Inspiring Excellence

CSE341 : Microprocessors
Project Name: Tic Tac Toe
Section: 6
Group:8

ID	Name
20301202	Asibur Rahman Marin
21101088	Sreezon Das Gupta
21101298	Anika Islam

Additional Instructions Document

In our project the functions below are the ones that were not taught to us in class so here we have written a short documentation explaining what each of these functions do briefly.

CALL

CALL : In emu8086 assembly language, subroutine or procedure calls are handled by the call function. Control transfer to a subroutine or procedure, typically situated elsewhere in the program, is its intended function. The address of the instruction that follows the call is moved onto the stack and control is transferred to the specified subroutine upon encountering a call instruction.

Upon the completion of its execution, the subroutine customarily transfers control back to the instruction that prompt.

```
;Example Code
.MODEL SMALL
.STACK 100H
.DATA
   msg db "The sum is: $"
   msg1 db " enter the first number: $"
   msg2 db " enter the second number: $"

.CODE
main:
   mov ax, @data
```

```
mov ds, ax

; Display message to enter the first number
mov ah, 09h
lea dx, msg1
int 21h

; Read the first number
mov ah, 01h
int 21h
sub al, '0'  ; Convert ASCII to binary
mov bl, al   ; Store the first number in BL

; Display message to enter the second number
mov ah, 09h
lea dx, msg2
int 21h

; Read the second number
mov ah, 01h
int 21h
sub al, '0'  ; Convert ASCII to binary
mov cl, al   ; Store the second number in CL

; Call subroutine to calculate the sum
call addNumbers

; Display the sum
mov ah, 09h
lea dx, msg
int 21h

; Display the result (stored in AL)
add al, '0'  ; Convert binary to ASCII
mov dl, al
mov ah, 02h
```

```
    int 21h

    ; Exit the program
    mov ah, 4Ch
    int 21h

addNumbers:
    ; Add the numbers (BL and CL) and store the result in AL
    add bl, cl
    mov al, bl

    ; Return to the caller
    ret
end main
```

For example :-
In project code, the CALL function has been used to CALL the statement set_positionArray_grid which sets the position of the input both the players give. In the statement we have declared logics that were necessary for setting the positions. The CALL function has been used to repeat the process until all the positions are filled up.

RET

RET : In EMU8086 assembly language, the RET (return) function is used to return control from a subroutine to the main program. It passes program control back to the instruction after the call instruction that launched the function.

```
;Example Code:

.MODEL SMALL
.STACK 100H

.DATA
   msg1 DB 'Inside the subroutine.', '$'
   msg2 DB 'Back in the main program.', '$'

.CODE
MAIN PROC
   MOV AX, @DATA
   MOV DS, AX

   ; Call the subroutine
   CALL Subroutine

   MOV AH, 09H
   LEA DX, msg2
   INT 21H

MAIN ENDP

Subroutine PROC

   MOV AH, 09H
   LEA DX, msg1
   INT 21H
```

```
   RET                 ; Return control to the main program
Subroutine ENDP

END MAIN
```

For example:
In our project, we have used a statement named check_diagonal_win where we are checking if any of the players have won diagonally. The statement is only a portion of the code. After its execution the control must be returned to the main function. The RET function serves the purpose of returning the control to main function.

XOR

XOR : In EMU8086 assembly language, the XOR function performs a bitwise XOR operation between two operands.

The XOR operation sets each bit of the destination operand to 1 if the corresponding bits of the source and destination operands are different; otherwise, it sets the bit to 0.

Example Code:

```
.MODEL SMALL
.STACK 100H

.DATA
    num1 DW 0010b  ; Binary number 0010 (2 in decimal)
    num2 DW 1010b  ; Binary number 1010 (10 in decimal)


.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    MOV BX, num1   ; Load num1 into BX
    XOR BX, num2   ; Perform XOR operation with num2


    MOV AH, 4CH
    INT 21H
MAIN ENDP

END MAIN
```

For example:
In our project, We have used a statement named swap_players where we have used the XOR function to our advantage and ensure that players are swapped properly. As we know the XOR function compares between the two operands given to it and after comparing the values it either sends a 0 or 1 signal. In our case, from the Data Segment we first fetch the value for the current player who just entered his input. By Comparing that value with 3 using the XOR function we find that the number of the current player is not equal to 3. Then the players are swapped.

ADDRESSING MODES

Addressing modes such as :-

```
MOV [BX],SI
; register [direct] addressing
; The data stored in SI register is stored in the memory location at the memory address found using the BX as offset of DS.
; WRITE operation
```

```
MOV SI,[BX]
; register [direct] addressing
; The data stored in the memory location at the memory address found using the BX as offset of DS is stored in the SI register.
; READ operation
```

For example:
In our project, Turns for each player is stored in the positionArray and accessed by storing the position in register in the memory and data fetched for win declare from that memory.

AH,15 and AH,0

```
MOV AH,15
; set the outputs on the screen like a video game
```

```
MOV AH,0
; remove the outputs from the screen
```

For example,
In our project, the screen is viewed for all outputs unless the condition, call_screen, is called. The condition is called for game_over. For game_over, all the outputs previously viewed disappear using AH,0 and the final result is only displayed using AH,15