Course Name:	**Microprocessors and Microcontrollers Laboratory**	**Semester:**	**IV**
Date of Performance:	**11 / 02 / 2024**	**Batch No:**	**A - 2**
Faculty Name:	**Kirti Sawlani**	**Roll No:**	**16014022050**
Faculty Sign & Date:		**Grade / Marks:**	**___ / 25**

Experiment No.: 3
Title: Programmable Delays

Aim and Objective of the Experiment:

Aim: Write an 8086 based ALP to Generate delay with software instructions using procedures.

Case study:

1. Display two strings without delay.
2. Display two strings on monitor with some specific delay.

Objectives:

1. To study procedures and macros.
2. To study timing calculations of instructions.
3. To study DOS interrupts.

This experiment covers:

1. Data transfer instructions
2. DOS interrupts for displaying strings and characters on monitor.
3. Delay calculations

COs to be achieved:

CO2: Develop 8086 based assembly language programs.

Stepwise-Procedure:

1. Open EMU8086 and write your ASM code in the empty workspace.
2. Click on emulate button and it should open the emulator window.
3. You can run the code using single step execution and monitor the internal registers / flags.

Algorithm / Flowchart:

11/02/24

MPMC experiment 3 : Programmable Delay

→ AIM : Write an 8086 based ALP to display two strings "Hello" & "KJSCE" on monitor with a software delay of 1sec.
Assume process or frequency as 20MHz.

→ Algorithm :

1. Display string 1.
2. Delay (time gap)
3. Display string 2 on new line (new line = 10 = /n)
4. Delay (time gap)
5. Repeat 1-4.

- How to initialise strings in data segment,

Str1 db "Hello", 10, 13, '$'

- Str1 : array name
- db : datatype
- 10 : newline
- 13 : carriage return
- '$' : end of string → if not included, garbage values after the string.

- How to print string on monitor ALP,

mov ah, 09
int 21h
lea dx, str1

displaying string on monitor
have to give address in dx register

- Types of functions,
Functions are known as procedures.
  - near proc. : procedure written in same code segment.
  - far proc. : procedure written in separate code segment

Assembly Language Program:

1. **Displaying two strings without delay:**

EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.

exp3_printingString.exe_ -- emu8086 assembler version: 4.08

[11-03-2024 -- 19:16:25]

```
================================================================
[LINE]     LOC: MACHINE CODE                    SOURCE
================================================================

[   1]       :
[   2]       :                              ; MPMC Experiment 3 - Programmable Delay
[   3]       :                              ; A-2 / 16014022050
[   4]       :
[   5]       :
[   6]       :                              ; Printing two strings
[   7]       :
[   8]       :                              ; Initialize data and stack segments
[   9]       :
[  10]       :                              data segment
[  11]    0000: 48 65 6C 6C 6F 0A 0D 24          str1 db "Hello", 10, 13, "$"     ; Define
"Hello" string
[  12]    0008: 4B 4A 53 43 45 0A 0D 24          str2 db "KJSCE", 10, 13, "$"     ; Define
"KJSCE" string
[  13]       :                              data ends
[  14]       :
[  15]       :                              stack segment
[  16]    0010: 00 00                            20 dw dup(0)                ; Define stack with 20
words
[  17]    0012:                                stack_top label word          ; Stack top label
[  18]       :                              stack ends
[  19]       :
[  20]       :                              code segment
[  21]       :                                assume ds: data, cs: code, ss: stack ; Assume segment
registers
[  22]       :
```

```
[ 23]   0020:                   start:
[ 24]   0020: B8 00 00            mov ax, data          ; Set data segment
[ 25]   0023: 8E D8               mov ds, ax
[ 26]   0025: B8 01 00            mov ax, stack         ; Set stack segment
[ 27]   0028: 8E D0               mov ss, ax
[ 28]   002A: B3 02               mov bl, 02            ; Unused register
[ 29]      :
[ 30]   002C:                   repeat:
[ 31]   002C: B4 09               mov ah, 09            ; Display string
[ 32]   002E: BA 00 00            lea dx, str1          ; Load string address
[ 33]   0031: CD 21               int 21h               ; DOS interrupt for string
display
[ 34]   0033: E8 0C 00            call delay            ; Call delay subroutine
[ 35]   0036: B4 09               mov ah, 09            ; Display string
[ 36]   0038: BA 08 00            lea dx, str2          ; Load string address
[ 37]   003B: CD 21               int 21h               ; DOS interrupt for string
display
[ 38]   003D: E8 02 00            call delay            ; Call delay subroutine
[ 39]   0040: EB EA               jmp repeat            ; Repeat loop
[ 40]      :
[ 41]   0042:                   delay proc near
[ 42]   0042: B9 E8 03            mov cx, 1000          ; Set delay count
[ 43]      :
[ 44]   0045:                   back:
[ 45]   0045: E2 FE               loop back             ; Loop for delay
[ 46]   0047: C3                  ret                   ; Return from delay subroutine
[ 47]      :                    delay endp             ; End of delay subroutine
[ 48]      :
[ 49]   0048: B4 4C               mov ah, 4ch           ; Exit program
[ 50]   004A: CD 21               int 21h               ; DOS interrupt
[ 51]      :                    code ends              ; End of code segment
[ 52]      :                    end start              ; End of program
[ 53]      :
```

==========

EXE HEADER - bytes from 0000 to 01FF inclusive.

0000: 4D - exe signature (M)
0001: 5A - exe signature (Z)

0002: 4C - bytes on last page (l.byte)
0003: 00 - bytes on last page (h.byte)
0004: 02 - 512 byte pages in file (l.byte)
0005: 00 - 512 byte pages in file (h.byte)
0006: 02 - relocations (l.byte)
0007: 00 - relocations (h.byte)
0008: 20 - paragraphs in header (l.byte)
0009: 00 - paragraphs in header (h.byte)
000A: 00 - minimum memory (l.byte)
000B: 00 - minimum memory (h.byte)
000C: FF - maximum memory (l.byte)
000D: FF - maximum memory (h.byte)
000E: 01 - SS - stack segment (l.byte)
000F: 00 - SS - stack segment (h.byte)
0010: 02 - SP - stack pointer (l.byte)
0011: 00 - SP - stack pointer (h.byte)
0012: D0 - check sum (l.byte)
0013: 5E - check sum (h.byte)
0014: 00 - IP - instruction pointer (l.byte)
0015: 00 - IP - instruction pointer (h.byte)
0016: 02 - CS - code segment (l.byte)
0017: 00 - CS - code segment (h.byte)
0018: 1E - relocation table adress (l.byte)
0019: 00 - relocation table adress (h.byte)
001A: 00 - overlay number (l.byte)
001B: 00 - overlay number (h.byte)
001C: 01 - signature (l.byte)
001D: 00 - signature (h.byte)
001E: 01 - relocation table - offset inside segment (l.byte)
001F: 00 - relocation table - offset inside segment (h.byte)
0020: 02 - relocation table - segment anchor (l.byte)
0021: 00 - relocation table - segment anchor (h.byte)
0022: 06 - relocation table - offset inside segment (l.byte)
0023: 00 - relocation table - offset inside segment (h.byte)
0024: 02 - relocation table - segment anchor (l.byte)
0025: 00 - relocation table - segment anchor (h.byte)
0026 to 01FF - reserved relocation area (00)

==========

2. Displaying two strings with some specific delay:

EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.

```
noname.exe_ -- emu8086 assembler version: 4.08
[ 11-03-2024  --  19:26:05 ]
================================================================
[LINE]     LOC: MACHINE CODE                    SOURCE
================================================================

[   1]       :
[   2]       :                              ; MPMC Experiment 3 - Programmable Delay
[   3]       :                              ; A-2 / 16014022050
[   4]       :
[   5]       :                              ; Displaying two strings with some specific delay
[   6]       :
[   7]       :                              data segment                    ; Define data segment
[   8]    0000: 48 65 6C 6C 6F 0A 0D 24     str1 db "Hello",10,13,'$'       ; Define string
1 with newline and carriage return characters
[   9]    0008: 4B 4A 53 43 45 0A 0D 24     str2 db "KJSCE",10,13,'$'       ; Define string
2 with newline and carriage return characters
[  10]       :                              data ends                       ; End data segment
[  11]       :
[  12]       :                              stack segment                   ; Define stack segment
[  13]    0010: 00 00 00 00 00 00 00 00 00 00 00 00   dw 20 dup(0)          ; Define stack
with 20 words initialized to 0
                00 00 00 00 00 00 00 00 00 00 00 00
                00 00 00 00 00 00 00 00 00 00 00 00
                00 00 00 00
[  14]    0038:                             stack_top label word            ; Define stack top label
[  15]       :                              stack ends                      ; End stack segment
[  16]       :
[  17]       :                              code segment                    ; Define code segment
[  18]       :                              assume ds:data, cs:code, ss: stack ; Set segment registers
[  19]    0040:                             start:                          ; Start of program
[  20]    0040: B8 00 00                       mov ax,data                  ; Load data segment into
AX
[  21]    0043: 8E D8                          mov ds,ax                    ; Move data segment into
DS
[  22]    0045: B8 01 00                       mov ax,stack                 ; Load stack segment into
```

```
AX
[ 23]   0048: 8E D0                mov ss,ax              ; Move stack segment into
SS
[ 24]   004A:                    repeat:                ; Loop label
[ 25]   004A: B4 09                mov ah, 09             ; Set AH to 09 (for string
output)
[ 26]   004C: BA 00 00             lea dx,str1            ; Load address of string 1
into DX
[ 27]   004F: CD 21                int 21h                ; Print string 1
[ 28]   0051: E8 0C 00             call delay             ; Call delay subroutine
[ 29]   0054: B4 09                mov ah,09              ; Set AH to 09 (for string
output)
[ 30]   0056: BA 08 00             lea dx,str2            ; Load address of string 2
into DX
[ 31]   0059: CD 21                int 21h                ; Print string 2
[ 32]   005B: E8 02 00             call delay             ; Call delay subroutine
[ 33]   005E: EB EA                jmp repeat             ; Jump to repeat label to
continue loop
[ 34]       :
[ 35]   0060:                    delay proc near          ; Define delay subroutine
[ 36]   0060: B9 E8 03             mov cx,1000            ; Load CX with 1000
[ 37]   0063:                    back:                  ; Loop label
[ 38]   0063: E2 FE                loop back              ; Decrement CX and loop
if CX is not zero
[ 39]   0065: C3                   ret                    ; Return from subroutine
[ 40]       :                    delay endp             ; End delay subroutine
[ 41]       :
[ 42]   0066: B4 4C                mov ah,4ch             ; Set AH to 4C (for
program termination)
[ 43]   0068: CD 21                int 21h                ; Call DOS interrupt
[ 44]       :                    code ends              ; End code segment
[ 45]       :                    end start              ; End program
[ 46]       :
[ 47]       :
[ 48]       :
[ 49]       :
```

==========

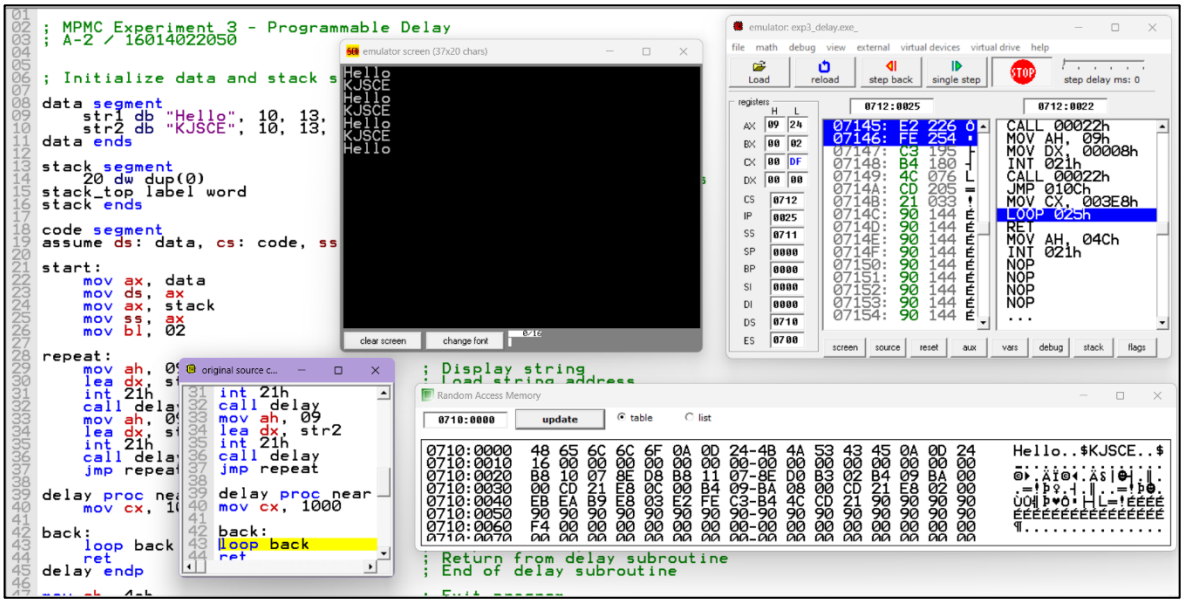
EXE HEADER - bytes from 0000 to 01FF inclusive.
0000: 4D - exe signature (M)

0001: 5A - exe signature (Z)
0002: 6A - bytes on last page (l.byte)
0003: 00 - bytes on last page (h.byte)
0004: 02 - 512 byte pages in file (l.byte)
0005: 00 - 512 byte pages in file (h.byte)
0006: 02 - relocations (l.byte)
0007: 00 - relocations (h.byte)
0008: 20 - paragraphs in header (l.byte)
0009: 00 - paragraphs in header (h.byte)
000A: 00 - minimum memory (l.byte)
000B: 00 - minimum memory (h.byte)
000C: FF - maximum memory (l.byte)
000D: FF - maximum memory (h.byte)
000E: 01 - SS - stack segment (l.byte)
000F: 00 - SS - stack segment (h.byte)
0010: 28 - SP - stack pointer (l.byte)
0011: 00 - SP - stack pointer (h.byte)
0012: 39 - check sum (l.byte)
0013: 61 - check sum (h.byte)
0014: 00 - IP - instruction pointer (l.byte)
0015: 00 - IP - instruction pointer (h.byte)
0016: 04 - CS - code segment (l.byte)
0017: 00 - CS - code segment (h.byte)
0018: 1E - relocation table adress (l.byte)
0019: 00 - relocation table adress (h.byte)
001A: 00 - overlay number (l.byte)
001B: 00 - overlay number (h.byte)
001C: 01 - signature (l.byte)
001D: 00 - signature (h.byte)
001E: 01 - relocation table - offset inside segment (l.byte)
001F: 00 - relocation table - offset inside segment (h.byte)
0020: 04 - relocation table - segment anchor (l.byte)
0021: 00 - relocation table - segment anchor (h.byte)
0022: 06 - relocation table - offset inside segment (l.byte)
0023: 00 - relocation table - offset inside segment (h.byte)
0024: 04 - relocation table - segment anchor (l.byte)
0025: 00 - relocation table - segment anchor (h.byte)
0026 to 01FF - reserved relocation area (00)

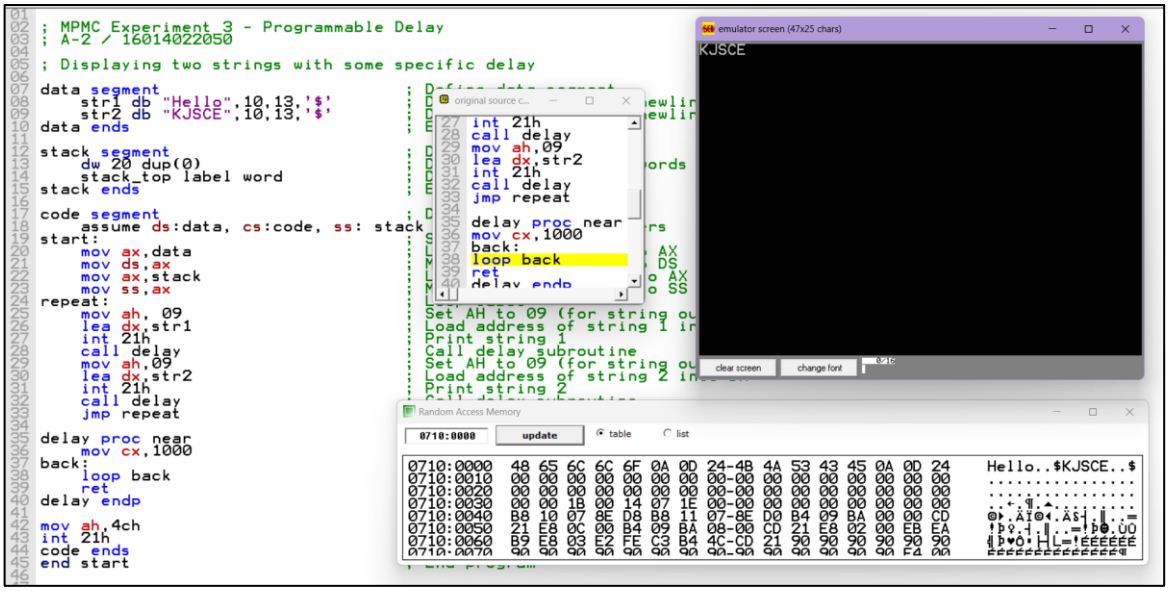
==========

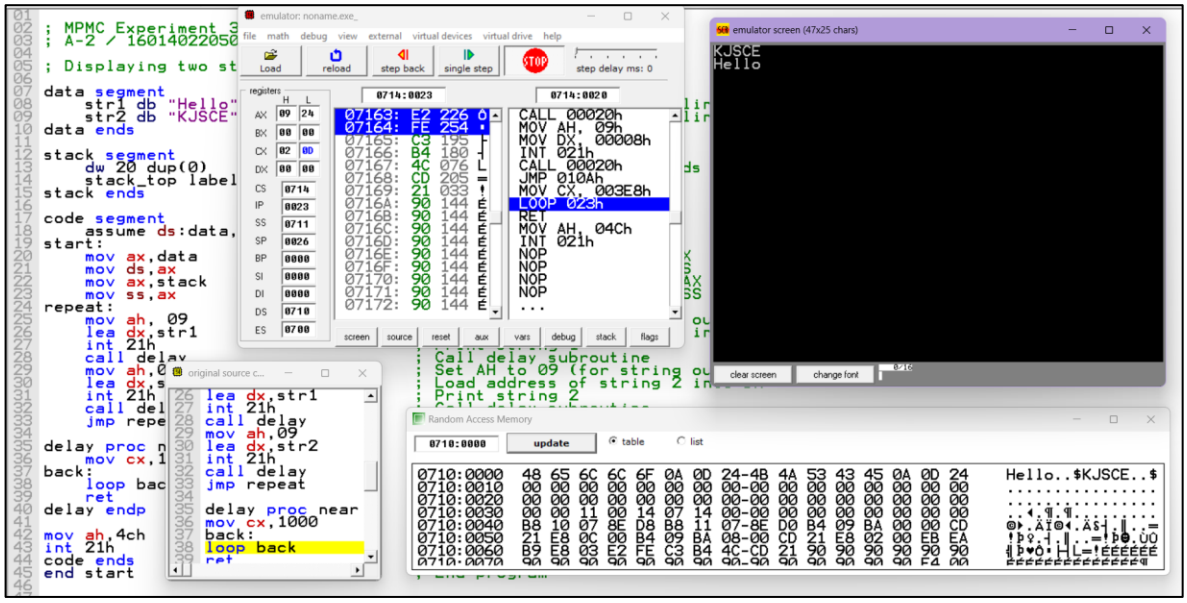
Output Screenshots:

1. Displaying two strings without delay:



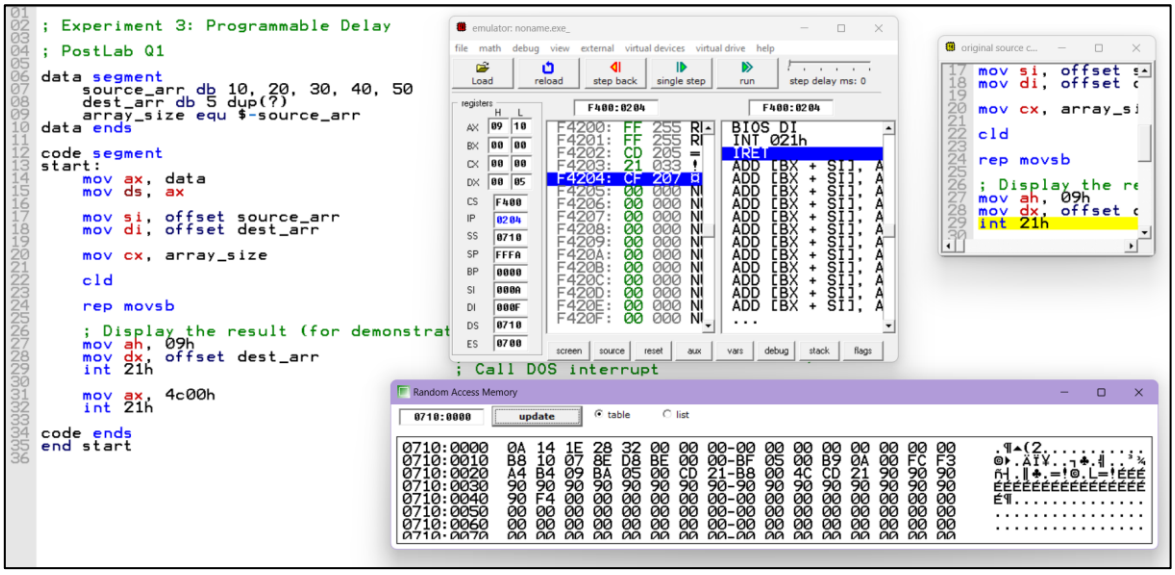
2. Display two strings on monitor with some specific delay:
https://drive.google.com/file/d/1G4yQvRy7ETNjTsblb6xT5bw4k_iLkk3y/view?usp=sharing





Post Lab Subjective/Objective type Questions:

1. **Write an 8086 based ALP of block transfer.**



EMU8086 GENERATED LISTING. MACHINE CODE <- SOURCE.

noname.exe_ -- emu8086 assembler version: 4.08

[11-03-2024 -- 19:47:49]

==========

```
[LINE]    LOC: MACHINE CODE                SOURCE
============================================================


[  1]       :
[  2]       :                          ; Experiment 3: Programmable Delay
[  3]       :
[  4]       :                          ; PostLab Q1
[  5]       :
[  6]       :                          data segment
[  7]    0000: 0A 14 1E 28 32                source_arr db 10, 20, 30, 40, 50    ; Source array
[  8]    0005: 00 00 00 00 00                dest_arr db 5 dup(?)                ; Destination array
[  9]    000A:                            array_size equ $-source_arr         ; Calculate array size
[ 10]       :                           data ends
[ 11]       :
[ 12]       :                          code segment
[ 13]    0010:                             start:
[ 14]    0010: B8 00 00                        mov ax, data                    ; Load data segment
address to AX
[ 15]    0013: 8E D8                           mov ds, ax                      ; Move data segment
address to DS
[ 16]       :
[ 17]    0015: BE 00 00                      mov si, offset source_arr         ; Point SI to source
array
[ 18]    0018: BF 05 00                          mov di, offset dest_arr             ; Point DI to
destination array
[ 19]       :
[ 20]    001B: B9 0A 00                        mov cx, array_size                ; Load array size to
CX
[ 21]       :
[ 22]    001E: FC                              cld                             ; Clear direction flag for
forward movement
[ 23]       :
[ 24]    001F: F3 A4                           rep movsb                       ; Block transfer from
source to destination
[ 25]       :
[ 26]       :                          ; Display the result (for demonstration)
[ 27]    0021: B4 09                         mov ah, 09h                       ; Print function
[ 28]    0023: BA 05 00                        mov dx, offset dest_arr           ; Load address of
destination array
[ 29]    0026: CD 21                         int 21h                         ; Call DOS interrupt
[ 30]       :
[ 31]    0028: B8 00 4C                        mov ax, 4c00h                     ; Exit program
[ 32]    002B: CD 21                         int 21h                         ; Call DOS interrupt
[ 33]       :
[ 34]       :                          code ends
```

```
[ 35]     :                    end start
[ 36]     :
[ 37]     :
```

==========

EXE HEADER - bytes from 0000 to 01FF inclusive.

0000: 4D - exe signature (M)
0001: 5A - exe signature (Z)
0002: 2D - bytes on last page (l.byte)
0003: 00 - bytes on last page (h.byte)
0004: 02 - 512 byte pages in file (l.byte)
0005: 00 - 512 byte pages in file (h.byte)
0006: 01 - relocations (l.byte)
0007: 00 - relocations (h.byte)
0008: 20 - paragraphs in header (l.byte)
0009: 00 - paragraphs in header (h.byte)
000A: 00 - minimum memory (l.byte)
000B: 00 - minimum memory (h.byte)
000C: FF - maximum memory (l.byte)
000D: FF - maximum memory (h.byte)
000E: 00 - SS - stack segment (l.byte)
000F: 00 - SS - stack segment (h.byte)
0010: 00 - SP - stack pointer (l.byte)
0011: 00 - SP - stack pointer (h.byte)
0012: EE - check sum (l.byte)
0013: 09 - check sum (h.byte)
0014: 00 - IP - instruction pointer (l.byte)
0015: 00 - IP - instruction pointer (h.byte)
0016: 01 - CS - code segment (l.byte)
0017: 00 - CS - code segment (h.byte)
0018: 1E - relocation table adress (l.byte)
0019: 00 - relocation table adress (h.byte)
001A: 00 - overlay number (l.byte)
001B: 00 - overlay number (h.byte)
001C: 01 - signature (l.byte)
001D: 00 - signature (h.byte)
001E: 01 - relocation table - offset inside segment (l.byte)
001F: 00 - relocation table - offset inside segment (h.byte)
0020: 01 - relocation table - segment anchor (l.byte)
0021: 00 - relocation table - segment anchor (h.byte)
0022 to 01FF - reserved relocation area (00)

==========

2. Difference between hardware and software delay.

Hardware Delay: Hardware delays are implemented using hardware components like timers or counters. These delays are precise and independent of the execution of the main program. They are often used in real-time systems where precise timing is critical. Hardware delays are not affected by changes in the execution speed of the processor or other software activities.

Software Delay: Software delays are implemented using software loops or software timers. These delays rely on the execution time of instructions or loops in the program. Software delays are less precise compared to hardware delays as they are affected by factors such as processor speed, other tasks running on the system, and variations in execution time due to factors like caching and pipelining. They are commonly used in situations where precision is not critical or where hardware support for delays is unavailable.

Conclusion:

This experiment aimed to study software-based delay generation in 8086 assembly language using procedures. By displaying two strings without delay and then introducing a specific delay between them, the objectives included understanding procedures, macros, timing calculations of instructions, and DOS interrupts for string display on the monitor.

Signature of faculty in-charge with Date: