```
[org 0x0100]

; start of code

mov  ax, 5              ; move the constant 5 into register ax
mov  bx, 10

add  ax, bx             ; add value of bx into the value of ax

mov  bx, 15             ; add constant 15 into the value of bx
add  ax, bx

mov  ax, 0x4c00         ; exit ..
int  0x21               ; .. is what the OS should do for me
```

```
; watch the listing carefully

; a program to add three numbers using memory variables
[org 0x0100]

    mov  ax, [num1]         ; load first number in ax
    ; mov  [num1], [num2]     ; illegal
    mov  bx, [num2]
    add  ax, bx
    mov  bx, [num3]
    add  ax, bx
    mov  [num4], ax
    mov  ax, 0x4c00
    int  0x21


num1: dw   5
num2: dw   10
num3: dw   15
num4: dw   0
```

```
; watch the listing carefully

; a program to add three numbers accessed using a single label
[org 0x0100]

    mov  ax, [num1]
    mov  bx, [num1 + 2]     ; notice how we can do arithmetic here
    add  ax, bx             ; also, why +2 and not +1?
    mov  bx, [num1 + 4]
    add  ax, bx
    mov  [num1 + 6], ax     ; store sum at num1+6
    mov  ax, 0x4c00
    int  0x21

num1:   dw  5
        dw  10
        dw  15
        dw  0
```

```

; a program to add three numbers accessed using a single label
[org 0x0100]

    mov  ax, [num1]
    mov  bx, [num1 + 2]
    add  ax, bx
    mov  bx, [num1 + 4]
    add  ax, bx
    mov  [num1 + 6], ax
    mov  ax, 0x4c00
    int  0x21

num1:   dw  5,  10,  15,  0


; a program to add three numbers directly in memory
[org 0x0100]

    mov  ax, [num1]
    mov  [num1 + 6], ax    ; add this value to result

    mov  ax, [num1 + 2]
    add  [num1 + 6], ax

    mov  ax, [num1 + 4]
    add  [num1+6], ax

    mov  ax, 0x4c00
    int  0x21


num1:   dw  5, 10, 15, 0


; should have the result separate!
; let's change that!


; a program to add three numbers using byte variables
[org 0x0100]

    mov  ax, [num1]

    mov  bx, [num1+1]
    add  ax, bx

    mov  bx, [num1+2]
    add  ax, bx

    mov  [num1+3], ax

    mov  ax, 0x4c00
    int  0x21

num1: db  5, 10, 15, 0

; something's wrong with this code.
; let's figure out what that is!


; a program to add three numbers using byte variables
[org 0x0100]
    ; mov  ax, 0x8787
    ; xor  ax, ax             ; We need to make sure AX is empty! Or do we?

    mov  ah, [num1]         ; Intel Sotware Developer Manual - Figure 3-5 (Page 76)

    mov  bl, [num1+1]
    add  ah, bh

    mov  bh, [num1+2]
    add  ah, bh

    mov  [num1+3], ah

    mov  ax, 0x4c00
    int  0x21

num1: db  5, 10, 15, 0


; a program to add three numbers using byte variables
[org 0x0100]

    mov  ax, 0x8787
    xor  ax, ax          ; we need to make sure AX is empty!

    mov  al, [num1]

    mov  bl, [num1+1]
    add  al, bl

    mov  bl, [num1+2]
    add  al, bl

    mov  [num1+3], al



    ; mov  ax, bl         ; ... assemble time error. Make sure you understand the error!

    mov  ax, 0x4c00
    int  0x21

num1: db  5, 10, 15, 0


; a program to add three numbers using byte variables
[org 0x0100]
    xor  ax, ax                 ; check effect on ZF

    mov  bx, num1

    add  ax, [bx]
    add  bx, 2

    add  ax, [bx]
    add  bx, 2

    add  ax, [bx]
    add  bx, 2


    mov  [result], ax

    mov  ax, 0x4c00
    int  0x21


    ; to turn this into an iteration, we need a couple of things:
    ; - branching instruction
    ; - checking constraints -- e.g. c > 0       ; Intel Sotware Developer Manual - Figure 3-8 (Page 80)


num1: dw  5, 10, 15
result: dw 0


; a program to add three numbers using byte variables
[org 0x0100]

     ; for (int c = 3    c > 0     c--) {
     ;   result += data[c];
     ;}



    ; initialize stuff
    mov  ax, 0              ; reset the accumulator
    mov  cx, 3              ; set the iterator count
    mov  bx, num1           ; set the base

    outerloop:
        add  ax, [bx]
        add  bx, 2

        sub  cx, 1
        jnz  outerloop


    mov  [result], ax

    mov  ax, 0x4c00
    int  0x21


    ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)

num1: dw  5, 10, 15
result: dw 0


; a program to add three numbers using byte variables
[org 0x0100]

    ; initialize stuff
    mov  ax, 0              ; reset the accumulator
    mov  cx, 10              ; set the iterator count
    mov  bx, 0           ; set the base

    outerloop:
        add  ax, [num1 + bx]
        add  bx, 2

        sub  cx, 1
        jnz  outerloop


    mov  [result], ax

    mov  ax, 0x4c00
    int  0x21


    ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)

num1: dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
result: dw 0


; a program to add three numbers using byte variables
[org 0x0100]

    ; initialize stuff
    mov  ax, 0              ; reset the accumulator
    mov  bx, 0              ; set the counter

    outerloop:
        add  ax, [num1 + bx]
        add  bx, 2

        cmp bx, 20          ; sets ZF=0 when they are equal
        jne  outerloop


    mov  [result], ax

    mov  ax, 0x4c00
    int  0x21


    ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)

num1: dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
result: dw 0


[org 0x0100]

    jmp start       ; see next instructions when you haven't yet executed this!

    num1: dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
    result: dw 0



    start:
    ; initialize stuff
    mov  ax, 0              ; reset the accumulator
    mov  bx, 0              ; set the counter

    outerloop:
        add  ax, [num1 + bx]
        add  bx, 2

        cmp bx, 20          ; sets ZF=0 when they are equal
        jne  outerloop


    mov  [result], ax

    mov  ax, 0x4c00
    int  0x21



[org 0x0100]

jmp start

data: dw   6, 4, 5, 2


start:
    mov  cx, 4                          ; make 4 passes, has to be outside the loop!

    outerloop:
        mov  bx, 0

        innerloop:
            mov  ax, [data + bx]
            cmp  ax, [data + bx + 2]    ; why did we move the value to AX?

            jbe  noswap                 ; if we don't have to swap, we just jump over the swap thing
                                        ; think of this as the "if"

                ; the swap potion
                mov  dx, [data + bx + 2]
                mov  [data + bx + 2], ax    ; again with the AX?
                mov  [data + bx], dx

            noswap:
            add  bx, 2
            cmp  bx, 6
            jne  innerloop

        ; check outer loop termination
        sub cx, 1
        jnz outerloop


    ; exit system call
    mov  ax, 0x4c00
    int  0x21



[org 0x0100]

jmp start

data: dw   6, 2, 4, 5
swap: db   0    ; use this as a flag

start:
    ; mov  cx, 4                          ; make 10 passes, has to be outside the loop!

    outerloop:
        mov  bx, 0
        mov  byte [swap], 0             ; why the "byte"?

        innerloop:
            mov  ax, [data + bx]
            cmp  ax, [data + bx + 2]    ; why did we move the value to AX?

            jbe  noswap                 ; if we don't have to swap, we just jump over the swap thing

                ; the swap potion
                mov  dx, [data + bx + 2]
                mov  [data + bx + 2], ax    ; again with the AX?
                mov  [data + bx], dx
                mov  byte [swap], 1

            noswap:
            add  bx, 2
            cmp  bx, 6
            jne  innerloop

        ; if we didn't swap even once, we should be done
        cmp  byte [swap], 1     ; don't need to load this in register?
        je   outerloop

        ; check outer loop termination
        ; sub cx, 1
        ; jnz outerloop


    ; exit system call
    mov  ax, 0x4c00
    int  0x21



[org 0x0100]

jmp start

multiplicand: db 13      ; 4-bit number, save space of 8-bits
multiplier:   db 5       ; 4-bit

result:       db 0       ; 8-bit result
```

```

start:

    mov  cl, 4            ; how many times we need to run the loop
    mov  bl, [multiplicand]
    mov  dl, [multiplier]


    checkbit:
        shr  dl, 1        ; do the rotation so that right bit is thrown in CF
        jnc  skip
            add [result], bl     ; only add if CF IS SET


        skip:
        shl  bl, 1         ; always shift the multiplicand

    dec  cl
    jnz  checkbit


    mov  ax, 0x4c00
    int  0x21



[org 0x0100]

jmp start

num1:   dw 0x40FF    ; 4400,  40FF

dest:   dw 0x40FF
src:    dw 0x1001


start:

    ; shift
    shl byte [num1], 1
    rcl byte [num1 + 1], 1







    ; addition
    xor ax, ax   ; clear

    mov al, byte[src]
    add byte[dest], al

    mov al, [src  + 1]
    adc byte[dest + 1], al



    mov  ax, 0x4c00
    int  0x21


[org 0x0100]

jmp start

multiplicand:   dw 0xC8    ; 200  =   0b 11001000
multiplier:     db 0x32    ; 50   =   0b 00110010
result:         dw 0       ; should be 10,000 = 0x2710

start:

mov  cl, 8
mov  dl, [multiplier]


checkbit:
    shr  dl, 1
    jnc  skip

        mov  al, [multiplicand]      ; extended addition
        add  byte [result], al
        mov  al, [multiplicand + 1]
        adc  byte [result + 1], al

    skip:
    shl  byte [multiplicand], 1      ; extended shift
    rcl  byte [multiplicand + 1], 1


    dec  cl
    jnz  checkbit




; exit syscall
mov  ax, 0x4c00
int  0x21





; helpful bash commands

; hex to dec
; echo $((16#   F))

; dec to hex
; printf '%x\n'   15

; bin to hex
; printf '%x\n' "$((2#   110010))"

; hex to bin
; printf   '\x32'    | xxd -b | cut -d' ' -f2


; Let's run a 32-bit program in Ubuntu!

; Install NASM in Ubuntu:
;   sudo apt install nasm

; Create this code file

; Assemble:
;   nasm -f elf32 -l c05-01.lst -o c05-01.o c05-01.asm
;
;   We want to create a format that Linux understand
; i.e. ELF format in 32-bits
;   (we also create a listing file)
;   Read more about ELF here: https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/

; Link with shared library that 'understands' the format: ld.so in Linux
;   ld -m elf_i386 -o c05-01 c05-01.o

; Run it:
;   ./c05-01



; Now let's discuss the code!

; in modern OSs, programs do not start executing
; "from the first instruction"

; Instead, there is a library (ld.so) that looks for the "start symbol"
; and executes from there.


; a section "directive" marks the parts of a program
; for the ELF format  (or whatever binary format you are using)
SECTION .text:

; We mark the start for this library using the following:
GLOBAL _start

_start:
  ; write the string to console
  mov eax, 0x4        ; write syscall is 0x4
  mov ebx, 1          ; param - std output should be used
  mov ecx, message      ; the string to write
  mov edx, message_length   ; the length of the string
  int 0x80          ; invoke the system call


  ; exit the program
  mov eax, 0x1        ; exit system call is 0x1
  mov ebx, 0          ; exit code is 0 (return 0)
  int 0x80          ; Comment out and see!

  ; note that int is NOT the right way to do things!
  ; (more on this later)


; data section here. We can also move it above .code
```

```
SECTION .data:
  ; 0xA is new line, 0x0 is null terminator
  message: db "Hello!",  0xA,  0x0
  message_length: equ $-message

  ; message_length: equ 8
  ; .... is exactly the same as
  ; #define message_length 8



; Some useful ELF details
; readelf -a c05-01.o     ; shows everything

; readelf -h c05-01.o   ; shows headers
; readelf -S c05-01.o       ; shows sections

; readelf -x 2 c05-01.o     ; shows section number 2
; readelf -x 2 c05-01     ; see the difference between above and this





; View program in GDB

; gdb ./c05-01
; layout regs       ; shows registers and disassembled code
; starti      ; start the program interactively
; si        ; execute one machine instruction
; quit        ; exit GDB



[org 0x100]

jmp start

data:   dw  60, 55, 45, 50
swap:   db  0


bubblesort:
    dec  cx
    shl  cx, 1                   ; we will be jumping by 2 every time. So, *2

    mainloop:
        mov  si, 0               ; use as array index
        mov  byte[swap], 0       ; reset swap flag for this iteration

        innerloop:
            mov  ax, [bx + si]
            cmp  ax, [bx + si + 2]
            jbe  noswap

                mov  dx, [bx + si + 2]
                mov  [bx + si], dx
                mov  [bx + si + 2], ax
                mov  byte[swap], 1

            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop

        cmp  byte[swap], 1
        je   mainloop

    ret    ; notice this!!




start:
    mov  bx, data
    mov  cx, 4

    ; make a function call
    call bubblesort

    ; data is now sorted!

    mov ax, 0x4c00
    int 0x21


[org 0x100]

jmp start

data:   dw  60, 55, 45, 50
swapflag:   db  0


swap:
    mov  ax, [bx + si]          ; this changes ax
    xchg ax, [bx + si + 2]
    mov  [bx + si], ax

    ret


bubblesort:
    dec  cx
    shl  cx, 1                   ; This changes cx

    mainloop:
        mov  si, 0               ; This changes si
        mov  byte[swapflag], 0

        innerloop:
            mov  ax, [bx + si]   ; This changes ax
            cmp  ax, [bx + si + 2]
            jbe  noswap

               call swap          ; another call here
               mov  byte[swapflag], 1

            noswap:
```

```
            add  si, 2
            cmp  si, cx
            jne  innerloop

        cmp  byte[swap], 1
        je   mainloop

    ret    ; notice this!!




start:
    mov  bx, data
    mov  cx, 4

    ; make a function call
    call bubblesort

    ; data is now sorted!

    mov ax, 0x4c00
    int 0x21


[org 0x100]

jmp start

data:   dw  60, 55, 45, 50
swapflag:   db  0


swap:
    push ax  ; -------------------------;
    ; push cx  ; ------------------;      ;
                                 ;      ;
    mov  ax, [bx + si]           ;      ;
    xchg ax, [bx + si + 2]       ;      ;
    mov  [bx + si], ax           ;      ;
                                 ;      ;
    dec  cx                      ;      ;
    ; do some storage here       ;      ;
    ; pop cx   ; ------------------;      ;
    pop ax   ; -------------------------;

    ret


bubblesort:
    push ax         ; three new pushes
    push cx
    push si


    dec  cx
    shl  cx, 1

    mainloop:
        mov  si, 0                  ; use as array index
        mov  byte[swapflag], 0       ; reset swap flag for this iteration

        innerloop:
            mov  ax, [bx + si]
            cmp  ax, [bx + si + 2]
            jbe  noswap

               call swap    ; another call here
               mov  byte[swapflag], 1

            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop

        cmp  byte[swap], 1
        je   mainloop


    ; pops in reverse order
    pop si
    pop cx
    pop ax
    ret    ; notice this!!




start:
    mov  bx, data
    mov  cx, 4

    ; make a function call
    call bubblesort

    ; data is now sorted!

    mov ax, 0x4c00
    int 0x21


[org 0x100]

jmp start

data:   dw  60, 55
swapflag:   db  0


swap:
    push ax  ; -----------------------;
                                      ;
    mov  ax, [bx + si]                ;
    xchg ax, [bx + si + 2]            ;
    mov  [bx + si], ax                ;
                                      ;
    pop ax   ; -----------------------;

    ret

```

```

bubblesort:
    ; handle stack issue for parameters -------------
    push bp
    mov  bp, sp

    push ax
    push bx
    push cx
    push si

    mov  bx, [bp + 6]   ; address of data to sort
    mov  cx, [bp + 4]   ; number of elements to sort

    ; same old code from here -----------------------
    dec  cx
    shl  cx, 1

    mainloop:
        mov  si, 0                   ; use as array index
        mov  byte[swapflag], 0       ; reset swap flag for this iteration

        innerloop:
            mov  ax, [bx + si]
            cmp  ax, [bx + si + 2]
            jbe  noswap

               call swap    ; another call here
               mov  byte[swapflag], 1

            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop

        cmp  byte[swap], 1
        je   mainloop


    ; handle parameter stack issue at end again -------------------
    pop si
    pop cx
    pop bx       ; check removal
    ; pop ax
    pop bp       ; bp was the first thing pushed, so last popped!
    ; stack cleared? ----------------------------------------------

    ret 4        ; what is this guy?



start:
    mov  bx, data
    mov  cx, 2

    push bx
    push cx
    ; make a function call
    call bubblesort

    ; data is now sorted!

    mov ax, 0x4c00
    int 0x21


[org 0x100]

jmp start

data:   dw  60, 55
; swapflag:   db  0             ; Globals are bad! Let's make this local.


swap:
    push ax  ; -----------------------;
                                      ;
    mov  ax, [bx + si]                ;
    xchg ax, [bx + si + 2]            ;
    mov  [bx + si], ax                ;
                                      ;
    pop ax   ; -----------------------;

    ret


bubblesort:
    ; handle stack issue for parameters -------------
    push bp
    mov  bp, sp

    sub sp, 2           ; make space on the stack, just below BP
                        ; only if you want to do local variables


    push ax
    push bx
    push cx
    push si

    mov  bx, [bp + 6]   ; address of data to sort
    mov  cx, [bp + 4]   ; number of elements to sort

    ; same old code from here ----------------------
    dec  cx
    shl  cx, 1

    mainloop:
        mov  si, 0                     ; use as array index
        ; mov  byte[swapflag], 0       ; reset swap flag for this iteration
        mov  word [bp - 2], 0             ; has to be a word

        innerloop:
            mov  ax, [bx + si]
            cmp  ax, [bx + si + 2]
            jbe  noswap

               call swap    ; another call here
               ; mov  byte[swapflag], 1
               mov  word [bp - 2], 1

            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop

        cmp  word [bp - 2],  1
        je   mainloop


    ; handle parameter stack issue at end again -------------------
    pop si
    pop cx
    pop bx
    pop ax

    mov sp, bp   ; sp should be restored

    pop bp       ; bp was the first thing pushed, so last popped!
    ; stack cleared? ----------------------------------------------

    ret 4        ; what is this guy?



start:
    mov  bx, data
    mov  cx, 2

    push bx
    push cx
    ; make a function call
    call bubblesort

    ; data is now sorted!

    mov ax, 0x4c00
    int 0x21


[org 0x0100]

mov  ax, 0xb800         ; video memory base
mov  es, ax             ; cannot move to es through IMM
mov  di, 0              ; top left location

nextpos:
    mov  word [es:di], 0x0776       ; 0x07 -- full white  (try 41)
                                    ; 0x20 is the space character
    add  di, 2
    cmp  di, 4000
    jne  nextpos

    mov  ax, 0x4c00
    int  0x21


[org 0x0100]

    jmp start
```

```

message:     db   'hello world'
length:      dw   11

clrscr:
    push es
    push ax
    push di

    mov  ax, 0xb800
    mov  es, ax
    mov  di, 0

    nextloc:
        mov  word [es:di], 0x0720
        add  di, 2
        cmp  di, 4000
        jne  nextloc

    pop  di
    pop  ax
    pop  es
    ret


printstr:
    push bp
    mov  bp, sp
    push es
    push ax
    push cx
    push si
    push di

    mov ax, 0xb800
    mov es, ax
    mov di, 0


    mov si, [bp + 6]
    mov cx, [bp + 4]
    mov ah, 0x07 ; only need to do this once

    nextchar:
        mov al, [si]
        mov [es:di], ax
        add di, 2
        add si, 1

        ; dec cx
        ; jnz nextchar

        ; alternatively
        loop nextchar


    pop di
    pop si
    pop cx
    pop ax
    pop es
    pop bp
    ret 4


start:
    call clrscr

    mov ax, message
    push ax
    push word [length]
    call printstr



    ; wait for keypress
    mov ah, 0x1        ; input char is 0x1 in ah
    int 0x21

    mov ax, 0x4c00
    int 0x21


[org 0x0100]

    jmp start

clrscr:
    push es
    push ax
    push di

    mov  ax, 0xb800
    mov  es, ax
    mov  di, 0

    nextloc:
        mov  word [es:di], 0x0720
        add  di, 2
        cmp  di, 4000
        jne  nextloc

    pop  di
    pop  ax
    pop  es
    ret


printnum:
    push bp
    mov  bp, sp
    push es
    push ax
    push bx
    push cx
    push dx
    push di

    ; first, let's split digits and push them onto the stack

    mov ax, [bp+4]   ; number to print
    mov bx, 10       ; division base 10
    mov cx, 0        ; total digit counter

    nextdigit:
        mov dx, 0    ; zero out
        div bx       ; divides ax/bx .. quotient in ax, remainder in dl
        add dl, 0x30 ; convert to ASCII
        push dx      ; push to stack for later printing
        inc cx       ; have another digit
        cmp ax, 0    ; is there something in quotient?
        jnz nextdigit

    ; now let's do the printing

    mov ax, 0xb800
    mov es, ax

    mov di, 0
    nextpos:
        pop dx          ; digit to output. Already in ASCII
        mov dh, 0x04    ; why is this inside the loop here?
        mov [es:di], dx
        add di, 2
        loop nextpos    ; cx has already been set, use that
        ;dec cx
        ;jnz nextpos

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp
    ret 2



start:
    call clrscr

    mov ax, 452
    push ax
    call printnum


    ; wait for keypress
    mov  ah, 0x1        ; input char is 0x1 in ah
    int 0x21

    mov ax, 0x4c00
    int 0x21


[org 0x0100]
    jmp  start

clrscr:
    push es
    push ax
    push cx
    push di

    mov  ax, 0xb800             ; same as before
    mov  es, ax

    xor  di, di                 ; starting at index 0

    mov  ax, 0x0720             ; what to write
    mov  cx, 2000               ; how many times to write
                                ; holds the count, NOT bytes!

    cld                         ; auto-increment
    rep stosw                   ; automatically writes starting from [es:di]

    pop di
    pop  cx
    pop  ax
    pop  es
    ret

start:
    call clrscr
    mov  ax, 0x4c00
    int  0x21



[org 0x0100]

jmp  start

clrscr:
    push es
    push ax
    push cx
    push di

    mov  ax, 0xb800
    mov  es, ax
    xor  di, di
    mov  ax, 0x0765
    mov  cx, 2000

    cld                 ; auto-increment mode
    rep stosw           ; rep cx times, store words
                        ; source is ax for word, al for bytes
                        ; destination is es:di
                        ; inc/dec di as well by 2 bytes

    pop di
    pop  cx
    pop  ax
    pop  es
    ret


start:

    call clrscr
    mov  ax, 0x4c00
    int  0x21


[org 0x0100]

    jmp start

message: db 'hello world', 0

clrscr:
    push es
    push ax
    push cx
    push di

    mov  ax, 0xb800
    mov  es, ax
    xor  di, di
    mov  ax, 0x0720
    mov  cx, 2000

    cld                 ; auto-increment mode
    rep stosw           ; rep cx times, store words
                        ; source is ax for word, al for bytes
                        ; destination is es:di
                        ; inc/dec di as well by 2 bytes

    pop  di
    pop  cx
    pop  ax
    pop  es
    ret

printnum:
    push bp
    mov  bp, sp
    push es
    push ax
    push bx
    push cx
    push dx
    push di

    ; first, let's split digits and push them onto the stack

    mov ax, [bp+4]   ; number to print
    mov bx, 10       ; division base 10
    mov cx, 0        ; total digit counter

    nextdigit:
        mov dx, 0    ; zero out
        div bx       ; divides ax/bx .. quotient in ax, remainder in dl
        add dl, 0x30 ; convert to ASCII
        push dx      ; push to stack for later printing
        inc cx       ; have another digit
        cmp ax, 0    ; is there something in quotient?
        jnz nextdigit

    ; now let's do the printing

    mov ax, 0xb800
    mov es, ax

    mov di, 0
    nextpos:
        pop dx          ; digit to output. Already in ASCII
        mov dh, 0x07    ; why is this inside the loop here?
        mov [es:di], dx
        add di, 2
        loop nextpos    ; cx has already been set, use that

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp
    ret 2


strlen:
    push bp
    mov  bp,sp
    push es
    push cx
    push di

    les  di, [bp+4]     ; load DI from BP+4 and ES from BP+6
    mov  cx, 0xffff     ; maximum possible length

    xor  al, al         ; value to find
    repne scasb         ; repeat until scan does not become NE to AL
                        ; decrement CX each time

    mov  ax, 0xffff
    sub  ax, cx         ; find how many times CX was decremented

    dec  ax             ; exclude null from the length

    pop  di
    pop  cx
    pop  es
    pop  bp
    ret  4


start:
    call clrscr

    push ds
    mov  ax, message
    push ax
    call strlen          ; return value is in AX

    push ax
    call printnum        ; print out the length


    mov  ah, 0x1
    int 0x21
    mov  ax, 0x4c00
    int 0x21


SECTION .DATA
  hello:     db 'Hello from ASM!',10
  helloLen:  equ $-hello

SECTION .TEXT
GLOBAL say_hi


say_hi:
    mov rax, rdi          ; first param goes in RDI
  push rax          ; save the value sent to us

  mov eax, 4            ; write()
  mov ebx, 1            ; STDOUT
  mov ecx, hello
  mov edx, helloLen

  int 80h              ; Interrupt

  pop rax        ; get the value sent to us
  inc rax        ; increment it
  ret                  ; return val is in rax



# Assemble using: nasm -f elf64 c09-01.asm  -o c09-01-asm.o
```