
   - changes DOS maintained clock
   - DOS version 3.3+ also update CMOS clock where applicable

A.4 Algorithm:

- **I.** Write a procedure which will convert the given binary code into ASCII code.
- **II.** Use 2c function to get system time and call procedure.
- **III.** Display this data by putting data into a buffer.

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the ERP or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no ERP access available)

Roll No. : 50	Name: Amey Thakur
Class: TE-Comps B	Batch: B3
Date of Experiment: 08/09/2020	Date of Submission: 08/09/2020
Grade:	

B.1 Observations and learning:

(Software Code written by a student and output of the program)

- **Input to display System Time -**

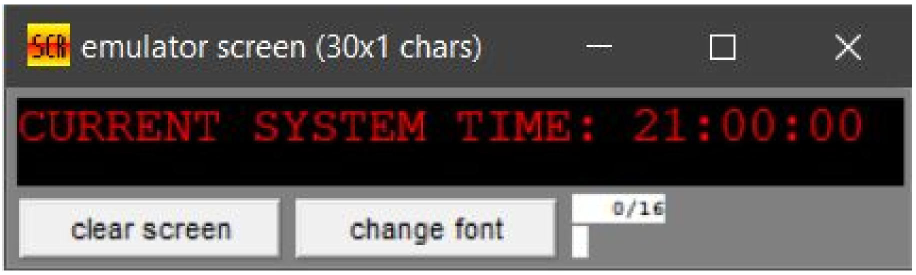
```
.MODEL SMALL
.STACK 100H
DATA SEGMENT
        M  DB  'CURRENT SYSTEM TIME : $'
        T  DB  '00:00:00$'
DATA ENDS

CODE SEGMENT
        MAIN PROC
                MOV AX, @DATA
                MOV DS, AX
                LEA BX, T
                CALL TIME
                LEA DX, M
                MOV AH, 09H
                INT 21H
                LEA DX, T
                MOV AH, 09H
                INT 21H
                MOV AH, 4CH
                INT 21H
        MAIN ENDP
```

```
          TIME PROC
                    PUSH AX
                    PUSH CX
                    MOV AH, 2CH
                    INT 21H
                    MOV AL, CH
                    CALL FILLY
                    MOV [BX], AX
                    MOV AL, CL
                    CALL FILLY
                    MOV [BX+3], AX
                    MOV AL, DH
                    CALL FILLY
                    MOV [BX+6], AX
                    POP CX
                    POP AX
                    RET
          TIME ENDP

          FILLY PROC
                    PUSH DX
                    MOV AH, 0
                    MOV DL, 10
                    DIV DL
                    OR AX, 3030H
                    POP DX
                    RET
          FILLY ENDP
END MAIN
```

- **Output -**



```
CURRENT SYSTEM TIME: 21:00:00
```

B.2 Conclusion:
(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.1)

We successfully learned assembly language programs to display System time using DOS/BIOS Interrupt.

B.5 Question of Curiosity

Q1. Define Procedure and its functioning with an example program in 8086

Ans:

➔ Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size. Procedures are identified by a name. Following this name, the body of the procedure is described which performs a well-defined job. End of the procedure is indicated by a return statement.

➔ Syntax
Following is the syntax to define a procedure –

```
proc_name:
        procedure body
        ...
ret
```

➔ The procedure is called from another function by using the CALL instruction. The CALL instruction should have the name of the called procedure as an argument as shown below –

```
CALL proc_name
```

➔ The called procedure returns the control to the calling procedure by using the RET instruction.

➔ Procedure Definition PROC is a statement used to indicate the beginning of a procedure or subroutine. ENDP indicates the end of the procedure.

➔ The procedure, or subroutine, is an important part of any computer system's architecture. A procedure is a group of instructions that usually performs one task, it is a reusable section of the software that is stored in memory once, but used as often as necessary. This saves memory space and makes it easier to develop software. The only disadvantage of a procedure is that it takes the computer a small amount of time to link to the procedure and return from it. The CALL instruction links to the procedure and the RET instruction returns from the procedure.

Q2. Explain the instruction set for shift and rotate operations with example?

Ans:

➔ Shift instructions move a bit string (or operand treated as a bit string) to the right or left, with excess bits discarded (although one or more bits might be preserved in flags). In arithmetic, shift left or logical shift left zeros are shifted into the low-order bit. In arithmetic shift right the sign bit (most significant bit) is shifted into the high-order bit. In logical shift right zeros are shifted into the high-order bit.

➔ Rotate instructions are similar to shift instructions, except those rotate instructions are circular, with the bits shifted out one end returning on the other end. Rotates can be to the left or right. Rotates can also employ an extended bit for multi-precision rotates.

Shift and Rotate Instructions -

- ➔ Shifting means to move bits right and left inside an operand.
- ➔ All of the Shift and Rotate instructions affect Overflow and Carry Flags.
- ➔ The Shift and Rotate instructions include:
    1. SHR: Shift Right
    2. SAR: Shift Arithmetic Right
    3. SHL: Shift Left
    4. SAL: Shift Arithmetic Left
    5. ROL: Rotate Left
    6. ROR: Rotate Right
    7. RCL: Rotate Carry Left
    8. RCR: Rotate Carry Right

1. SHR: Shift Right

- ➔ The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).
- ➔ Syntax: SHR Register, Bits to be shifted
- ➔ Example: SHR AX, 2

2. SAR: Shift Arithmetic Right

- ➔ The SAR instruction stands for 'Shift Arithmetic Right'. This instruction shifts the mentioned bits in the register to the right side one by one, but instead of inserting the zeroes from the left end, the MSB is restored. The rightmost bit that is being shifted is stored in the Carry Flag (CF).
- ➔ Syntax: SAR Register, Bits to be shifted
- ➔ Example: SAR BX, 5

3. SHL: Shift Left

- ➔ The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).
- ➔ Syntax: SHL Register, Bits to be shifted
- ➔ Example: SHL AX, 2

4. SAL: Shift Arithmetic Left

- ➔ The SAL instruction is an abbreviation for 'Shift Arithmetic Left'. This instruction is the same as SHL.
- ➔ Syntax: SAL Register, Bits to be shifted
- ➔ Example: SAL CL, 2

5. ROL: Rotate Left
➔ The ROL instruction is an abbreviation for ‘Rotate Left’. This instruction rotates the mentioned bits in the register to the left side one by one such that the leftmost bit that is being rotated is again stored as the rightmost bit in the register, and it is also stored in the Carry Flag (CF).
➔ Syntax: ROL Register, Bits to be shifted
➔ Example: ROL AH, 4

6. ROR: Rotate Right
➔ The ROR instruction stands for ‘Rotate Right’. This instruction rotates the mentioned bits in the register to the right side one by one such that the rightmost bit that is being rotated is again stored as the MSB in the register, and it is also stored in the Carry Flag (CF).
➔ Syntax: ROR Register, Bits to be shifted
➔ Example: ROR AH, 4

7. RCL: Rotate Carry Left
➔ This instruction rotates the mentioned bits in the register to the left side one by one such that the leftmost bit that is being rotated is stored in the Carry Flag (CF), and the bit in the CF moved as the LSB in the register.
➔ Syntax: RCL Register, Bits to be shifted
➔ Example: RCL CH, 1

8. RCR: Rotate Carry Right
➔ This instruction rotates the mentioned bits in the register to the right side such that the rightmost bit that is being rotated is stored in the Carry Flag (CF), and the bit in the CF moved as the MSB in the register.
➔ Syntax: RCR Register, Bits to be shifted
➔ Example: RCR BH, 6

Shift and Rotate Applications -
➔ Shift and Rotate instructions are included because they are helpful in certain applications.
➔ These applications includes:
    1. Shifting Multiple Doublewords (for bit-mapped graphics images)
    2. Binary multiplication
    3. Display Binary Bits
    4. Isolating a BitString