Open-Ended Asssignment 2

Aditya Gupta

Note: The syntax is MASM syntax (DOS), failed to run other emulators

1. CRC
2. CGPA calculation
3. Reversing string
4. Parenthesis Matching
5. Virhanka Numbers (aka Factorials)
6. Most frequent letter
7. Implementing a 3-case switch statement using conditional jumps

Using Interrupts:

8. `Print Capitallised String using DOS 21H syscall`
9. Draw Line using BIOS 10H interrupt
10. Implementing **atoi()** (C library function)

0. Common Code in all:

```
title CGPA        ; title of program (optional)
page 60,132       ; tell the assembler to create a .lst file (optional)
.model small      ; maximum of 64KB for data and code respectively
.stack 64         ; Setup initial size of stack segment to be 64 byte
.data             ; segment directive for data segment
.code             ; segment directive for code segment
main proc far     ; declare a far procedure named ‘main’
      mov ax,@DATA ; load data segment address, "@" is the opcode for fetching
the offset of "DATA"
      mov ds,ax     ; assign value to ds,"mov" cannot be used for copying data
directly to segment registers(cs,ds,ss,es)
      ; Code
; DOS 21h Exit function code
exit:
      mov ah,4ch ; function code for exit
      int 21h    ; terminate program by a normal way
      main endp  ; end the "main" procedure
      end main   ; end the entire program centering around the "main" procedure
```

1906082 Aditya Gupta

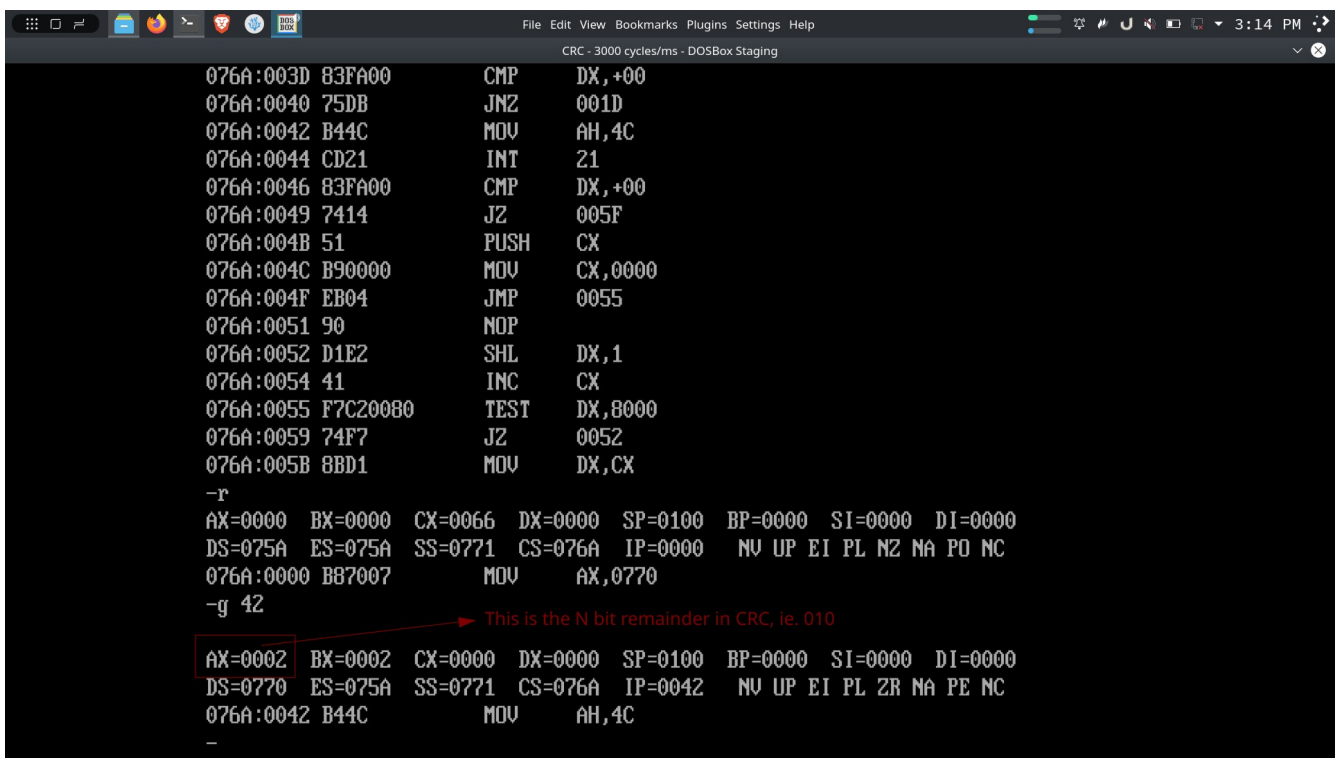
P1. CRC (Cyclic Redundancy Check)

Ref: https://en.wikipedia.org/wiki/Cyclic_redundancy_check

```
; Result can be verified, since used the same example bits as shown in the example
(though 13 bit message instead of 14 bit, wikipedia uses 17 bit result)
; It will work for other CRCs too, this is N=3, so CRC3

; Returns ‘remainder’ bits in AX, for given message and generator polynomial
(here 1011)

title CRC
page 60,132
.model small
.stack 100h
.data
Message  DW 1A76h ; must be exactly (16-N) bits -> 1 1010 0111 0110
N        equ 03h ; n in crc, eg. 3 for CRC3, 5 for CRC5, etc
GEN_POLY equ 0Bh ; (N+1 bits) -> 1011

; Storage required:
; MESSAGE (will work as result too) -> 16-bit
; Generator polynomial N bits -> (but will be stored in 16-bit to be xorred) ->
16-bit

.code
main proc far
      mov ax,@DATA
      mov ds,ax

      ; Stores the message/result in AX
      mov ax, Message ; ax = MESSAGE

store_gen_poly_in_16_bit:
      mov bx, GEN_POLY ; Store gen poly in bx

      ; Loop while the 1st non-0 bit of the poly becomes the MSB
      ; while bx & 0x8000 == 0
      ; shl bx, 1
      jmp loop1
shift1:
      shl bx, 1 ; Shift left 1 bit
loop1:
      test bx, 8000h ; ANDs bx and 0x8000 (ie. 1000 0000 0000 0000) to check if
MSB is 1, if not, zero flag is set
      jz shift1 ; If bx & 0x8000 == 0, jump to shift1
      ; Now shifting message bits to have N zeroes at end
      mov cx, N
      shift2:
```

```
      shl ax, 1 ; append n zeroes, by shifting left N bits
      loop shift2

      ; Now, ax stores the message(&result), bx stores generator

      ; Algo:
      ; Keep shifting generator xor with message
      ; until the message part of result is 0
      ; last n bit of result is remainder

      ; while ax<15(MSB):3(N)> != 0
      ; xor ax, bx
      ; shr bx, 1
loop2:
      xor ax, bx

      ; Now updating bx, 'for next iteration'
      mov cx, 0
      ; available registers: cx, dx
      ; add condition here to shift bx, so that first 1 of ax and bx matches
      ; Finding leading zeroes in result
      mov dx, ax
      ; add cx, number_of_leading_zeroes_in_dx
      call num_leading_zeroes_in_dx ; answer stored in dx itself
      add cx, dx

      ; Now, finding leading zeroes in bx
      mov dx, bx
      ; sub cx, number_of_leading_zeroes_in_dx
      call num_leading_zeroes_in_dx ; answer is stored in dx
      sub cx, dx
shift_bx:
      shr bx, 1
      loop shift_bx
      ; Shifting right N bits, (to ignore last N bit, to get ax<15:N>)
      mov cx, N
      mov dx, ax ; Temporarily copying ax into dx
shift3:
      shr dx, 1 ; Shift 1 bit right
      loop shift3

      cmp dx, 0 ; Set zero flag for next jump
      jnz loop2 ; loop to xor again

exit:
      mov ah,4ch ; function code for “Exit”
      int 21h ; DOS syscall interrupt
main endp ; End procedure ‘main’
```

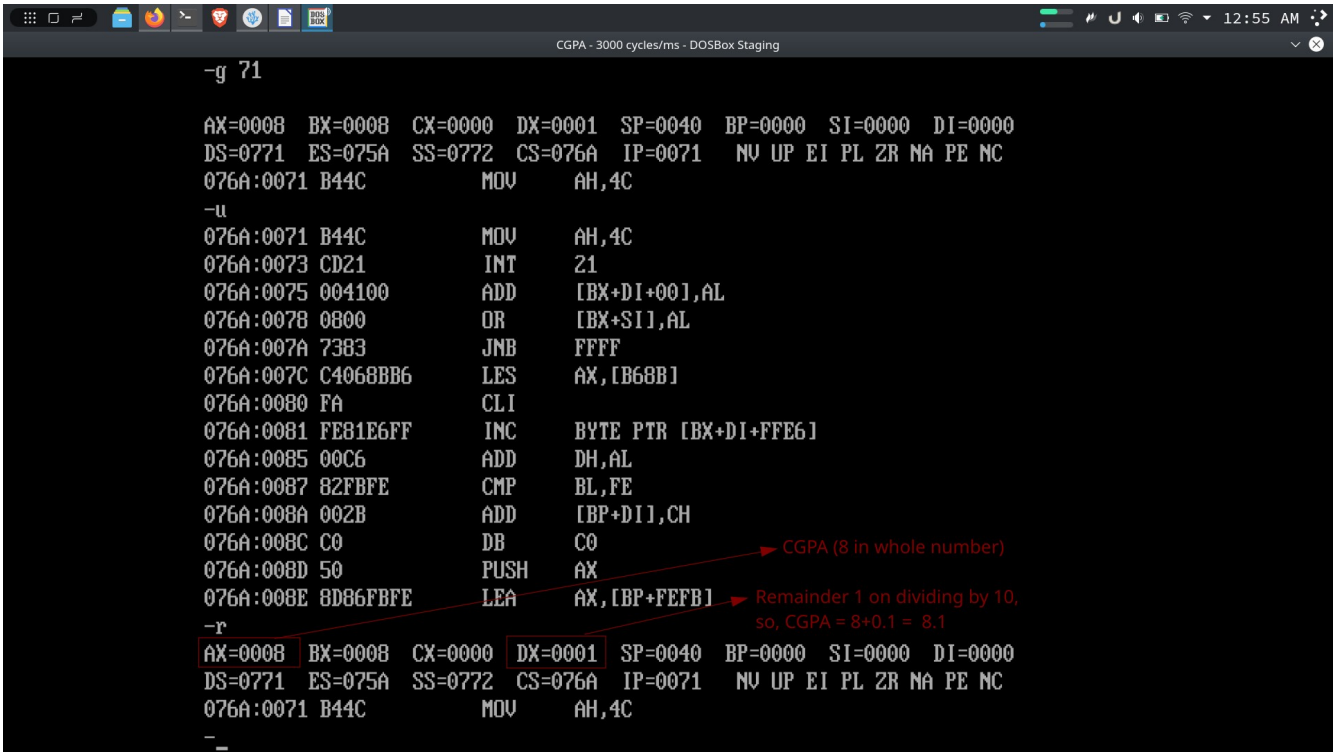
```
num_leading_zeroes_in_dx proc near
      cmp dx, 0
      jz all_zeroes ; Handling the corner case, when no 1 in 16-bit then answer
'would have been' > 16
      push cx ; push cx onto stack, to save it's value in memory
      mov cx, 0 ; cx = 0
      jmp loop_num
      ; while dx & 0x8000 == 0
      ; ++cx
shift_num:
      shl dx, 1 ; Shift left 1 bit
      inc cx ; ++cx, increase count
loop_num:
      test dx, 8000h ; AND dx and 1000 0000 0000 0000, to test if MSB is 1
      jz shift_num ; true when MSB was NOT 1
      mov dx, cx ; since caller expects output in dx, copy cx's value to dx
      pop cx ; after use of cx done, we restore it's previous value
      ret ; return
all_zeroes:
      mov dx, 16 ; dx = 16 = number_of_bits_in_dx
      pop cx ; after use of cx done, we restore it's previous value
      ret
num_leading_zeroes_in_dx endp
end main
```



P2. CGPA calculation (like done in NITP (*assuming), & taking input from the stack as if some other called a function)

```
; This requires some caller to put the arguments on stack,
; Manually added, the stack contains: N (on top), ('C', 4),('A',3),('A','+',1)
(each 2 bytes)
; Explanation/Assumption:
; A+ -> 10
; A  -> 9... and so on
;
; So, if get A+, (lab)course of credit 1, the weighted grade point is 10*1 = 10
;     if get A,  in a course of credit 3, the weighted grade point is 9*3  = 27
;     if get C,  in a course of credit 4, the weighted grade point is 7*4  = 28
;
; Considering only those three courses (as this program does), the GPA will be:
; (10 + 27 + 28)/(1+3+4) = 8.1 :)

title CGPA
page 60,132
.model small
.stack 64
.data
      accumulated_grade_points DW 0 ; Stores total marks (will be divided by dx
to obtain cg)
      total_credit DW 0 ; Stores total credits

.code
main proc far
      mov ax,@DATA
      mov ds,ax


      pop cx ; N will be 2 bytes

      ; while cx > 0
      ; 2 popped characters
      ; add
      cmp cx, 0
      jle exit ; Invalid input N <= 0

iter:
      pop bx ; pop grade
      jmp credit
reinput_cred:
      dec bx ; --bx, so 'A' becomes 'A'-1 (so that it is 'farther')
      pop dx ; MUST contain credits
      jmp continue
credit:
```

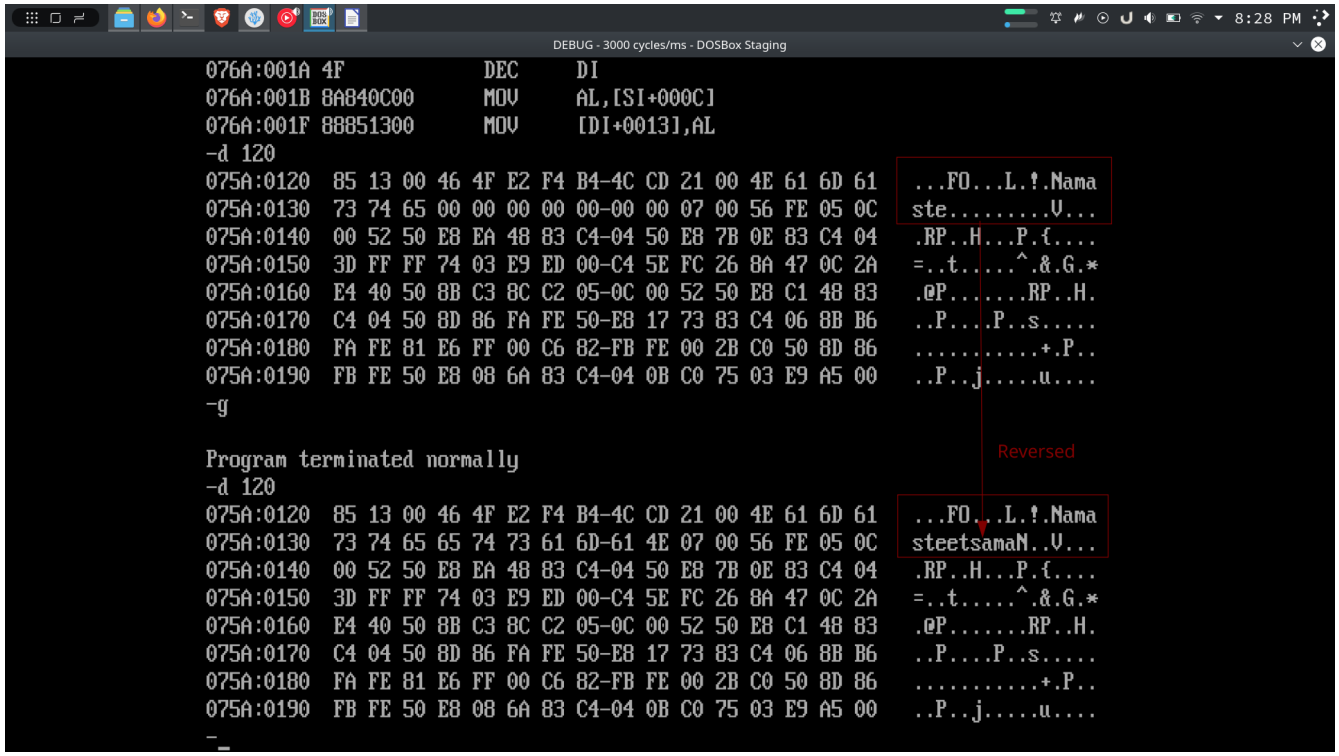
```
      pop dx ; contains credits
      cmp dx, '+' ; If it was '+' instead of a number
      je reinput_cred
continue:
      mov ax, total_credit
      add ax, dx ; add current course's credit

      push bx ; store value of bx on stack
      lea bx, total_credit ; load effective address (ie. to get address of
total_credit in bx)
      mov WORD PTR [bx], ax ; store updated total_credit
      pop bx ; restore value of bx

      sub bx,'J' ; bx = bx - 'J'; 'A'-'J' becomes -9
      neg bx ; negate bx, ie. -9 -> 9

      mov ax, 1
      push dx ; Store dx value, as mul may use DX for overflow if any
      mul bx ; 1*grade_point
      pop dx ; Restore dx value
      mul dx ; 1*grade_point*credits
; CRITICAL WARNING: I am ignoring DX, that may also have value, but not
; working with larger than 1 byte, so if it occurs it's a logical error
      mov dx,ax ; save the result of multiplication in dx

      mov ax, accumulated_grade_points
      lea bx, accumulated_grade_points ; load effective address, get address
      add ax, dx ; add current sum to ax
      mov WORD PTR [bx], ax

      dec cx
      jnz iter

      mov ax, accumulated_grade_points ; ax contains cummulative grade point
      ('A+' (ie. 10) + 'A'(ie. 9) +'B'(ie. 8))
      mov bx, total_credit ; dx contains total credits
      mov dx, 0 ; dx:ax is divided by bx in next step
      div bx ; ax = ax/dx is cgpa (whole number in ax,
      ; remainder in dx)

; DOS 21h Exit function code
exit:
      mov ah,4ch
      int 21h
      main endp
      end main
```


12:55 AM
CGPA - 3000 cycles/ms - DOSBox Staging
-g 71
AX=0008 BX=0008 CX=0000 DX=0001 SP=0040 BP=0000 SI=0000 DI=0000
DS=0771 ES=075A SS=0772 CS=076A IP=0071 NV UP EI PL ZR NA PE NC
076A:0071 B44C MOV AH,4C
-u
076A:0071 B44C MOV AH,4C
076A:0073 CD21 INT 21
076A:0075 004100 ADD [BX+DI+00],AL
076A:0078 0800 OR [BX+SI],AL
076A:007A 7383 JNB FFFF
076A:007C C4068BB6 LES AX,[B68B]
076A:0080 FA CLI
076A:0081 FE81E6FF INC BYTE PTR [BX+DI+FFE6]
076A:0085 00C6 ADD DH,AL
076A:0087 82FBFE CMP BL,FE
076A:008A 002B ADD [BP+DI],CH
076A:008C C0 DB C0
076A:008D 50 PUSH AX
076A:008E 8D86FBFE LEA AX,[BP+FEFB]
-r
AX=0008 BX=0008 CX=0000 DX=0001 SP=0040 BP=0000 SI=0000 DI=0000
DS=0771 ES=075A SS=0772 CS=076A IP=0071 NV UP EI PL ZR NA PE NC
076A:0071 B44C MOV AH,4C
-
CGPA (8 in whole number)
Remainder 1 on dividing by 10,
so, CGPA = 8+0.1 = 8.1

P3. Reversing string

```
title Reverse String
page 60,132
.model small
.stack 100h
.data
      src db 'Namaste'
      dest db 7 dup(?)
      count dw 7

.code
main proc far ; declare the 'main' procedure
      mov ax,@DATA ; temporary step to store value of @data in data segment
register
      mov ds,ax ; ds = ax = @DATA

begin: mov es,ax ; storing address in ax, in extra segment register
      mov cx,count ; cx = count
      mov si,0 ; si = 0 (used for source indexing)
      mov di, count ; di = count
      dec di ; --di = count - 1

; could use "movsb", and "stosb" instruction too
again: mov al, src[si] ; store byte at src[si] in al
      mov dest[di], al ; store byte in al, at dest[di]
      inc si ; ++si
      dec di ; --di
      loop again ; jmp to 'again' label, till cx != 0

; DOS interrupt 21h, exit function code 4ch part
exit:
      mov ah,4ch
      int 21h
      main endp
      end main
```

1906082 Aditya Gupta

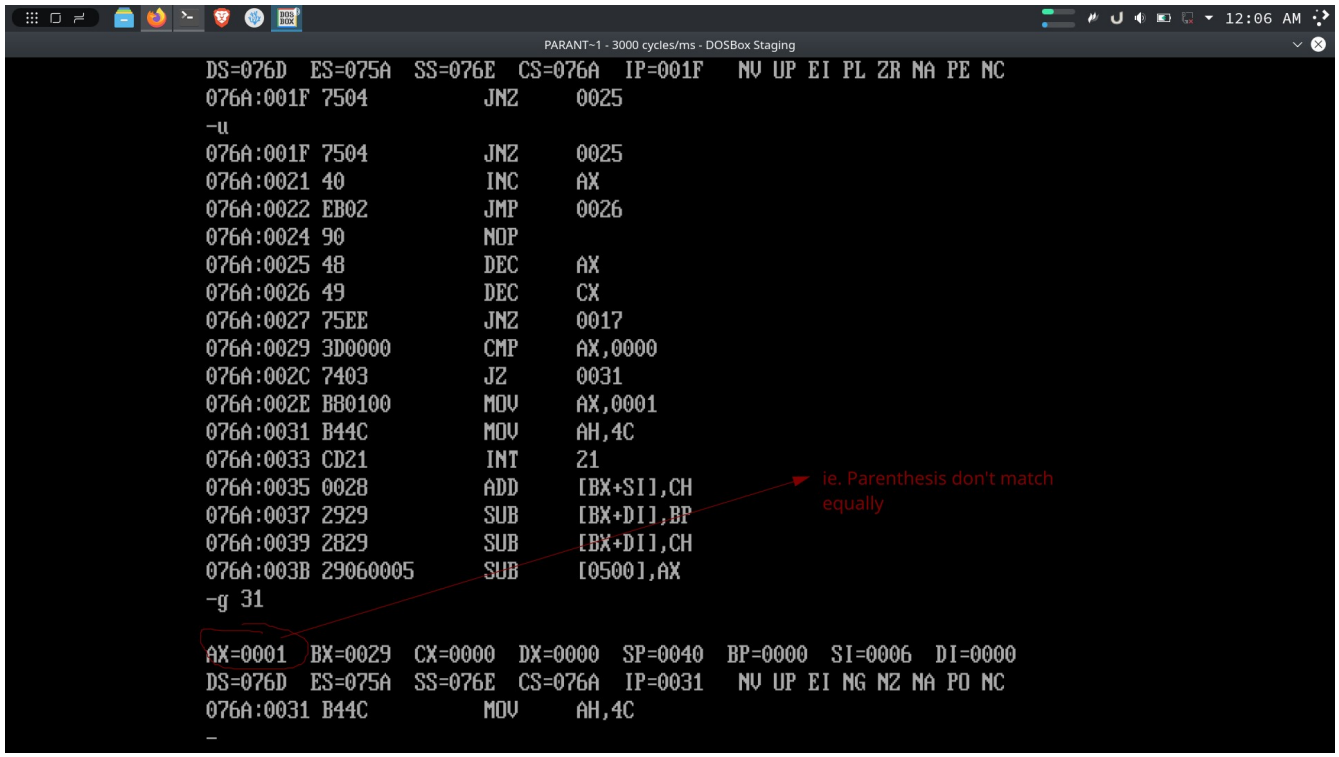


P4. Parenthesis matching

Note: Not 'ballanced parenthesis problem', that is a subset of this, here we just check if equal number of open and close brackets are there in the set, so also accepts '())(' is also accepted in this

```
title Parenthesis Matching
page 60,132
.model small
.stack 64
.data
      string DB "()()()" ; The string with open and close parenthesis
      N equ 6h ; String length

.code
main proc far
      mov ax,@DATA
      mov ds,ax

      mov cx, N ; Initialise counter with length
      mov si, 0 ; SI = 0
      ; while cx>0:
      ; next
      ; add

      cmp cx,0 ; To set flags for the next jmp
      jle exit ; N <= 0, Exit rightaway for invalid input

      mov ax, 0 ; ax = 0
loop1:
      mov bl, string[si] ; Load a byte from string[si] into bl
      inc si ; ++SI
      cmp bl,'(' ; Checking if the character is '('
      jne decrease ; If not, then --ax
      inc ax ; If it is '(', then ++ax
      jmp next ; to skip the decrease step
decrease:
      dec ax ; --ax
next:
      dec cx ; --cx
      jnz loop1 ; if cx != 0, then go to loop1
; Could use 'loop' statement too

; If ax == 0, matched, else not matched
      cmp ax, 0
      jz exit
      mov ax, 1 ; ax = 1, signifies parenthesis not matched

exit:
      mov ah,4ch
```

```
      int 21h
main endp
end main
```

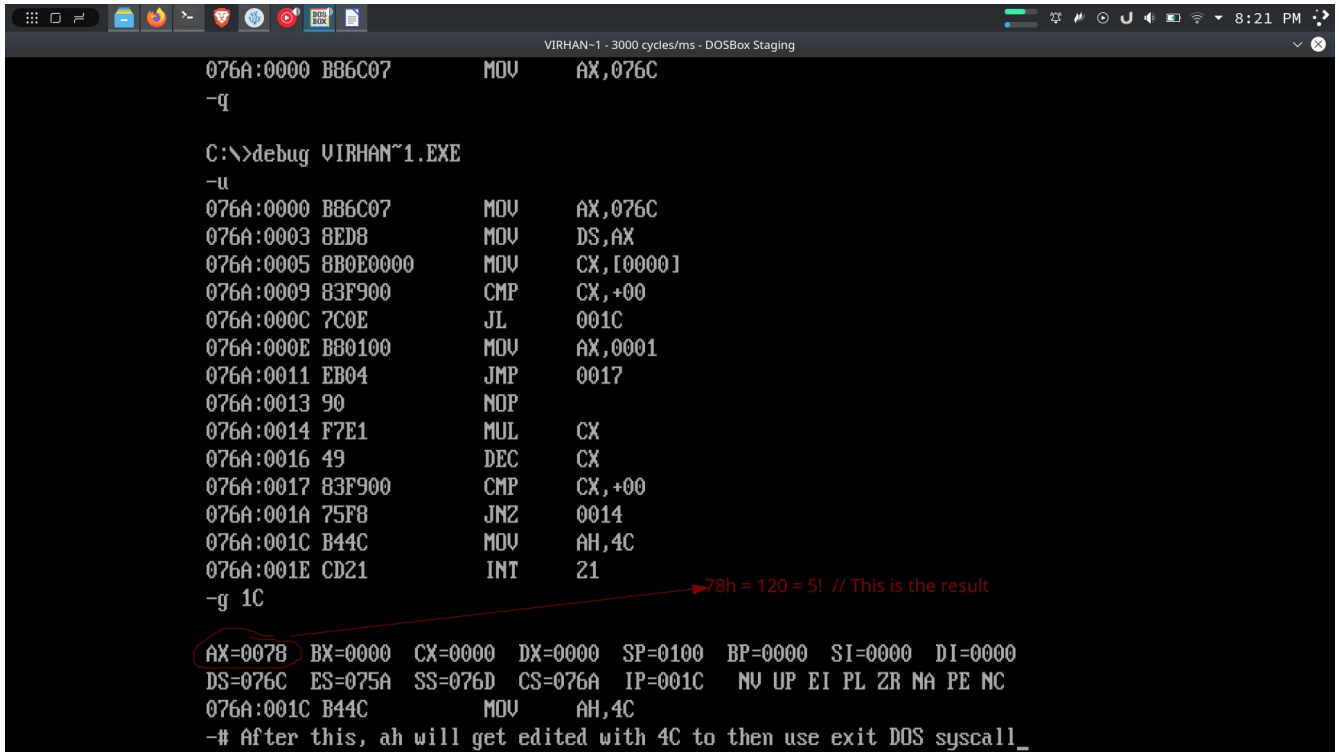


P5. Virhanka Numbers (aka Factorials)

Ref: https://www.cse.iitb.ac.in/~cs101/2012.2/resources/VirahankaNumbers.pdf

```
title Virhanka
page 60,132
.model small
.stack 100h
.data
      N equ 5h    ; Edit to change number
.code
      main proc far
      mov ax,@DATA
      mov ds,ax

      mov cx,N    ; cx = N
      cmp cx, 0   ; compares value in cx with 0, and sets flags
      jl exit     ; Exit for invalid input (ie. for cx < 0)
      mov ax,1    ; ax = 1, this will then be multiplied to get the factorial
      jmp iter    ; Unconditional jump to 'iter' label
; while cx != 0:
;     mul cx
;     dec cx
multiply:
      mul cx      ; ax = ax * cx
      dec cx      ; --cx
iter:
      cmp cx, 0   ; compare cx with 0
      jnz multiply ; Repeat till cx becomes zero
      ; ax has the result
exit:
      mov ah,4ch
      int 21h
main endp
end main
```

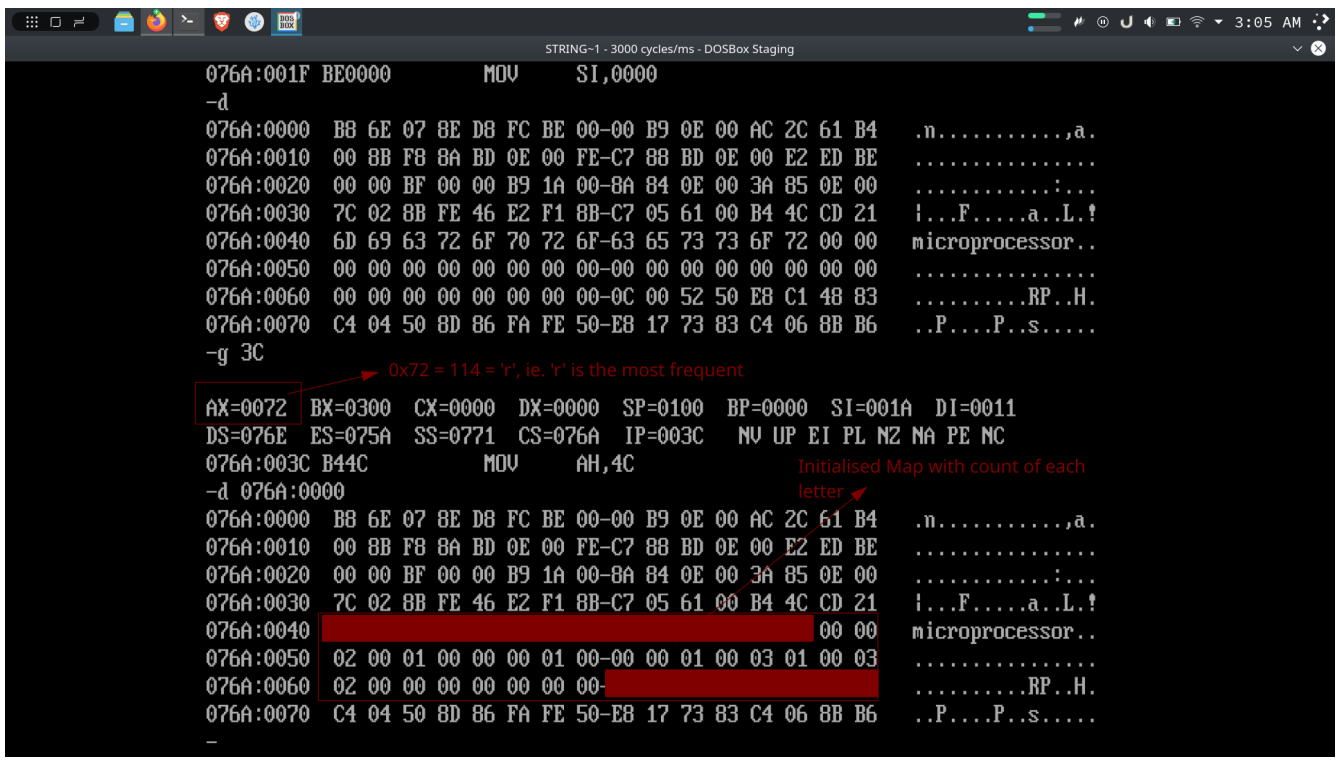
Output:


8:21 PM
VIRHAN~1 - 3000 cycles/ms - DOSBox Staging
076A:0000 B86C07 MOV AX,076C
-q
C:\>debug VIRHAN~1.EXE
-u
076A:0000 B86C07 MOV AX,076C
076A:0003 8ED8 MOV DS,AX
076A:0005 8B0E0000 MOV CX,[0000]
076A:0009 83F900 CMP CX,+00
076A:000C 7C0E JL 001C
076A:000E B80100 MOV AX,0001
076A:0011 EB04 JMP 0017
076A:0013 90 NOP
076A:0014 F7E1 MUL CX
076A:0016 49 DEC CX
076A:0017 83F900 CMP CX,+00
076A:001A 75F8 JNZ 0014
076A:001C B44C MOV AH,4C
076A:001E CD21 INT 21
-g 1C
78h = 120 = 5! // This is the result
AX=0078 BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=076C ES=075A SS=076D CS=076A IP=001C NV UP EI PL ZR NA PE NC
076A:001C B44C MOV AH,4C
-# After this, ah will get edited with 4C to then use exit DOS syscall

P6. Most frequent letter in string

```
title Max frequency
page 60,132
.model small
.stack 100h
.data
      STRING db "microprocessor" ; ALL LOWERCASE
      LEN equ $ - STRING ; '$' evaluates to current address, so $-STRING is
length of string, ie. 14, this feature is by assembler so didn't use in other
      MAP db 26 DUP(0)

; MAP is basically a 26 length array, where 'a' maps to index 0, and 'z' to 25
; Each index stores a count of how many times the number occured

.code
main proc far
      mov ax,@DATA
      mov ds,ax

      cld ; set direction flag = 0
      mov si, OFFSET STRING ; init SI
      mov cx, LEN
count:
      lodsb ; load 1 byte from string to AL
      sub al, 'a' ; Now al will have 0 for 'a', 1 for 'b', so on...

      mov ah, 0 ; ax = 0
      mov di,ax ; di = ax, setting a destination index
      mov bh, MAP[di] ; get previous count of occurences of the current letter
      inc bh ; ++bh, increment number of current letter
      mov MAP[di], bh ; Updating the count, stored at MAP indexed by di
      loop count ; loop until cx != 0

      ; while(cx!=0):
      ; if MAP[si] > MAP[di]:
      ; dx = MAP[si]
      ; si++
      mov si, 0 ; si = 0
      mov di, 0 ; di = 0; di will store max index
      mov cx, 26 ; to iterate over 26 character map
find_max:
      mov al, MAP[si]
      cmp al, MAP[di]
      jl continue ; if lesser, then continue
update_max:
      mov di, si ; else update index of max element, ie. di
continue:
```

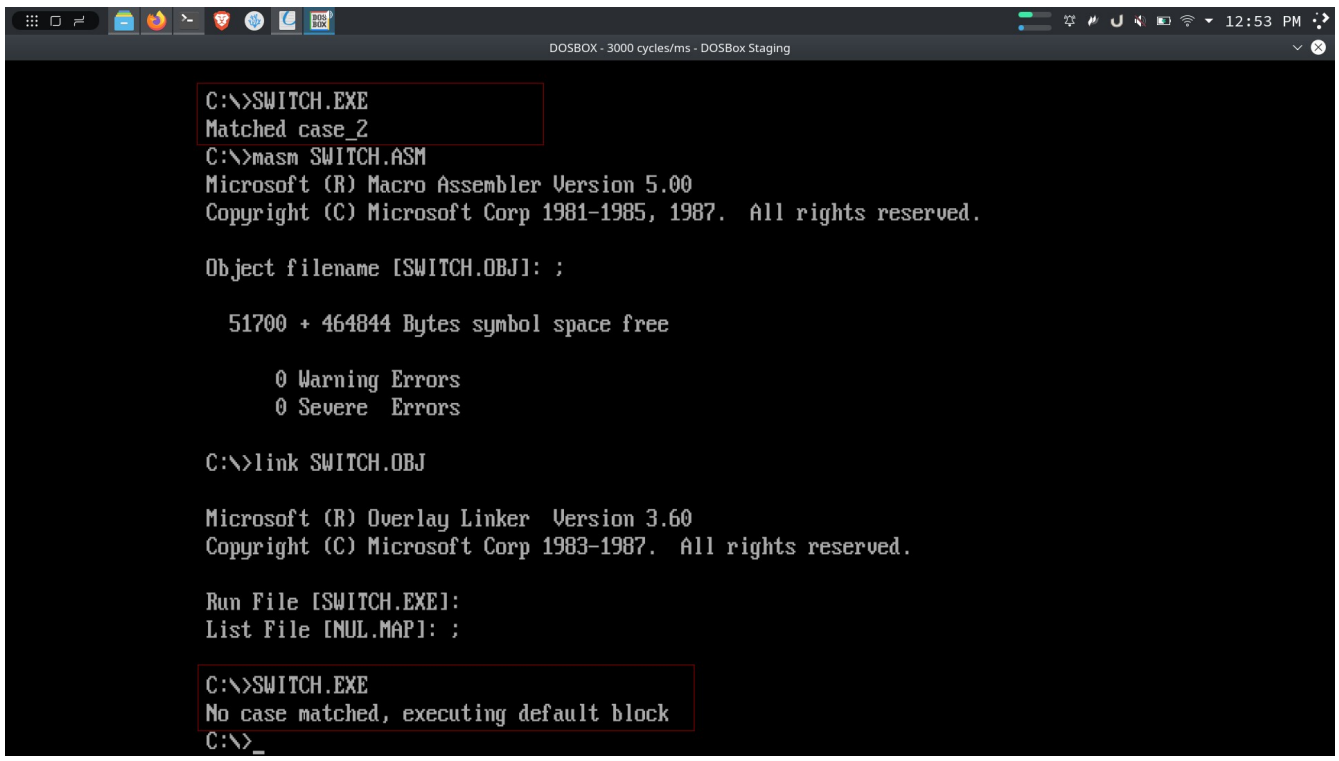
```
      inc si ; ++si
      loop find_max ; loop until cx != 0

      ; di has index of most frequent letter
      mov ax,di ; ax = di
      add ax,'a' ; ax = ax + 'a' = 'a' + di, ie. we get the most frequent
character in ax

; ax has most frequent letter

; DOS Exit function
exit:
      mov ah,4ch
      int 21h
main endp
end main
```



P7. Implementing 3-case switch using conditional jumps

```
title switch
page 60,132
.model small
.stack 64
.data
; Expecting input switch on stack top, ie. some caller added it

      CASE_1 equ 1Ah
      CASE_2 equ 3Bh
      CASE_3 equ 04h

      CASE_1_STR db "Matched case_1$"
      CASE_2_STR db "Matched case_2$"
      CASE_3_STR db "Matched case_3$"
      CASE_DEF_STR db "No case matched, executing default block$"

.code
main proc far
mov ax,@DATA
mov ds,ax

 ; mov ax, 0bh (example: pushing the input onto stack manually)
 ; push ax

      pop bx      ; SWITCH_INPUT is on top of stack, pop it into bx

      mov ax, bx
      xor ax, CASE_1 ; ax = ax^bx = SWITCH_INPUT ^ CASE_1
      jz case1 ; if ax^bx = 0 => SWITCH_INPUT == CASE_1

      xor ax, CASE_2
      xor ax, CASE_1 ; ax = (switch^case1)^(case1^case2) =
      ; (switch^case2)^0 = switch ^ case2
      jz case2

      xor ax, CASE_3
      xor ax, CASE_2 ; => ax = SWITCH_INPUT ^ CASE_3 ^ 0
      jz case3 ; if ax == 0, then CASE_3 matched SWITCH_INPUT
      jmp default ; Unconditional jump to the default case

case1:
      lea dx, CASE_1_STR     ; load effective address, load location/offset of
CASE1STR into dx
      jmp break
case2:
      lea dx, CASE_2_STR
```

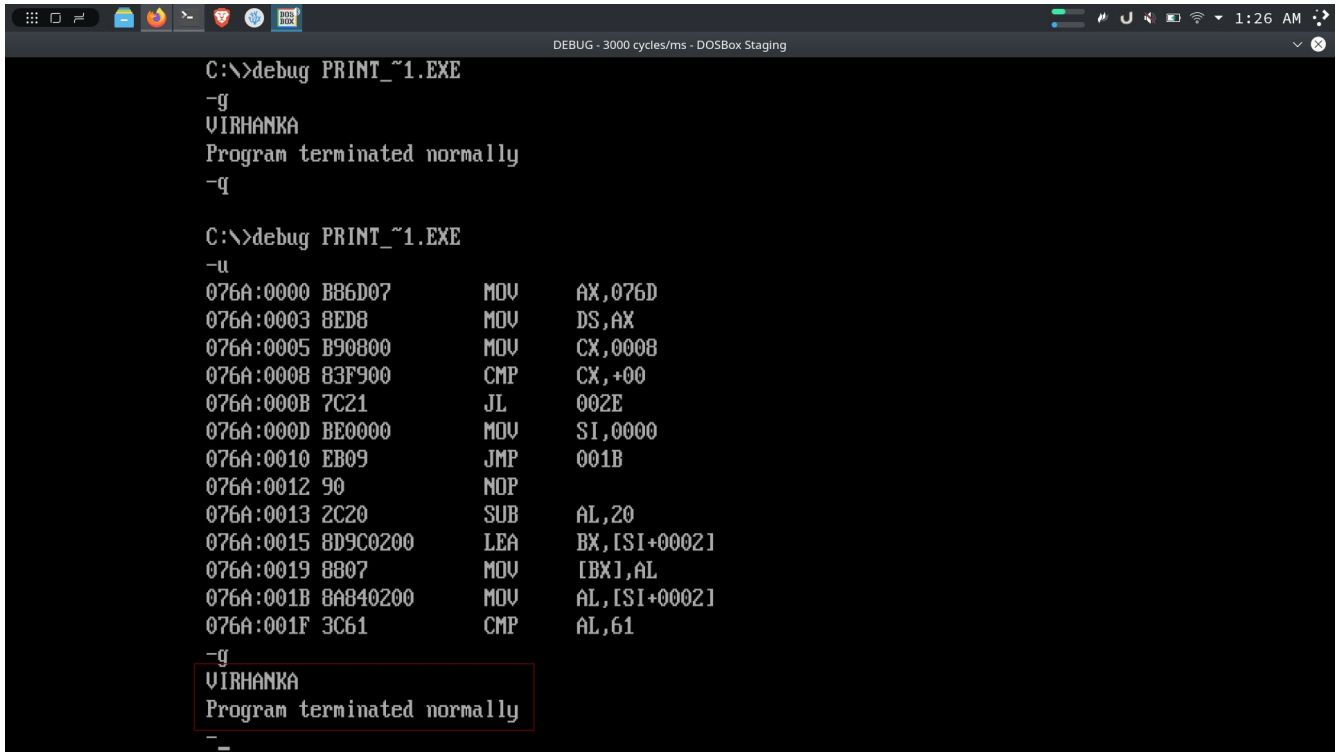
```
      jmp break
case3:
      lea dx, CASE_3_STR
      jmp break
default:
      lea dx, CASE_DEF_STR
      jmp break

break:
      mov ah, 09h
      int 21h
; DOS 21h Exit function code
exit:
      mov ah,4ch
      int 21h
main endp
end main
```



P8. Using Interrupts: (a) Print Capitallised String using DOS 21H syscall

```
title Using Interrupt (a)
page 60,132
.model small
.stack 100h
.data
      string DB "VirHAnKa$"
      N equ 8h
.code
main proc far
      mov ax,@DATA
      mov ds,ax

      mov cx,N
      cmp cx, 0 ; compares value in cx with 0, and sets flags
      jl exit ; Exit for invalid input (ie. for N < 0)

      mov si, 0
      jmp iter
to_upper:
      sub al, 20h ; al -= 20h, because 'a'- 32 = 'A', 32 = 0x20
      lea bx, string[si] ; storing the updated character back at its place
      mov BYTE PTR [bx], al
iter:
      mov al, string[si]
      cmp al, 61h ; 'a' = 97 = 0x61, assuming ONLY alphabets in string,
; this WILL be lowercase letter if >= 97
      jge to_upper ; lowercase letter, convert to upper
      inc si
      loop iter ; loop until cx != 0
print:
      mov ah, 09h ; Function Code: WRITE STRING TO STANDARD OUTPUT
      lea dx, string
      int 21h
exit:
      mov ah,4ch ; function code for “Exit”
      int 21h ; DOS syscall interrupt
main endp ; End procedure ‘main’
end main
```

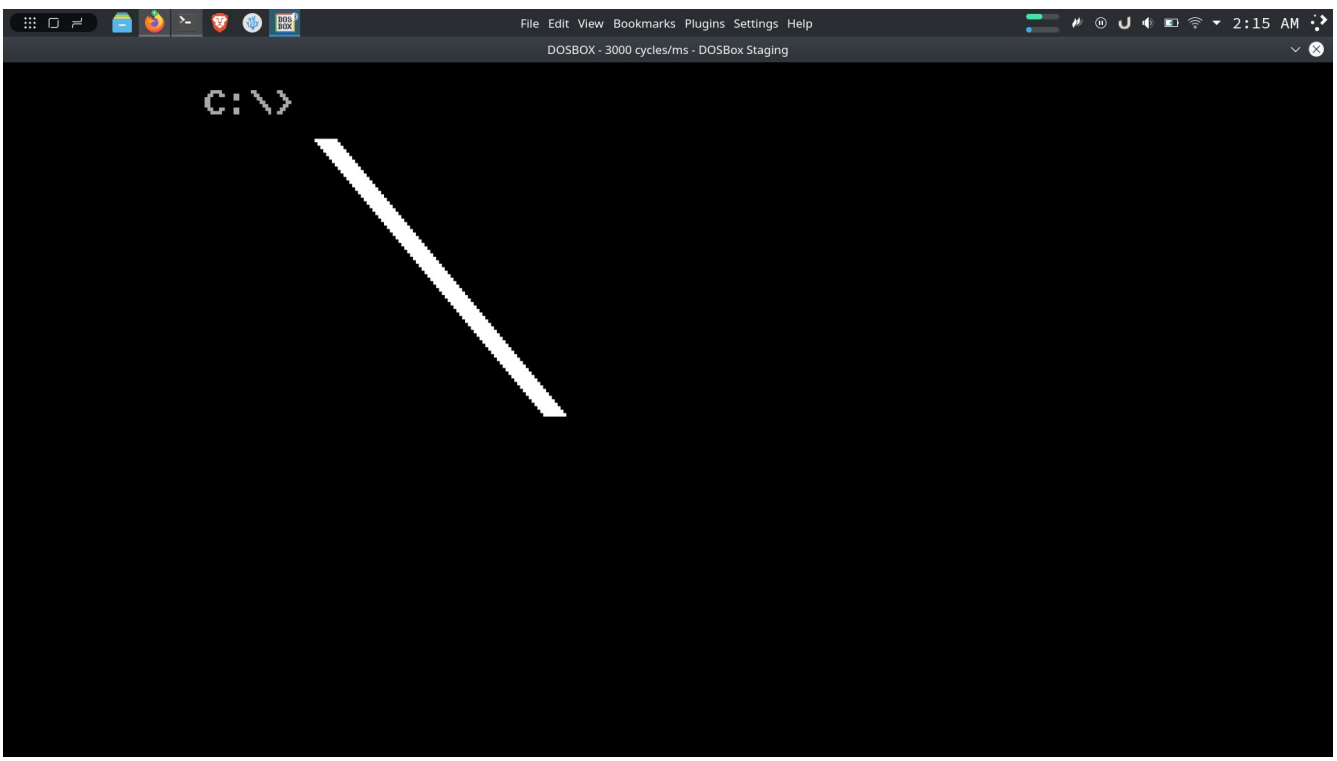
1906082 Aditya Gupta



P9. Using Interrupts: (b) Drawing a line using DOS 21H syscall

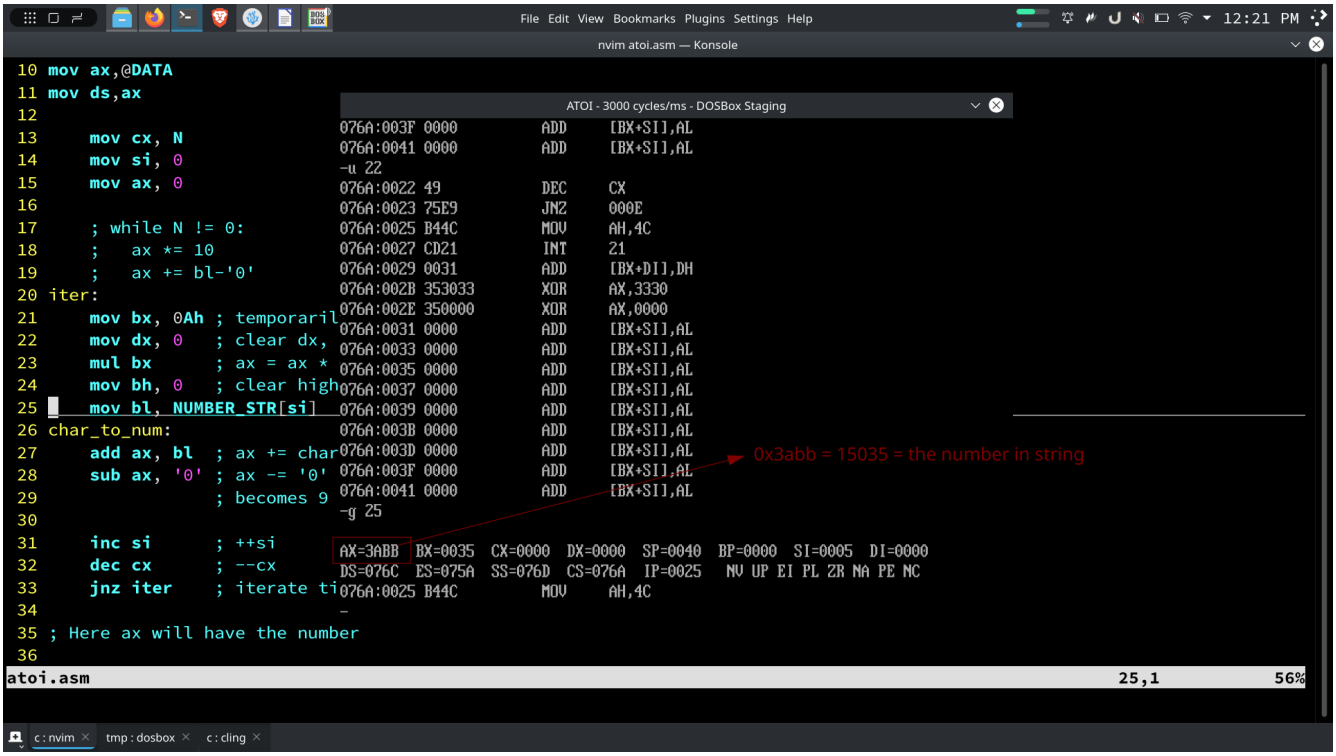
```
title Using Interrupt (b)
page 60,132
.model small
.stack 100h
.data
      LEN equ 50h
      PADDING equ 8h
      START_X equ 24h
      START_Y equ 15h
.code
main proc far
      mov ax,@DATA
      mov ds,ax

setup_graphics:
      mov ah, 00h ; set config to video mode
      mov al, 13h ; chosing the video mode
      int 10h ; BIOS interrupt

      mov ah,0Bh ; set config to
      mov bh, 0 ; background color
      mov bl, 0 ; chosing black as background
      int 10h

      mov bx,START_X ; x_coord
      mov dx,START_Y ; y_coord
      mov cx,LEN ; cx = LEN
      inc cx ; ++cx
      jmp draw_line

draw_horizontal:
      push cx
      push bx

      mov cx,PADDING
pixel:
      push cx

      inc bx
      mov cx,bx
      call draw_pixel

      pop cx
      dec cx
      jnz pixel

      pop bx
```

```
      pop cx
draw_line:
      inc bx ; x_coord++
      inc dx ; y_coord++

      dec cx
      jnz draw_horizontal

exit:
      mov ah,4ch ; function code for “Exit”
      int 21h ; DOS syscall interrupt
main endp ; End procedure ‘main’

draw_pixel proc near
      mov ah, 0Ch ; set config to drawing pixel
      mov al, 0Fh ; chosing white colour
      mov bh, 0 ; page number
      ; Expecting cx & dx to be set to coord, done in draw_horizontal
      int 10h

      ret
draw_pixel endp

end main
```



P10. atoi() (only for whole numbers, C library function, implementing in 8086 assembly)

```
; Limitation: Max 2 byte number, ie. max 65535

title atoi
page 60,132
.model small
.stack 64
.data
      NUMBER_STR DB "15035"
      N equ $ - NUMBER_STR
.code
main proc far
      mov ax,@DATA
      mov ds,ax

      mov cx, N
      mov si, 0
      mov ax, 0
; while N != 0:
; ax *= 10
; ax += bl-'0'
iter:
      mov bx, 0Ah ; temporarily storing 10 in dx
      mov dx, 0 ; clear dx, since that is used in mul of words
      mul bx ; ax = ax * 0x0A = ax * 10
      mov bh, 0 ; clear higher byte of bx
      mov bl, NUMBER_STR[si]
char_to_num:
      add ax, bl ; ax += character (bl)
      sub ax, '0' ; ax -= '0' = character - '0'; ie. '0' will become 0, '9'
; becomes 9

      inc si ; ++si
      dec cx ; --cx
      jnz iter ; iterate till cx != 0

; Here ax will have the number

; DOS 21h Exit function code
exit:
      mov ah,4ch
      int 21h
main endp
end main
```


File Edit View Bookmarks Plugins Settings Help
12:21 PM
nvim atoi.asm — Konsole
10 mov ax,@DATA
11 mov ds,ax
12
13 mov cx, N
14 mov si, 0
15 mov ax, 0
16
17 ; while N != 0:
18 ; ax *= 10
19 ; ax += bl-'0'
20 iter:
21 mov bx, 0Ah ; temporaril
22 mov dx, 0 ; clear dx,
23 mul bx ; ax = ax *
24 mov bh, 0 ; clear high
25 mov bl, NUMBER_STR[si]
26 char_to_num:
27 add ax, bl ; ax += char
28 sub ax, '0' ; ax -= '0'
29 ; becomes 9
30
31 inc si ; ++si
32 dec cx ; --cx
33 jnz iter ; iterate ti
34
35 ; Here ax will have the number
36
atoi.asm 25,1 56%
ATOI - 3000 cycles/ms - DOSBox Staging
076A:003F 0000 ADD [BX+SI],AL
076A:0041 0000 ADD [BX+SI],AL
-u 22
076A:0022 49 DEC CX
076A:0023 75E9 JNZ 000E
076A:0025 B44C MOV AH,4C
076A:0027 CD21 INT 21
076A:0029 0031 ADD [BX+DI],DH
076A:002B 353033 XOR AX,3330
076A:002E 350000 XOR AX,0000
076A:0031 0000 ADD [BX+SI],AL
076A:0033 0000 ADD [BX+SI],AL
076A:0035 0000 ADD [BX+SI],AL
076A:0037 0000 ADD [BX+SI],AL
076A:0039 0000 ADD [BX+SI],AL
076A:003B 0000 ADD [BX+SI],AL
076A:003D 0000 ADD [BX+SI],AL
076A:003F 0000 ADD [BX+SI],AL
076A:0041 0000 ADD [BX+SI],AL
-g 25
AX=3ABB BX=0035 CX=0000 DX=0000 SP=0040 BP=0000 SI=0005 DI=0000
DS=076C ES=075A SS=076D CS=076A IP=0025 NV UP EI PL ZR NA PE NC
076A:0025 B44C MOV AH,4C
0x3abb = 15035 = the number in string
c : nvim
tmp : dosbox
c : cling