Micro processor

Software project

#7: Marathon results project

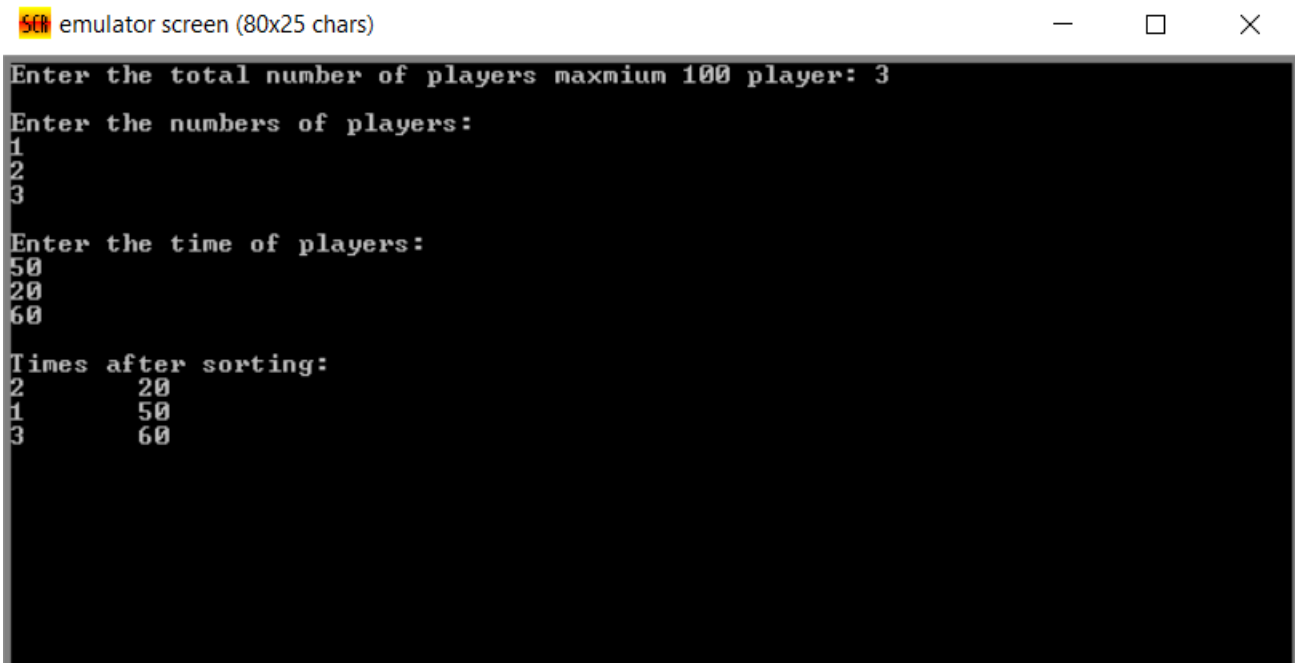
21010881	عمر محمد أحمد شويخ محفوظ
21010947	فرح السيد السيد مصطفى
21010339	آية أحمد سليمان
21011022	مارك زكريا سليمان

Code:

```
include 'emu8086.inc'

JMP START

DATA SEGMENT
        N       DW   1 DUP(?)
    NUMBERS     DB   100 DUP (?)
    TIME        DB   100 DUP (?)
    MSG0        DB 'Enter the total number of players maxmium 100 player : ',0
    MSG1        DB 0DH,0AH,'Enter the numbers of players: ',0AH,0DH,0       ;0D FOR ENTER , 0A FOR NEW LINE
    MSG2        DB 0DH,0AH,'Enter the time of players: ',,0AH,0DH,0
    MSG3        DB 0DH,0AH,'Times after sorting: ',0AH,0DH,0

DATA ENDS

CODE SEGMENT

                ASSUME DS:DATA CS:CODE
START:   MOV AX, DATA
                MOV DS, AX

                DEFINE_SCAN_NUM
                DEFINE_PRINT_STRING
                DEFINE_PRINT_NUM
                DEFINE_PRINT_NUM_UNS


                 LEA  SI,MSG0
                 CALL PRINT_STRING
                 CALL SCAN_NUM        ; scan total number of playes
                 MOV  BYTE PTR[N],CL
                 PRINT 0AH


                LEA SI,MSG1
                CALL PRINT_STRING
                MOV SI, 0


LOOP1:          CALL SCAN_NUM        ;Scan the numbers of plyers one by one and store it in the memory
                MOV NUMBERS[SI],CL
                INC SI
                PRINT 0AH
                PRINT 0DH
                CMP SI,[N]     ; if SI reached to N ...scans all numbers of players and will exit from the loop
                JNE LOOP1


                LEA SI,MSG2
                CALL PRINT_STRING          ; the same for scanning time
                MOV SI, 0

LOOP2:          CALL SCAN_NUM
                MOV TIME[SI],CL
                INC SI
                PRINT 0AH
```

```
059             PRINT 0DH
060             CMP SI,[N]
061             JNE LOOP2
062
063
064
065             ;SORTING using bubble sort method we will put in cl number of players that i want to cmp between them and in
066             ; ch ..number of comparisons
067             ;which every loop we will cmp between the one ml and the ml after it and the biggest number will put in the last ML
068             ; we will repeat this numper of comparisons which stored in ch dep on the number of players that we cmp beteem them
069             ; so at the end whole 2 tables will be arranged
070             MOV CL,BYTE PTR[N]
071             DEC CL
072 OUTER:
073             LEA SI,TIME
074             LEA DI,NUMBERS
075             MOV CH,BYTE PTR[N]
076             DEC CH
077
078
079 INNER:      MOV  AL, [SI]
080             MOV  DL, [DI]
081             INC  SI
082             INC  DI
083             CMP  [SI], AL
084             JA   SKIP           ;if cf=0 num2>num1  num2 != num1 skip and dont exchange as the smallest number in the right place
085             XCHG AL, [SI]
086             MOV  [SI-1], AL
087             XCHG DL, [DI]
088             MOV  [DI-1], DL
089
090 SKIP:
091             DEC CH
092             JNZ INNER   ;if not equal to zero continue comparison between the elements of the table if not dec the cl
093             DEC CL
094             JNZ OUTER
095
096
097
098
099             LEA SI,MSG3
100             CALL PRINT_STRING
101             LEA SI,TIME
102             LEA DI,NUMBERS
103             MOV AH,0
104             MOV CL,BYTE PTR[N]
105             MOV CH,0    ; put in cx the number of players that will loop and print dep on it
106
107 PRINT_TABLE:    MOV AL,[DI]
108             CALL PRINT_NUM_UNS
109             PRINT 09H
110             MOV AL,[SI]
111             CALL PRINT_NUM_UNS

112             PRINT 0AH
113             PRINT 0DH
114             INC SI
115             INC DI
116             LOOP PRINT_TABLE
117
118
119 CODE ENDS
120
121 END START
122
123
124 ret
```

Output:



<u>The instructions which are used:</u>

Instruction	Purpose	How to use
MOV	**transfers bytes or words of data between two registers or between registers and memory in the 8086**	**MOV** destination, source
LEA	**loads a 16- or 32-bit register with the offset address of the data specified by the operand.**	**LEA** register, offset
JMP	**jumps to the location address by the operand**	**JMP** memory_location
CALL	**modify the flow of the program by calling a subroutine and saves a return address on the stack.**	**CALL** procedure_address
INC	**adds 1 to a register or a memory location.**	**INC** operand
JNE/JNZ	**Conditional jump , Jump if not equal or jump if not zero where ZF=0**	**JNZ** destination_address
DEC	**subtracts 1 from any register or memory location.**	**DEC** operand
JA	**Conditional jump , Jump if above where Z = 0 and C = 0**	**JA** destination_address
XCHG	**exchanges the contents of a register(operand_1) with the contents of any other register or memory location(operand_2).**	**XCHG** operand_1, operand_2
RET	**remove the return address from the stack after ending the program.**	**RET**

```
CODE SEGMENT

              ASSUME DS:DATA CS:CODE
START:   MOV  AX, DATA
              MOV DS, AX

              DEFINE_SCAN_NUM
              DEFINE_PRINT_STRING
              DEFINE_PRINT_NUM
              DEFINE_PRINT_NUM_UNS


               LEA   SI,MSG0
               CALL  PRINT_STRING
               CALL  SCAN_NUM        ; scan total number of playes
               MOV   BYTE PTR[N],CL
               PRINT 0AH


              LEA SI,MSG1
              CALL PRINT_STRING
              MOV SI, 0
```

- ✓ The CODE SEGMENT defines a segment for the code instructions.

- ✓ The START label marks the beginning of the code execution.
  The following instructions initialize the data segment and define functions for scanning numbers from user, printing strings messages on screen, and printing numbers entered by the user and the final numbers after sorting.

- ✓ Load the message MSG0 to prompt the user to enter the number of players by using the PRINT_STRING subroutine and stores the value in the N byte variable.

- ✓ The PRINT 0AH is used to take a new line before entering the second input from the user.

- ✓ Load the message MSG1 to prompt the user to enter the number of each player. And then initialize SI pointer with 0.

```
LOOP1:      CALL SCAN_NUM
            MOV NUMBERS[SI],CL
            INC SI
            PRINT 0AH
            PRINT 0DH
            CMP SI,[N]
            JNE LOOP1


            LEA SI,MSG2
            CALL PRINT_STRING
            MOV SI, 0
LOOP2:      CALL SCAN_NUM
            MOV TIME[SI],CL
            INC SI
            PRINT 0AH
            PRINT 0DH
            CMP SI,[N]
            JNE LOOP2
```

- ✓ Enter the (LOOP1) to read the player numbers from the user and store them at the base address NUMBERS. The SCAN_NUM subroutine is called to read a single digit from the user, and the value is stored in the CL register. The CL value is then stored at the NUMBERS base address at the current index (SI) position. The loop continues until the index reaches the value of N.

- ✓ Similarly, for the (LOOP2), the code prompts the user to enter the time for each player using the PRINT_STRING subroutine and stores the values in the TIME base address and increments SI each time until it reaches the value of N.

Some external functions for input/output:

emu8086.inc:

emu8086 provides some libraries that have macro functions for IO, mathematic operation and so on. emu8086.inc is the most common file. You can use the library if you write (`include "emu8086.inc)` at the first line of source file.

Some library functions:

Function	Purpose	How to use
SCAN_NUM	**procedure that gets the multi-digit SIGNED number from the keyboard and stores the result in CX register.**	**DEFINE_SCAN_NUM**
PRINT_STRING	**procedure that prints a string that addressed by DS:SI register**	**DEFINE_PRINT_STRING**
PRINT_NUM	**procedure that prints a decimal number in AX.**	**DEFINE_PRINT_NUM**
PRINT_NUM_UNS	**procedure that prints a signed decimal number in AX**	**DEFINE_PRINT_NUM_UNS**

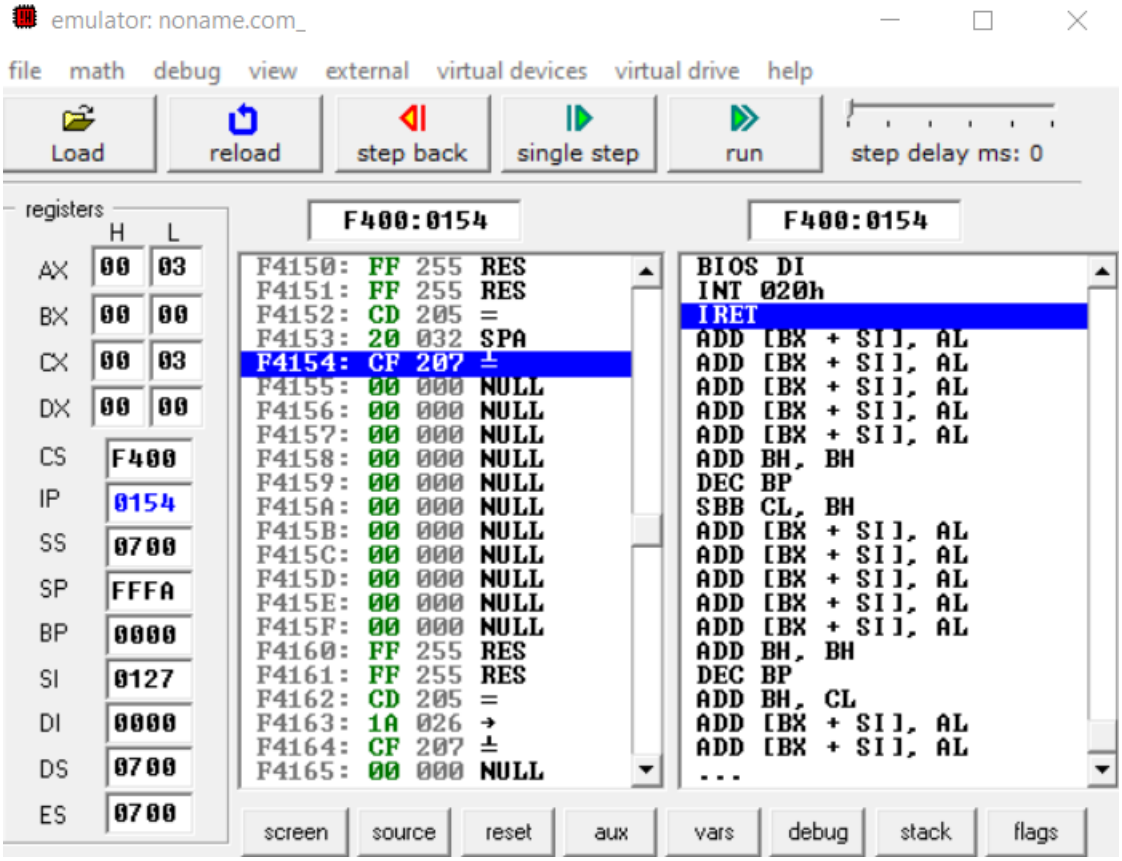
Example of using these functions:

Code:

original source code

```
include 'emu8086.inc'

ORG     100h

LEA     SI, msg1        ; ask for the number
CALL    print_string    ;
CALL    scan_num        ; get number in CX.

MOV     AX, CX          ; copy the number to AX.

; print the following string:
CALL    pthis
DB  13, 10, 'You have entered: ', 0

CALL    print_num       ; print number in AX.

RET                     ; return to operating system.

msg1    DB  'Enter the number: ', 0

DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS    ; required for print_num.
DEFINE_PTHIS

END                     ; directive to stop the compiler.
```

Changes on the registers in the memory:



Output on the screen:

Bubble Sort Algorithm Implementation for Sorting Players' Scores.

```
064
065             ;SORTING using bubble sort method we will put in cl number of players that i want to cmp between them and in
066             ; ch ..number of comparisons
067             ;which every loop we will cmp between the one ml and the ml after it and the biggest number will put in the last ML
068             ; we will repeat this numper of comparisons which stored in ch dep on the number of players that we cmp beteem then
069             ; so at the end whole 2 tables will be arranged
070             MOV CL,BYTE PTR[N]
071             DEC CL
072 OUTER:
073             LEA SI,TIME
074             LEA DI,NUMBERS
075             MOV CH,BYTE PTR[N]
076             DEC CH
077
078
079 INNER:      MOV  AL, [SI]
080             MOV  DL, [DI]
081             INC  SI
082             INC  DI
083             CMP  [SI], AL
084             JA   SKIP           ;if cf=0 num2>num1  num2 != num1 skip and dont exchange as the smallest number in the right place
085             XCHG AL, [SI]
086             MOV  [SI-1], AL
087             XCHG DL, [DI]
088             MOV  [DI-1], DL
089
090 SKIP:
091             DEC CH
092             JNZ INNER   ;if not equal to zero continue comparison between the elements of the table if not dec the cl
093             DEC CL
094             JNZ OUTER
095
096
097
098
099             LEA SI,MSG3
100             CALL PRINT_STRING
101             LEA SI,TIME
102             LEA DI,NUMBERS
103             MOV AH,0
104             MOV CL,BYTE PTR[N]
105             MOV CH,0    ; put in cx the number of players that will loop and print dep on it
```

- by initializing the loop counter CL with the number of players pointed by the address (N) to be compared.
- The outer loop, labeled as "OUTER," is responsible for repeating the sorting process for the number of comparisons (CH) specified.
- The SI register is loaded with the address of the TIME array, and DI is loaded with the address of the NUMBERS array and CH is initialized as number of comparisons.
- The CH register is decremented to reflect the remaining number of comparisons.

• The inner loop, labeled as "INNER," compares adjacent elements in the NUMBERS array.
• The AL register is loaded with the current element from the TIME array, and DL is loaded with the current element from the NUMBERS array.
• SI and DI registers are incremented to point to the next elements in the TIME and NUMBERS arrays, respectively.
• The current element in the TIME array is compared with AL using the CMP instruction.
• If the carry flag (CF) is set (indicating that the second number is greater than the first), the code jumps to the SKIP label. No exchange is made in this case, as the smaller number is already in the correct position.
• If the CF is not set (indicating that the first number is greater than or equal to the second), an exchange is performed using the XCHG instruction. AL is swapped with the current element in the TIME array, and DL is swapped with the current element in the NUMBERS array.
• The SKIP label is reached if no exchange was made, and the code proceeds to decrement the CH register.
• If CH is not zero, the inner loop is repeated by jumping to the INNER label. This continues the comparison and exchange process for the remaining elements in the NUMBERS array.
• Once the inner loop completes (CH reaches zero), the CL register is decremented to reflect that one player (element) is sorted.
• If CL is not zero, the outer loop is repeated by jumping to the OUTER label. This continues the sorting process until all players are sorted.
• After the sorting is complete, the code proceeds to print the sorted list.
• The message "MSG3" is loaded into the SI register, and the PRINT_STRING subroutine is called to display the message.
• The SI and DI registers are loaded with the addresses of the TIME and NUMBERS arrays, respectively.
• The AH register is cleared to prepare for printing the elements.
• The CL register is loaded with the number of players (N), and CH is set to 0, indicating that all players will be printed.

Printing Sorted Players' Scores and Times in table Format:

```
PRINT_TABLE:    MOV AL,[DI]
                CALL PRINT_NUM_UNS
                PRINT 09H
                MOV AL,[SI]
                CALL PRINT_NUM_UNS
                PRINT 0AH
                PRINT 0DH
                INC SI
                INC DI
                LOOP PRINT_TABLE
```

• The AL register is loaded with the value at the memory location pointed to by the DI register. This value represents a player's score or number.
• The PRINT_NUM_UNS subroutine is called to print the value stored in AL as an unsigned number.

- The ASCII character 09H (horizontal tab) is printed using the PRINT instruction to add spacing between the player number and its time .
- The AL register is loaded with the value at the memory location pointed to by the SI register. This value represents a player's time.
- The PRINT_NUM_UNS subroutine is called again to print the value stored in AL as an unsigned number.
- The ASCII characters 0AH (line feed) and 0DH (carriage return) are printed using the PRINT instruction to move to the next line.
- 0DH is used to move the cursor to the beginning of the next line after printing a pair of numbers.
- The SI and DI registers are incremented to point to the next elements in their respective arrays.
- The LOOP instruction is used to repeat the PRINT_TABLE loop until the loop counter (CX) becomes zero.
- The loop counter is decremented automatically by the LOOP instruction.
- Once the loop counter becomes zero, the code exits the loop and continues execution after the PRINT_TABLE loop.