CO215, Assignment 9

Roll number: CSB23018

Name: Swapnil Kalita

Session: Spring, Year:2025

Objective:

1. To learn working with array of structures using based-indexed addressing mode;
2. To learn working with multidimensional arrays;
3. To learn the use of division instructions in µP 8086.

Exercises:

1. Create a file of the 8086 program given below
2. **Execute the program with Debugger by single-stepping.**

Ans-

```
.MODEL SMALL

.STACK 100H

.DATA


No_Records EQU 5

No_Fields EQU 7

Bytes_Record EQU No_Fields * 2

Subjects db 3

Marks_Records DW 01, 22, 66, 54, 75, 00, 00

DW 02, 21, 70, 67, 77, 00, 00

DW 03, 21, 55, 60, 65, 00, 00

DW 04, 22, 75, 66, 83, 00, 00

DW 05, 21, 49, 59, 69, 00, 00 ;field 1: roll, field 2: age, field 3-5: marks,

; field 6: total, field 7: average


Subject_Av DW ?,?,?,?,?,?,?

Output_Msg1 DB 'Roll No $'

Output_Msg2 DB ': Total- $'

Output_Msg3 DB ' Average- $'
```

```
nwln db 10, 13, '$'
ten db 10
hun db 100

.CODE

main PROC
.STARTUP

mov cx, No_Records
sub bx, bx

Repeat1:
sub ax, ax
add ax, Marks_Records[bx+4]
add ax, Marks_Records[bx+6]
add ax, Marks_Records[bx+8]
mov Marks_Records[bx+10], ax
div Subjects
sub ah, ah
mov Marks_Records[bx+12], ax
add bx, Bytes_Record
loop Repeat1

mov cx, No_Records
sub bx, bx
sub dx, dx

Repeat2:
lea dx, nwln
mov ah, 09h
```

```
int 21h

lea dx, Output_Msg1
mov ah, 09h
int 21h

push ax
push bx
push cx
push dx
push sp
push si
 ; save all register values in stack so that they can be restored after printing
; integer is done

mov ax, Marks_Records[bx] ; to print roll
div ten
mov bx, ax ; saving value of ax in bx in order to avoid being overwritten
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov dl, bh ; bx = ax thus bh = ah that is bh = remainder of the div operation
add dl, 30h
mov ah, 02h
int 21h

pop si
pop sp
pop dx
pop cx
```

```
pop bx
pop ax ; restore all register values

lea dx, Output_Msg2
mov ah, 09h
int 21h

push ax
push bx
push cx
push dx
push sp
push si

mov ax, Marks_Records[bx+10] ;to print total
div hun
mov bx, ax
mov dl, al
add dl, 30h

mov ah, 02h
int 21h
mov ah, 0
mov al, bh
div ten
mov bx, ax
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov dl, bh
```

```
add dl, 30h
mov ah, 02h
int 21h

pop si
pop sp
pop dx
pop cx
pop bx
pop ax ; restore all register values

lea dx, Output_Msg3
mov ah, 09h
int 21h

push ax
push bx
push cx
push dx
push sp
push si

mov ax, Marks_Records[bx+12] ; to print average
div ten
mov bx, ax
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov dl, bh
add dl, 30h
```

```
mov ah, 02h
int 21h

pop si
pop sp
pop dx
pop cx
pop bx
pop ax ; restore all register values

lea dx, nwln
mov ah, 09h
int 21h

add bx, Bytes_Record ; to point bx to next record

dec cx             ; Decrement counter
cmp cx, 0          ; Compare with zero
je Exit_Loop       ; If equal to zero, exit loop
jmp Repeat2        ; Otherwise, jump back to Repeat2 (unconditional far jump)

Exit_Loop:
; .EXIT
mov ah, 4ch
mov al, 0
int 21h

main ENDP

END main
```

3. Add a procedure to the program to read in the Marks_Records from the keyboard and repeat the exercises.

Ans-

```
.MODEL SMALL

.STACK 100H

.DATA


No_Records    EQU 5

No_Fields     EQU 7

Bytes_Record  EQU No_Fields * 2

Subjects      DB 3

Marks_Records DW 01, 22, 66, 54, 75, 00, 00

              DW 02, 21, 70, 67, 77, 00, 00

              DW 03, 21, 55, 60, 65, 00, 00

              DW 04, 22, 75, 66, 83, 00, 00

              DW 05, 21, 49, 59, 69, 00, 00


Output_Msg1   DB 'Roll No $'

Output_Msg2   DB ':  Total- $'

Output_Msg3   DB '  Average- $'

Input_Msg1    DB 'Enter student data (Roll Age Mark1 Mark2 Mark3): $'

Input_Space   DB ' $'

nwln          DB 10, 13, '$'

ten           DB 10

hun           DB 100


.CODE


main PROC

   .STARTUP
```

```
    CALL InputRecords

    mov cx, No_Records
    sub bx, bx

Repeat1:
    sub ax, ax
    add ax, Marks_Records[bx+4]
    add ax, Marks_Records[bx+6]
    add ax, Marks_Records[bx+8]
    mov Marks_Records[bx+10], ax
    div Subjects
    sub ah,ah
    mov Marks_Records[bx+12], ax
    add bx, Bytes_Record
    loop Repeat1

    mov cx, No_Records
    sub bx, bx
    sub dx, dx

Repeat2:
    lea dx, nwln
    mov ah, 09h
    int 21h

    lea dx, Output_Msg1
    mov ah, 09h
    int 21h

    push ax
```

```
push bx
push cx
push dx
push sp
push si
mov ax, Marks_Records[bx]
div ten
mov bx, ax
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov dl, bh
add dl, 30h
mov ah, 02h
int 21h

pop si
pop sp
pop dx
pop cx
pop bx
pop ax

lea dx, Output_Msg2
mov ah, 09h
int 21h

push ax
push bx
push cx
```

```
push dx
push sp
push si

mov ax, Marks_Records[bx+10]
div hun
mov bx, ax
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov ah, 0
mov al, bh
div ten

mov bx, ax
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov dl, bh
add dl, 30h
mov ah, 02h
int 21h

pop si
pop sp
pop dx
pop cx
pop bx
pop ax
```

```
lea dx, Output_Msg3
mov ah, 09h
int 21h

push ax
push bx
push cx
push dx
push sp
push si

mov ax, Marks_Records[bx+12]
div ten
mov bx, ax
mov dl, al
add dl, 30h
mov ah, 02h
int 21h
mov dl, bh
add dl, 30h
mov ah, 02h
int 21h

pop si
pop sp
pop dx
pop cx
pop bx
pop ax
```

```
   lea dx, nwln
   mov ah, 09h
   int 21h

   add bx, Bytes_Record


   dec cx
   cmp cx,0
   je Exit_loop
   jmp Repeat2

Exit_loop:
   mov ah, 4ch
   mov al, 0
   int 21h

main ENDP

InputRecords PROC
   PUSH AX
   PUSH BX
   PUSH CX
   PUSH DX
   PUSH SI

   MOV CX, No_Records
   MOV BX, 0

InputLoop:
   LEA DX, nwln
   MOV AH, 09H
```

```
INT 21H

LEA DX, Input_Msg1
MOV AH, 09H
INT 21H

CALL InputNumber
MOV Marks_Records[BX], AX

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

CALL InputNumber
MOV Marks_Records[BX+2], AX

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

CALL InputNumber
MOV Marks_Records[BX+4], AX

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

CALL InputNumber
MOV Marks_Records[BX+6], AX

LEA DX, Input_Space
```

```
    MOV AH, 09H
    INT 21H

    CALL InputNumber
    MOV Marks_Records[BX+8], AX

    MOV WORD PTR Marks_Records[BX+10], 0
    MOV WORD PTR Marks_Records[BX+12], 0

    ADD BX, Bytes_Record
    LOOP InputLoop

    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
InputRecords ENDP

InputNumber PROC
    PUSH BX
    PUSH DX

    MOV AH, 01H
    INT 21H
    SUB AL, '0'
    MOV BL, 10
    MUL BL
    MOV BH, AL
```

```
    MOV AH, 01H
    INT 21H
    SUB AL, '0'
    ADD BH, AL
    MOV AX, 0
    MOV AL, BH

    POP DX
    POP BX
    RET
InputNumber ENDP

END main
```

4. Prepare a report comprising the objectives, exercises carried out, observations, learning outcome of the exercise.

Observations:

Variables:

Name	**No of Bytes**	**Location**	**Initial Value**	**Step No After Which Value Does Not Change**	**Final Value**
No_Records	0 (constant)	Code (EQU)	N/A	N/A	N/A
No_Fields	0 (constant)	Code (EQU)	N/A	N/A	N/A
Bytes_Record	0 (constant)	Code (EQU)	N/A	N/A	N/A
Marks_Records	70	Data segment	Student data with 0s in total and average	After Repeat1 loop	Total and Average fields computed
Subjects	1	Data segment	3	Constant	3
Output_Msg1	9	Data segment	'Roll No $'	Constant	'Roll No $'

Output_Msg2	12	Data segment	': Total- $'	Constant	': Total- $'
Output_Msg3	13	Data segment	' Average- $'	Constant	' Average- $'
Input_Msg1	44	Data segment	'Enter student data (Roll Age Mark1 Mark2 Mark3): $'	Constant	Same
Input_Space	2	Data segment	' $'	Constant	' $'
nwln	3	Data segment	LF (10), CR (13), '$'	Constant	Same
ten	1	Data segment	10	Constant	10
hun	1	Data segment	100	Constant	100

Learning Outcome:

1. Instruction structures in µP 8086 ALP for based-indexed addressing:

Based-indexed addressing is a powerful addressing mode in the Intel 8086 microprocessor that combines two registers to calculate an effective address. This addressing mode offers flexibility for accessing data structures such as arrays and tables.

Key components:

- Combines a base register (BX or BP) with an index register (SI or DI)
- Optional displacement value can be added
- Formula: Effective Address = Base Register + Index Register + Displacement

Common structures:

- MOV AX, [BX+SI] - Access memory at address BX+SI
- MOV [BX+DI+5], CX - Store CX at address BX+DI+5
- ADD DX, [BP+SI+10] - Add value at BP+SI+10 to DX

Applications:

- Array access with BX as base and SI/DI as index

- Stack frame references using BP as base pointer
- Table lookups with multiple dimensions
- Efficient traversal of complex data structures

2. Use of Various Registers in the Division Instruction Operation in µP 8086

The 8086 division instruction (DIV/IDIV) involves multiple registers working together to perform division operations. Understanding register roles is crucial for correct implementation.

Dividend registers:

- For 8-bit division: AX contains the 16-bit dividend
- For 16-bit division: DX

pair contains the 32-bit dividend (DX holds higher bits)

Divisor operand:

- Can be a register or memory location
- Size determines division type (8-bit or 16-bit)

Result registers:

- Quotient: Stored in AL (8-bit division) or AX (16-bit division)
- Remainder: Stored in AH (8-bit division) or DX (16-bit division)

Special considerations:

- Division by zero causes a Type 0 interrupt
- Quotient overflow (result too large for destination) causes Type 0 interrupt
- CX register often used as loop counter for repeated division operations
- BX commonly used for addressing when divisor is in memory

Example operation: For 16-bit division with DX

/ operand:

1. DX must be properly set before division (often cleared with SUB DX,DX if high bits are zero)
2. After DIV, AX contains quotient and DX contains remainder