How to display output on the screen:

1. This program prints the character 'a':

```
                org 0x100
                bits 16
                jmp main
message:        db 'a'          ; message is the address of char 'a'
main:           mov bx, message ; move message (i.e. an address) to bx
                mov dl,[ bx ]   ; move contents of address pointed to
                                ; by bx to dl.  This is an example
                                ; of register INDIRECT addressing
                mov ah,02
                int 21h
                int 20h
```

Notes:

- mov ah,02 is used for displaying ONE character (in dl) only. If message was 'abc', just 'a' would be displayed.
- To avoid indirect addressing, if you knew that the offset of message was 3, you could type mov dl,[103h] instead. It would be even simpler to type mov dl,'a' if you don't need to give the character a label.
- Of AX, BX, CX, and DX, only BX can be used for indirect addressing.

2.This program prints a sentence:

```
                org 0x100
                bits 16
                jmp main
message:        db 'long string with no dollar/appostrophe characters!','$'
main:           mov dx,message  ; move the start address of message

                mov ah,09
                int 21h
                int 20h
```

Notes:

- The string MUST end with '$' - otherwise garbage follows in the output. An optional 0ah, 0dh provide a line feed and carriage return.
- mov dx,message could be replaced with mov dx,103h because the offset of message is 3.

Remember:
Character output - When ah contains 2, a *character* must be in dl.
String output   - When ah contains 9, an *address* must be in dx.

3. This program uses INC to increment a character:

```
                org 0x100
                bits 16

                mov dl,'q'
                inc dl                  ; increment a VALUE

                mov ah,02
                int 21h                 ; displays 'r'
                int 20h
```

4. This program uses INC to advance a start address by one:

```
                org 0x100
                bits 16
                mov dx,message
                inc dx                  ; increment an ADDRESS

                mov ah,09
                int 21h                 ; displays 'werty'
                int 20h

message:        db 'q','w','e','r','t','y','$'  ; same as 'qwerty$'
```

Notes:

- message may appear after the program!

Accepting input from the keyboard and displaying it:

```
                org 0x100
                bits 16
                jmp main

input:          db 20           ; length of the buffer
                db 0            ; no of bytes read (written here later)
                resb 21         ; bytes reserved for input characters
prompt1:        db 'Enter a word', 0ah, 0dh, '$'
prompt2:        db 13,10,'Enter a character', 0ah, 0dh, '$'

main:           mov dx,prompt1
                mov ah,09
                int 21h

                mov dx,input
                mov ah,0ah
                int 21h         ; a string has been read into dx;

                mov dl,13       ; carriage return
                mov ah,2
                int 21h

                mov dl,10       ; line feed
                mov ah,2
                int 21h

                mov dx,input
                inc dx          ; skip over the length of the buffer
                inc dx          ; skip over the no of bytes read
                mov bx,dx       ; bx points to start of input characters
                mov al,[input+1]; the length of the string
                mov ah,0        ; (this fills up ax)
                add bx,ax       ; bx points one char past last char entered
                mov al,'$'
                mov [bx],al     ; put a $ sign after the last character
                mov ah,09
                int 21h         ; the string in dx is displayed on the screen

                mov dx,prompt2
                mov ah,09
                int 21h

                mov ah,01
                int 21h         ; a character has been read into al

                mov dl,al       ; the character is moved to dl for display
                mov ah,02
                int 21h
                int 20h
```

Notes:

- When displaying the string that was read, you have to leave out the info stored in the first two bytes, and add a $ sign to the end, with the help of bx.

Simplifying the above program by using procedures:

```
                org 0x100
                bits 16
                jmp main

input:          db 20
                db 0
                resb 21

prompt1:        db 'Enter a word', 0ah, 0dh, '$'
prompt2:        db 13, 10, 'Enter a character', 0ah, 0dh, '$'

get_string:     mov ah,0ah
                int 21h
                ret

display_string: mov ah,09
                int 21h
                ret

get_char:       mov ah,01
                int 21h;
                ret

display_char:   mov ah,02
                int 21h
                ret

main:           mov dx,prompt1
                call display_string     ; prompt for a string

                mov dx,input
                call get_string         ; read in the string

                mov dl,13
                call display_char

                mov dl,10
                call display_char

                mov dx,input
                inc dx
                inc dx
                mov bx,dx
                mov al,[input+1]
                mov ah,0
                add bx,ax
                mov al,'$'
                mov [bx],al
                call display_string     ; display the string

                mov dx,prompt2
                call display_string     ; prompt for a character

                call get_char           ; read in the character

                mov dl,al
                call display_char       ; display the character

                int 20h
```

Notes:

- When you want to display a character, you can use either single quotes around the character, as in mov al,’2’, or a number which is the ASCII equivalent, as in mov al,50 (which is also the character ‘2’). This won’t work if you want to display a string that is in dx.

Useful table to memorise:

	Input	**Output**
char	mov ah,**01** (char is now pointed to by **al**)	mov ah,**02** (char must first be pointed to by **dl**)
string	mov ah,**10** (string is now pointed to by **dx**)	mov ah,**09** (string must first be pointed to by **dx**)

Converting numbers to strings:

These programs display numbers as strings on the screen. They need to be converted to ASCII values because if you try printing a number (using mov ah,02) you get the ASCII equivalent, not the actual number. Number characters in ASCII start at position 48 (or 30h), so that’s the amount you need to add to each digit. One digit is converted at a time, starting with the right-most one and using modulo 10 to extract it. With each loop iteration the last digit is added to the next-last position in the buffer. The buffer pointer must be decremented each time for this to happen.

This version of the program uses a 16-bit divisor (pointed to by bx):

```
                bits 16
                org 0x100
                jmp main

buffer:         db '     ', '$' ; 5 spaces for max number of 65535 (FFFF)

main:           mov di,buffer   ; (di can be used for string operations)
                add di,4        ; di now points to the last buffer position
                mov bx,10       ; the numbers are going to be divided by 10
                mov ax,65535    ; FFFF is the largest number you can use in ax

loop1:          xor dx,dx       ; clear dx (because idiv uses dx:ax here)
                                ; not clearing dx gives a division overflow
                idiv bx         ; divide dx:ax by bx
                                ; result in ax, remainder in dx
                add dx,30h      ; convert remainder to ASCII
                mov [di],dl     ; move the ASCII digit to the next last pos
                cmp ax,0        ; test if there are more digits left
                je finis
                dec di          ; point to a previous position
                jmp loop1

finis:          mov dx,buffer   ; dx points to the beginning of the buffer
                mov ah,09
                int 21h

                int 20h
```

This version of the program uses an 8-bit divisor (pointed to by bl):

```
                bits 16
                org 0x100
                jmp main

buffer:         db '   ', '$'   ; 3 spaces for max number of 255 (FF)
```

```
main:               mov di,buffer
                    add di,2
                    mov bl,10
                    mov al,255      ; FF is the largest number you can use in al

loop1:              xor ah,ah       ; clear ah (because idiv uses the entire ax)
                                    ; not clearing ah gives a division overflow
                    idiv bl         ; divide ax by bl
                                    ; result in al, remainder in ah
                    add ah,30h
                    mov [di],ah
                    cmp al,0
                    je finis
                    dec di
                    jmp loop1

finis:              mov dx,buffer
                    mov ah,09
                    int 21h

                    int 20h
```

Reading in a number as a string, converting it to a numeric value, adding 5, and then converting it to a string for display:

```
                    org 0x100
                    bits 16
                    jmp main

inputbuffer:        db 6;
                    db 0;
                    resb 6;

outputbuffer:       db '     ','$'          ; length of 5 chars

prompt:             db 'Enter a number: ','$'

str_to_num:         xor ax,ax               ; ax will point to the number
                    xor bh,bh               ; clear bh because bx is used later
                    mov cx,10
                    mov si,dx               ; si points to the start address

next_char:          mov bl,[si]             ; move the first character to bl
                    cmp bl,'$'              ; check if it is the last char
                    je finis
                    cmp bl,39h              ; character '9'
                    jg error
                    cmp bl,30h              ; character '0'
                    jl error
                    sub bl,30h              ; convert to ASCII numeric value
                    imul cx                 ; dx:ax = ax * 10 (shift digits left)
                    add ax,bx               ; add the character to ax
                    inc si                  ; point to the next character
                    jmp next_char

error:              mov al,'E'

finis:              ret

main:               mov dx,prompt           ; display prompt
                    mov ah,09
                    int 21h

                    mov dx,inputbuffer      ; read in the number string
```

```
                mov ah,10
                int 21h

                mov dl,13               ; carriage return
                mov ah,2
                int 21h

                mov dl,10               ; line feed
                mov ah,2
                int 21h

                mov dx,inputbuffer
                inc dx
                inc dx
                mov bx,dx
                mov al,[inputbuffer + 1]
                mov ah,0
                add bx,ax
                mov al,'$'
                mov [bx],al             ; now the string with '$' is in dx

                call str_to_num         ; afterwards, the number is in ax
                cmp al,69               ; test for error character 'E'
                je display_error
                add ax,5                ; add 5 before making it a string

                mov di,outputbuffer
                add di,4
                xor bx,bx
                mov bx,10

num_to_str:     xor dx,dx
                idiv bx                 ; divide dx:ax by bx
                add dx,30h              ; convert the remainder to ASCII
                mov [di],dl
                cmp ax,0
                je display_str
                dec di
                jmp num_to_str

display_str:    mov dx,outputbuffer
                mov ah,09
                int 21h

                int 20h

display_error:  mov dl,al
                mov ah,02
                int 21h

                int 20h
```

Notes:

- You can't read in or print numbers using their actual values - they are always treated as ASCII strings.
- The largest number that can be displayed is 65535. Because 5 is added to the input value, the largest number you can enter is 65530.

Using the sieve of Eratosthenes to find the first 1000 primes:

Algorithm: Initialise 1000 bytes to '1' and use nested loops to replace bytes in non-prime positions with '0'. The outer loop (i) runs from 2 to 32 (you only need to iterate till the square root of 1000!) and the inner loop (j) goes from

i to 1000, in increments of i, replacing these multiples of i with ‘0’ (because multiples of anything cannot be prime!).
The program prints the list of ‘1’s and ‘0’s. A ‘1’ indicates that the number in that position (starting from position 1, not 0) is prime. If it is ‘0’, the number in that position is not prime.

```
                        org 0x100
                        bits 16
                        jmp main

buffer:                 times 1000 db '1'
                        db '$'

start:                  mov ax,1             ; the outer-loop counter
                        mov cl,30h           ; char '0' to wipe out non-primes

outer_loop:             inc ax               ; for loop from 2 to 32
                        cmp ax,32
                        jge print_primes     ; quit after 31 iterations

                        mov bx,ax            ; start from current outer index
inner_loop:             add bx,ax            ; check every jth number above j
                        cmp bx,1000          ; see if you've reached the end
                        jg outer_loop
                        mov [bx+buffer-1],cl ; turn multiples of j into '0'
                        jmp inner_loop

print_primes:           mov dx,buffer
                        mov ah,09
                        int 21h

                        int 20h

main:                   jmp start
```

Finding the factorial of a number and displaying it:

```
                bits 16
                org 0x100
                jmp main

buffer          db '     $'     ; 5 spaces reserved for output (FFFF = 65535)

number:         dw 8            ; the largest number you can use (8! = 40320)
                                ; equivalent to db 8,0 (little endian!)

main:           mov di,buffer   ; this will store the output string later
                add di,4        ; point to the last space
                mov ax,[number] ; the number, and product so far = in dx:ax
                mov cx,ax       ; used to multiply, with each loop iteration
                dec cx          ; because you start multiplying by n - 1

repeat:         imul cx         ; multiply ax by cx - product in dx:ax
                loop repeat     ; decrement cx and loop till cx = 0

                mov bx,10       ; you're goint to extract 1 digit at a time
loop1:          xor dx,dx       ; clear dx to prevent a divide overflow error
                div bx          ; divide dx:ax by 10
                                ; quotient in ax, remainder in dx
                add dx,30h      ; convert to string for display
                mov [di],dl     ; move the digit to the right-most space
                dec di          ; point to the next right-most space
                cmp ax,0        ; more digits to divide?
                je display
                jmp loop1       ; else loop
```

```
display:        mov dx,buffer   ; modified buffer, thanks to di operations
                mov ah,09
                int 21h

                int 20h
```

Setting the time on your computer clock:

```
                bits 16
                org 0x100
                jmp main

main:           mov ch,20        ; hours
                mov cl,58        ; minutes
                mov dh,1         ; seconds
                mov dl,9         ; hundredths of a second

                mov ah,2dh
                int 21h
                int 20h
```

Notes:

- You can shorten the program with mov cx,nn and mov dx,nn instead
- Remember: mov ch,10h and mov cl,23h is the same as mov cx,1023h!

Using the stack to reverse user input:

```
                bits 16
                org 0x100
                jmp main

input_buffer:   db 21
buffer_len:     db 00
string:         resb 21

array:          times 20 db ' '         ; 20 spaces = max output length

cr_lf:          db 13, 10, '$'

main:           mov dx,input_buffer
                mov ah,0ah
                int 21h                 ; read in a string

                mov dx,cr_lf
                mov ah,9
                int 21h                 ; move to the next line

                mov si,string           ; (the user input)
                mov cl,[buffer_len]
                xor ch,ch               ; cx will be the loop counter
                xor ah,ah               ; ax will be pushed on the stack

loop1:          mov al,[si]             ; work with one char at a time
                push ax
                inc si                  ; point to the next char
                loop loop1

unpack:         mov di,array            ; point to the start of the array
                mov dx,di               ; for display, later on
                mov cl,[buffer_len]     ; set the loop counter
                xor ch,ch

loop2:          pop ax
```

```
                mov [di],al             ; move one char at a time
                inc di                  ; point to the next space
                loop loop2

                add di,[buffer_len]     ; point to the end of the string
                mov bx,'$'
                mov di,bx               ; place a $ at the end
                mov ah,09
                int 21h                 ; display the string in reverse

                int 20h
```

Note: The stack is implicitly used when a procedure is called (with ‘call’). The address of the instruction following the call is pushed onto the stack and when the ‘ret’ instruction is executed in the subroutine, it’s popped off. Remember: when you’re in a subroutine, the return address is always on top of the stack!

Something to remember: Bytes are stored in reverse order in memory!

Comparing LEA with MOV:

```
                org 0x100
                bits 16
                jmp main
buffer1:        db 'Hello', 13, 10,'$'
buffer2:        db 'Goodbye','$'

main:           mov dx,buffer1
                mov ah,09
                int 21h

                lea dx,[buffer2]   ; loads the effective address of buffer2
                mov ah,09
                int 21h

                int 20h
```

- mov dx,buffer and lea dx,[buffer] have the same effect, but these different instructions may occupy a different number of bytes. You can always insert ‘nop’ to fill in the gap.

Parameter passing:

There are three ways of passing parameters:
1. Store the parameters in a *parameter block* in memory
2. Store the parameters in *registers* before calling the subroutine
3. Push the parameters onto the *stack*

1. Using a parameter block in memory (calling by name):
The address of the block is stored in a register before the subroutine is called.

In this program, the procedure that accepts a string from the keyboard makes use of the ‘input_buffer’, by storing the length of the string entered, etc. This buffer is passed as a parameter by the statement ‘mov dx,input_buffer’, just before the procedure call.

```
                org 0x100
                bits 16
                jmp main

; parameter block:
input_buffer:   db 5              ; max 5 chars
```

```
                db 0            ; length of string entered
                resb 5          ; 5 bytes reserved as input buffer

; procedure:                    ; uses the parameter block in dx
accept_string:  mov ah,0ah
                int 21h
                ret

main:           mov dx,input_buffer     ; mov parameter block in dx first!
                call accept_string
                int 20h
```

<u>2. Passing parameters in registers (calling by value):</u>
The calling program must simply store the parameters in the relevant registers before calling the subroutine. On entering the subroutine, nothing needs to be done in the **prolog** because the parameters are already in the required registers. This mechanism is fast but can only be used when we have to pass a few values that will fit into registers.

<u>3. Passing parameters on the stack (calling by value):</u>
The return address is pushed onto the top of the stack when the subroutine is called, and popped off when the subroutine returns.

```
                org 0x100
                bits 16
                jmp main

output_buffer:  db '   $'

number_3:       dw 3            ; dw, not db (because AX etc. = 2 bytes!)
number_7:       dw 7
number_1:       dw 1

return_second:                  ; the procedure returns the second parameter
prolog:         pop ax          ; store the RETURN ADDRESS temporarily
                pop bx          ; pop the three parameters
                pop cx          ; into bx, cx and dx
                pop dx          ; for local use

body:           ; this is where you could have additional program statements

epilog:         push cx         ; cx points to the second parameter
                push ax         ; restore the RETURN ADDRESS
                ret             ; the 2nd parameter is returned on the stack

main:           mov ax,[number_3] ; store three different numbers
                mov bx,[number_7] ; which will be sent as parameters
                mov cx,[number_1] ; on the stack to the procedure

start_up:       push ax         ; push the 3 numbers onto the stack
                push bx         ; the procedure will access them
                push cx         ; by popping them off

                call return_second      ; the return address is IMPLICITLY
                                        ; pushed on the stack!! so when
                                        ; you enter the procedure it is
                                        ; popped off FIRST

                ; the return address was implicitly popped off, so now:
clean_up:       pop dx          ; pop the return value off the stack into dx
                add dl,30h      ; convert to ascii string
                mov di,output_buffer
                mov [di],dl     ; move the digit into the buffer
                mov dx,output_buffer
                mov ah,09h
```

```
 int 21h          ; prints 7
int 20h
```