MASM's EQUATE directive

CR EQU 0dh

Tells MASM to replace every occurence of CR with 0dh in the program.

Lots of additional uses that we will see (MACROS)

Function 0Ah of int 21h

Gets a string and stores it in a buffer

Prog4_3.asm in the book

```
.STACK  100h

.DATA
        buf     db      80 dup('$')
        NEWLINE db      0Dh, 0Ah, 24h

.CODE
EXTRN   EXIT:NEAR, READ:NEAR, WRITE:NEAR
START:
        mov     ax, @DATA
        mov     ds, ax

        mov         dx, OFFSET buf    ;instead of "lea dx, buf"
        call    READ
        mov     cx, dx

        mov     dx, OFFSET NEWLINE
        call    WRITE
        .
        .
        mov         bx, 0
WhileLabel:
        mov     al, buf[bx]
                    ; = buf[0] like an array reference

;       mov         al, buf[0]
        ; will also work
;       mov         al, buf[5]
        ; will move 6th byte into AL
```

```
Consider
        mov     [bx], 0

Are we moving a byte or a word or a double word?

        mov     BYTE PTR [bx], 0
; Move a BYTE

        mov     WORD PTR [bx], 0
; Move a WORD

.DATA
        buf     db      80 dup('$')
        .
.CODE
        .
        mov     buf[0], 07h             ; This will work
; MASM looks at the SYMBOL TABLE entry for "buf"
; and realizes that it is "db" a sequence of bytes
; By Default MASM then assumes that you will be moving
; bytes

        mov     buf[0], 07h
        mov     buf[1], 1fh

        OR

        mov     WORD PTR buf[0], 1f07h
WORD PTR overides the default size of buf
```

```
TITLE Prog5_1.asm   ASCIIZ String Handling

LEN    EQU          10
CR     EQU          0dh
LF     EQU          0ah

DOSSEG
.MODEL SMALL

.STACK 100h

.DATA
                String     DB (LEN + 1) dup(?)
.CODE
Start:
        mov     ax, @DATA
        mov     ds, ax

        mov     bx, OFFSET String
                           ;point bx to String
        mov     ah, 01h
                           ;prepare to use function 01h
                           ; of int 21h
getLoop:
        int     21h
        cmp     al, CR                ;look for a CR in al
        je      getEnd
        mov     [bx], al
                           ;bx points to storage location
        inc     bx
        jmp     getLoop
```

```
getEnd:
        mov     BYTE PTR [bx], 0
                        ;CR is converted to null term.

        mov     ah, 02h                 ;these five lines print a
        mov     dl, CR                  ; newline, i.e. a CR
        int     21h                     ;  followed by a LF
        mov     dl, LF
        int     21h

        mov     bx, OFFSET String
                        ;point bx to start of String

nextChar:
        cmp     BYTE PTR [bx], 0        ;have we reached the nu
ll
        je      endNextChar             ;  terminator yet?
        mov     dl, [bx]                ;keep printing until
        int     21h                     ; we reach it
        inc     bx
        jmp     nextChar
endNextChar:
        mov     ax, 4c00h
;program termination with 00h in al as the "return code"
        int     21h

; Function 4Ch terminates program, but 4C00h ??
; 4C in AH, 00 in AL
DOS: System error level
00      ==      no error
Function 4C expects error level in AL and gives it to DOS
```

Procedures:

```
procName        PROC    NEAR
        ; code for procedure
        ret
procName        ENDP
        OR
procName        PROC    FAR
        ; code for procedure
        ret
procName        ENDP
```

Keywords: PROC, NEAR, FAR, ENDP

NEAR Callee procs are in the same segment as calling proc

FAR Callee procs are in a different segment from calling proc

NEAR CALL

1) pushes IP on stack
2) Transfers control to Callee

FAR CALL

1) Pushes CS and IP on stack (CS pushed first)
2) Transfers control to Callee

RET instruction
----

1) Pops Return Address
2) Transfers control to that address

RET knows whether to pop only IP or both CS and IP since it knows whether the procedure is a NEAR or FAR

```
TITLE Prog5_2.asm  An Improved Version Using Procedures
        LEN    EQU    80
        CR     EQU    0dh
        LF     EQU    0ah
DOSSEG
.MODEL SMALL

.STACK 100h

.DATA
        Prompt1    DB       "Input a String:", 0
        Prompt2    DB       "Do another?  ", 0
        String     DB       (LEN + 1) dup(?)
.CODE
;---------------------------------------------------
Start:

        mov     ax, @DATA
        mov     ds, ax

Begin:
        mov     bx, OFFSET Prompt1
                        ;point bx to the string to print
        call    PutStr

        call    newLine
        call    newLine

        mov     bx, OFFSET String
                        ;point bx to the input buffer
        call    getStr
```

```
        call    newLine
        call    newLine

        mov     bx, OFFSET String
        call    putStr

        call    newLine
        call    newLine

        mov     bx, OFFSET Prompt2
        call    putStr

        mov     bx, OFFSET String
        call    getStr

        call    newLine
        call    newLine

        mov     bx, OFFSET String
        cmp     BYTE PTR [bx], 'y'
        je      Begin

        mov     ax, 4c00h
        int     21h

;-------------------------------------------------
```

```
getStr PROC NEAR
        mov     ah, 01h
                                ;prepare to use function 01h
                                ; of int 21h
getLoop:
        int     21h
        cmp     al, CR                  ;look for a CR in al
        je      getEnd
        mov     [bx], al
                                ;bx points to storage location
        inc     bx
        jmp     getLoop
getEnd:
        mov     BYTE PTR [bx], 0
                                ;CR is converted to null term.
        ret
getStr  ENDP
;-----------------------------------------------------
putStr PROC NEAR                ;expects bx to point to string
        mov     ah, 2h          ;prepare to print using int 21h
nextChar:
        cmp     BYTE PTR [bx], 0h
                                ;check for null terminator
        je      foundEnd                ;when found, exit
        mov     dl, [bx]                ;print with int 21h
        int     21h
        inc     bx                      ;point to next char
        jmp     nextChar
foundEnd:
        ret
putStr ENDP
```

```
newLine PROC NEAR

        mov     ah, 02h   ;these five lines give a newline
        mov     dl, CR    ;  i.e. a CR followed by a LF.
        int     21h
        mov     dl, LF
        int     21h

        ret

newLine ENDP

;---------------------------------------------

END Start
```

```
TITLE Prog5_3.asm Passing Parameters on the Stack.
        .
Begin:
        mov     ax, OFFSET Prompt1
        push    ax
        call    putStr
        .
        mov     ax, OFFSET String
        push    ax
        call    getStr
        .
        .
        mov   ax, 4c00h
        int   21h
;-------------------------------------------------
getStr PROC NEAR
        push    bp
        mov     bp, sp
        push    ax
        push    bx

        mov     bx, [bp + 4]  ;base address of string buffer
        mov     ah, 01h
                ;prepare to use function 01h of int 21h
        ....
        pop     bx
        pop     ax
        pop     bp
        ret     2
getStr ENDP
END Start
```

Using the stack to pass parameters

```
        mov     ax, OFFSET STRING
        push    ax
        CALL    getStr
```

What does the stack look like now?
In getStr
First instructions are

```
        push    bp
        mov     bp, sp
```

These two lines will be used in EVERY procedure that gets its parameters on the stack.

Recall
When BX is used as a pointer [BX], its contents are considered an offset from DS (data segment)

When BP is used as a pointer [BP], its contents are considered an offset from SS (STACK SEGMENT!!)
[BP + displacement] can also be used
BP = 80h
SS = 123Ah
[BP + 6] references memory location 123A:0086
[BP - 6] references memory location 123A:007A
CONVENTION

ALL procedures MUST save (on stack) and restore (from stack) ALL registers that they use.