MPMC LAB ENDSEM

Joel Scaria | CSE B | 106118042

7th May 2021

Q1.

Assume 10 odd and even numbers mixed in an array of size 10. Write an Alp to Separate odd and even Numbers from it and store at memory locations 1000H and 2000H.

ASM code:

```
data segment

      arr1 db 0,1,2,3,4,5,6,7,8,9
      str_n db 'odd numbers: $'
        str_r db 'even numbers: $'


data ends

code segment
assume cs:code, ds:data

start:

      mov ax,data
      mov ds,ax

      lea bx,arr1
      mov bp, 0Ah
      mov ch,0h
      mov cl,0h
      mov dh,02
      mov si, 1000h
      mov di, 2000h
```

```
L1:
      mov ah, 0h
      mov al, [bx]
      mov dl, al
      div dh
      cmp ah, 0h
      je EVEN1
      mov [si], dl
      inc si
      inc bx
      inc ch
      dec bp
      cmp bp,0h
      jne L1
      jmp STOP


EVEN1:

      mov [di],dl
      inc di
      inc bx
      inc cl
      dec bp
      cmp bp,0h
      jne L1
      jmp STOP

PrintNextLine proc
    push ax
    push dx
    mov dl, 10
    mov ah, 02h
    int 21h
    mov dl, 13
    mov ah, 02h
    int 21h

    pop dx
    pop ax
    ret
PrintNextLine endp

STOP:

      call PrintNextLine
      mov ah, 09h
      lea dx, str_n
      int 21h
      call PrintNextLine
      mov si, 1000h
printodd:
      xor dl,dl
      mov dl, [si]
      add dl, 30h
      mov ah, 02h
      int 21h
      inc si
      dec ch
      call PrintNextLine
      cmp ch ,0h
      jne printodd
      call PrintNextLine
      mov ah, 09h
      lea dx, str_r
      int 21h
      call PrintNextLine
      mov si, 2000h
printeven:
      xor dl,dl
      mov dl, [si]
      add dl, 30h
      mov ah, 02h
      int 21h
      inc si
      dec cl
      call PrintNextLine
      cmp cl ,0h
      jne printeven

      mov ah, 1
      int 21h
      int 3

code ends
end start
```

Explanation:

- For input, I have given 10 digit array 0,1,2,3,4,5,6,7,8,9
- expected output: odd >> 1,3,5,7,9

  even >> 0,2,4,6,8

- input moved to bx

- bp=10 (size of array)

- ch=0 (initial count of odd numbers), cl=0 (initial count of even numbers)

- dh=2 (for checking modulo)

- si=1000h (location for storing odd numbers)

- di=2000h (location for storing even numbers)

- In L1 and EVEN1 :
  elements from bx taken 1 by 1
  they are then divided by 2
  if remainder stored in ah != 0, number stored in [si] (odd) and ch+=1 and si+=1else send to EVEN1
  In EVEN1, number stored in [di] (even) and cl+=1 and di+=1
  Also, bp-=1(one less number to check)
  These steps are repeated 10 times i.e. till bp reaches 0

- Above step gives us odd numbers and even numbers at consecutive locations starting from 1000h(si) and 2000h(di) respectively

- Rest of the program is related to printing the result

- In PrintNextLine proc,
  ah=02h (interrupt for printing ascii character from dl register)
  dl=10 i.e. ascii for new line
  dl=13 i.e. ascii for carriage return

- The strings “odd numbers:” and “even numbers:” are displayed are using interrupt
  mov ah, 09h(interrupt for printing string from dx register)
  int 21h

- printodd set of statements is used for printing odd numbers from 1000h register

  first of all, si register is assigned 1000h (odd)
  dl=0(xor-ing same number gives zero)
  dl=[si] (assigns 1st odd number to dl)
  dl+=30h (adding 30h to the number gives its ascii number)
  mov ah, 02h (interrupt for printing ascii)
  ch-=1 (one less odd number to print)

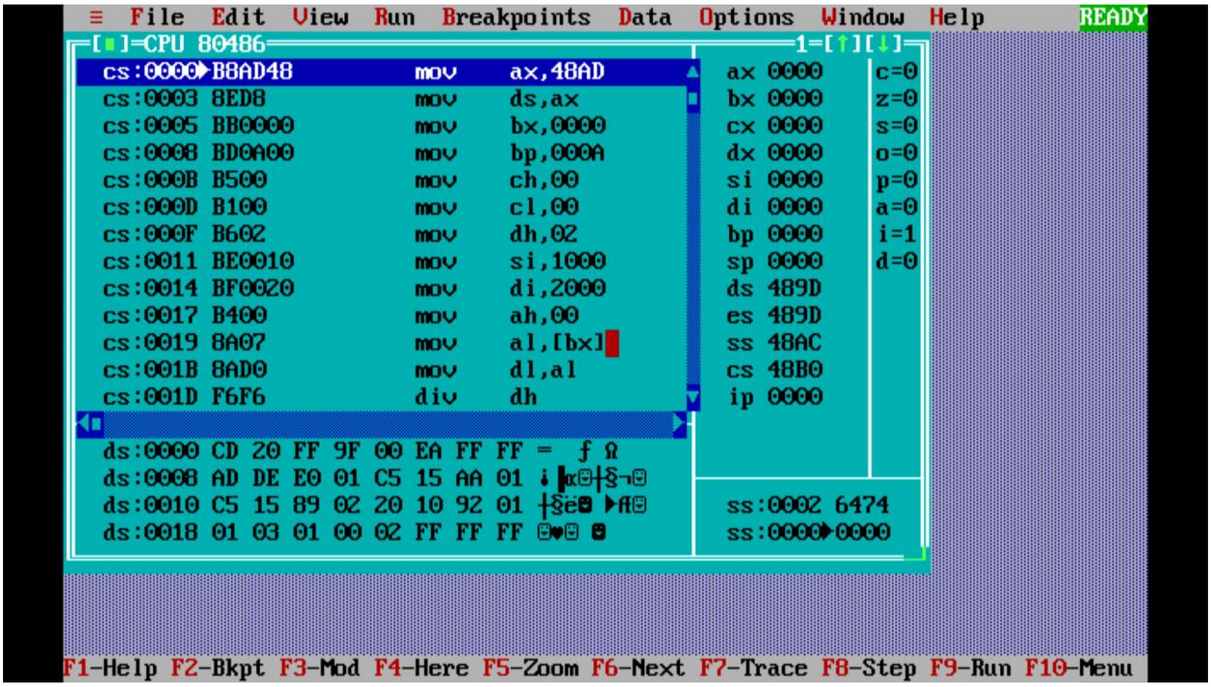
- This process (printodd) is repeated till ch becomes 0

- printeven is the same as printodd except for the following facts:

  si register is assigned 2000h instead of 1000h
  ch is replaced by cl

- Finally, mov ah, 1 interrupt is used to display the terminal (output)
- *Program ends*

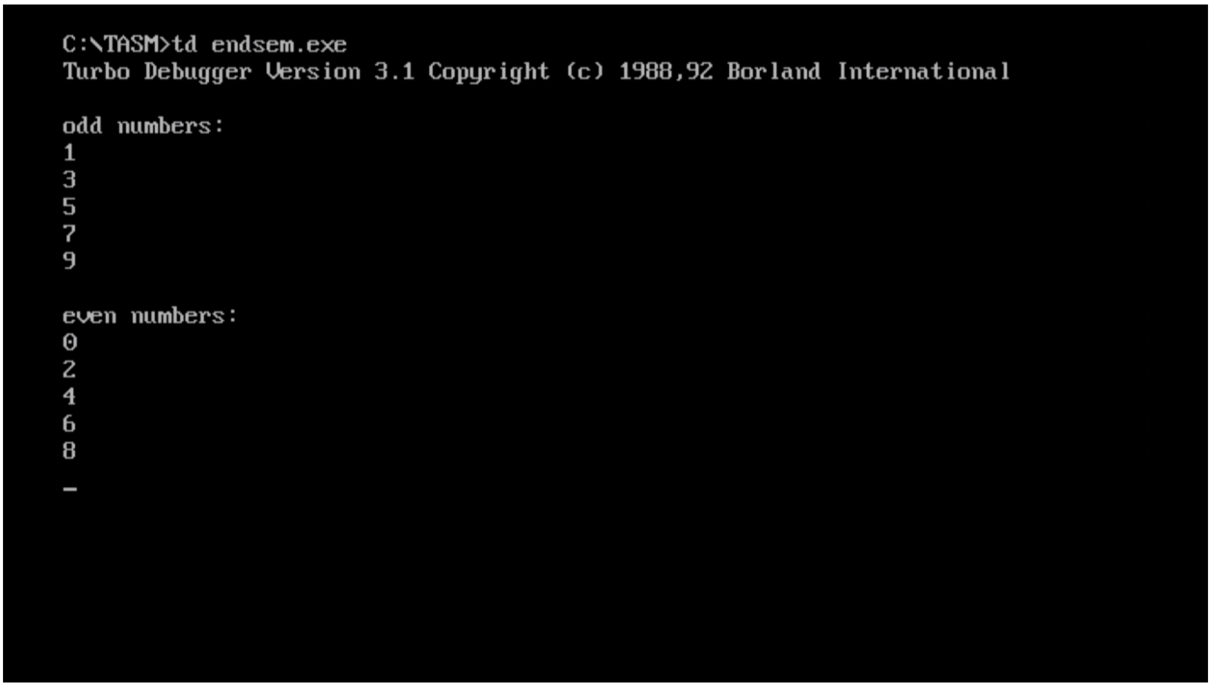
Output:


≡ File Edit View Run Breakpoints Data Options Window Help
READY
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```
[■]=CPU 80486=======================================1=[↑][↓]=
 cs:0000▶B8AD48         mov    ax,48AD          ax 0000   c=0
 cs:0003 8ED8           mov    ds,ax            bx 0000   z=0
 cs:0005 BB0000         mov    bx,0000          cx 0000   s=0
 cs:0008 BD0A00         mov    bp,000A          dx 0000   o=0
 cs:000B B500           mov    ch,00            si 0000   p=0
 cs:000D B100           mov    cl,00            di 0000   a=0
 cs:000F B602           mov    dh,02            bp 0000   i=1
 cs:0011 BE0010         mov    si,1000          sp 0000   d=0
 cs:0014 BF0020         mov    di,2000          ds 489D
 cs:0017 B400           mov    ah,00            es 489D
 cs:0019 8A07           mov    al,[bx]          ss 48AC
 cs:001B 8AD0           mov    dl,al            cs 48B0
 cs:001D F6F6           div    dh               ip 0000

 ds:0000 CD 20 FF 9F 00 EA FF FF =  ƒ Ω
 ds:0008 AD DE E0 01 C5 15 AA 01 ¡▌α☺┼§¬☺
 ds:0010 C5 15 89 02 20 10 92 01 ┼§ë☻ ►Æ☺       ss:0002 6474
 ds:0018 01 03 01 00 02 FF FF FF ☺♥☺ ☻          ss:0000▶0000
```

before running



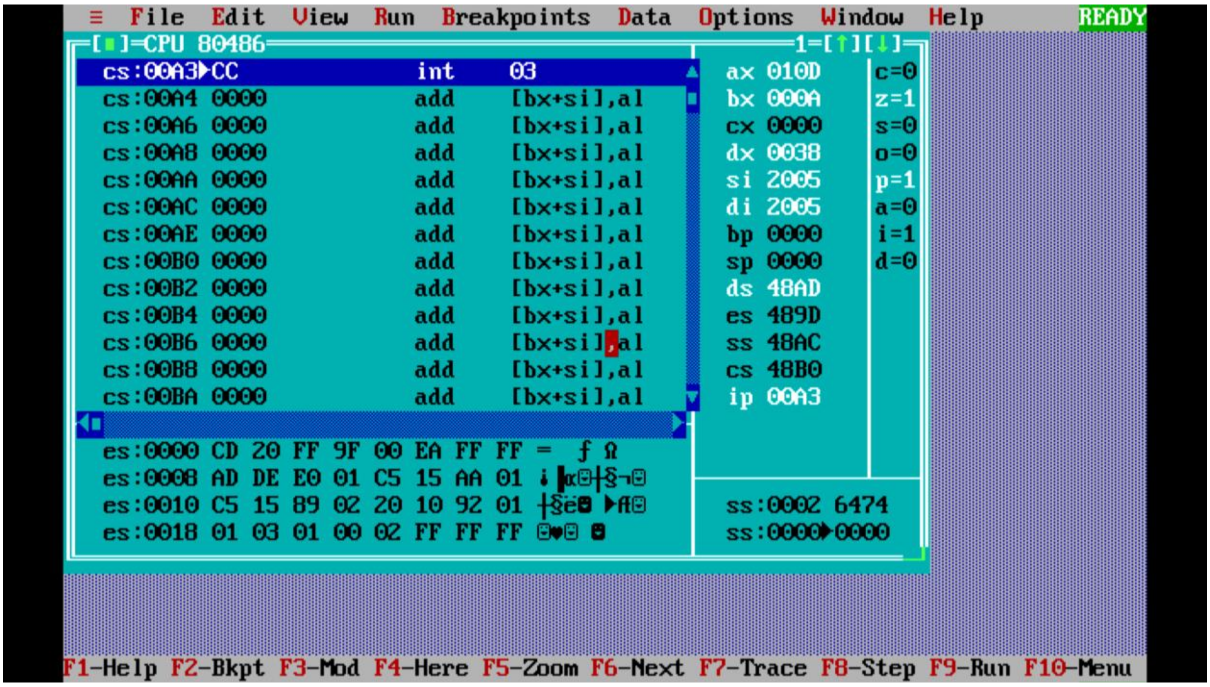
```
C:\TASM>td endsem.exe
Turbo Debugger Version 3.1 Copyright (c) 1988,92 Borland International

odd numbers:
1
3
5
7
9

even numbers:
0
2
4
6
8

_
```

output in terminal


≡ File Edit View Run Breakpoints Data Options Window Help READY
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```
[■]=CPU 80486                                        1=[↑][↓]
 cs:00A3►CC             int     03               ax 010D   c=0
 cs:00A4 0000           add     [bx+si],al       bx 000A   z=1
 cs:00A6 0000           add     [bx+si],al       cx 0000   s=0
 cs:00A8 0000           add     [bx+si],al       dx 0038   o=0
 cs:00AA 0000           add     [bx+si],al       si 2005   p=1
 cs:00AC 0000           add     [bx+si],al       di 2005   a=0
 cs:00AE 0000           add     [bx+si],al       bp 0000   i=1
 cs:00B0 0000           add     [bx+si],al       sp 0000   d=0
 cs:00B2 0000           add     [bx+si],al       ds 48AD
 cs:00B4 0000           add     [bx+si],al       es 489D
 cs:00B6 0000           add     [bx+si],al       ss 48AC
 cs:00B8 0000           add     [bx+si],al       cs 48B0
 cs:00BA 0000           add     [bx+si],al       ip 00A3

 es:0000 CD 20 FF 9F 00 EA FF FF = ƒ Ω
 es:0008 AD DE E0 01 C5 15 AA 01 ¡▌α☺┼§¬☺
 es:0010 C5 15 89 02 20 10 92 01 ┼§ë☻ ►Æ☺         ss:0002 6474
 es:0018 01 03 01 00 02 FF FF FF ☺♥☺ ☻            ss:0000►0000
```

after running

Q2.

Assume that all identifiers are variables and are associated with words. Check whether the following instructions are valid or not. Give explanation.

(a) MOV AX, WORD_OP1 + WORD_OP2

The above MOV instruction is **invalid**

word_op1 and word_op2 are addresses of memory locations

Adding address of memory units doesn't make any sense in assembly language

The MOV command requires 2 parameters, a source address and a destination address.

Destination address (AX) is valid. However, the sum of word_op1 and word_op2 cannot be considered as a source address. So the instruction as a whole becomes invalid

(b) MOV [BX][SI], 2

The above MOV instruction is **valid**

[BX][SI] is equivalent to [BX+SI] where SI denotes address of the SI register

For example, lets assume that location of SI register was 1000h. Then the instruction in question becomes

```
MOV [BX+1000], 2
```

This will store the value 2 at 1000th location from BX register

Judging from above trend, the given statement can be treated as valid