

comment <-----

NewBasic Assembler
Small Tutorial
Freeware

Forever Young Software
(C)opyright 1984-2001
All rights reserved
19 Mar 2001

This is a small tutorial on how to program in assembler. All the code, unless otherwise stated, is assembled with NBASM. This tutorial in no way will teach you the complex language of assembler. However, it is my goal to teach you the basics of assembler and how to you NBASM, or another assembler, to create your programs. We will discuss how to create workable, productable, .COM files. Once you know how to create .COM files, the move to .OBJ files is not to difficult.

I think the first thing we will discuss is program layout. How does DOS load and execute your program, what are the limits to .COM files, and what are the capabilities of .COM files.

DOS first allocates all available memory and creates a 256 byte block of memory at the beginning of this block. The block is called the PSP. The Program Segment Prefix is filled with information that is needed by DOS as well as your program.

Then DOS loads a mirror image of your .COM file in to memory just after this PSP and points CS:IP to your code (100h). DS, ES, and SS are all pointed to the same segment that CS points to. Since this happens, your .COM file can only be one segment in length (65536) and all code, data, and stack must be in this segment. For this reason, this puts a few restrictions on a .COM file:

1. - A valid .COM file can only be 65536 - 256 bytes in length. Since DOS places a 256 byte PSP at the first of the segment, your program can only by up to 65280 bytes in length.
2. - Since SS:SP points to the last word in the segment, if you intend to use the stack, you can not occupy the area at the end of this segment or your stack will overwrite it.
3. - Since DOS points CS:IP to offset 100h, you must have a valid asm instruction at offset 100h. This means the first thing in your program must be a valid code instruction.
4. - As mentioned before, all your code and data must be in this one segment. Therefore, you can not write extremely large programs. However, unless you have a lot of strings in your data, you can write a fairly large program using this .COM format.

A few benefits of a .COM file are that you can relocate the stack to a different place. The stack never contains predefined values (with one exception described later). This means that if you know you are going to use a larger stack that is left available by your program, you can create a small amount of code to allocate a different block, and point SS:SP to the end of this block. Now your program can occupy all of the 64k of the memory allocated. *A note* For that matter, you can always have your data saved in a file and load it into a different segment and

point DS to this segment. However, if you were going to do this, you might as well have just created a valid .EXE file.

Another benefit of a .COM file is that it is loaded much quicker. DOS must only load 64k or less, doesn't have to calculate relocation items, and doesn't have to calculate other program operations at start up. Also, most .EXE files must have a 256 byte header while a .COM has no such header. Therefore .COM file can be much smaller.

Let us start by thinking of a program we would like to create. This task must be simple enough for a beginner just learning to program assembly, but large enough to make a good tutorial covering a lot of areas.

What would you say to a small file viewer? Just a simple viewer that would display a file allowing you to scroll up and down. Maybe we could add small enhancements to it that would send it to the printer or something like that.

Well, first we must realize what we will need to do. To create a small file viewer like this, we will need to do the following:

- get command line
- memory allocation
- open/read/close a file
- screen output
- keyboard input

<-----

```
; Shall we start? First we must tell the assembler how and what we are
; going to do. We want a .COM file, we will be using x186 instructions,
; and we want to start at offset 100h (256d) to allow room for the PSP.
;
; Open a new file and name it V.ASM. Now enter the following at the start
; of the file: (Note that this nbasm tut.doc file is assemble-able as is)
```

```
.model tiny
.186
.code
```

```
org 100h
```

```
; We have told the assembler to create a .COM file with the 'model tiny'
; directive. A model is how the program will use segments. The tiny
; model uses one segment for all items.
;
```

```
; Then we tell the assembler to allow all instructions up to and includ-
; ing the 186. Next is the org 100h. This tells the assembler to that
; the following data/code will start at offset 100h in the segment. Then
; the .code directive telling the assembler this is the start of the
; code area, start assembling.
;
```

```
; To start our viewer we need to know what file to open. When DOS creates
; the PSP, it puts a count of the characters in the parameter list used
; on the command line at offset 80h. Then it has 127 characters of the
; parameter list at offset 81h. To get this parameter list, we need to
; know the length, so let's get the count at offset 80h. Then let's get
; the filename from this command line and place it in our filename area.
```

```
mov si,0080h ; the count byte in the PSP
```

```

        lodsb                ; 'mov al,[si]/inc si'
        or    al,al          ; is al = 0?
        jnz   short FIsThere ; if 0 then no command line there
        mov  dx,offset ParmErrS ; so use DOS print string service
        mov  ah,09           ; to print an error
        int  21h             ;
        .exit                ; exit to dos (user directive/macro)
FIsThere: inc  si            ; else if count > 0 then a command line
        mov  di,offset FileName ; skip first space and point di to
filename
GetFileN: lodsb             ; get the command line until a 0Dh
        cmp  al,13           ;
        je   short GotFileN ;
        stosb                ;
        jmp  short GetFileN  ;
GotFileN: xor  al,al         ; make string asciiz
        stosb                ;

```

```

; Now we have an asciiz filename ready for opening. An asciiz string is a
; string with a NULL char as the last character in the string. This is
; used by most services of DOS to denote eol (End Of Line). Please note
; that DOS service 09h that we used above, uses a dollar sign to denote
; eol. (We will add the data to our code later in this tutorial).
;

```

```

; Now that we have a valid filename asciiz string, we can open the file
; for viewing. However, we need to pick the correct service to open it.
; DOS has quite a few FILE OPEN services available to us. Let us pick
; service 3Dh, 'Open Existing File'. This service returns an error if
; the file does not exist.

```

```

        mov  ax,3D00h        ; open existing file for read only
        mov  dx,offset Filename ;
        int  21h             ;
        jnc  short File1OK   ; if carry, then error occurred
        mov  dx,offset File1ErrS ; print error string
        mov  ah,09h          ;
        int  21h             ;
        .exit                ;

```

File1OK:

```

; Now that we have opened the file and have a file handle in AX, allow us
; to read in the file. Being that this is a simple program, and we will
; assume all files are 32k or less, let us read only 32k of the file.

```

```

        mov  bx,ax           ; put handle in bx
        mov  ah,3Fh          ; read from handle
        mov  cx,32767        ; read at most 32k
        mov  dx,offset Buffer1 ; pointer to our buffer
        int  21h             ;
        mov  FileLen,ax      ; ax = amount read
        mov  ah,3Eh          ; close the file
        int  21h             ;

```

```

; Notice that service 3Fh returns amount read in AX. We will need this
; amount for later.
;

```

```

; Now that we have the contents of the file in our buffer, we need to set

```

```
; up our display. First let us clear out the part of the data buffer
; that is not being used by the file.
```

```
    mov di,offset Buffer1      ; clear the data buffer
    add di,FileLen            ; not being occupied by
    mov cx,32768              ; the file
    sub cx,FileLen            ;
    shr cx,1                  ; divide by two for words
    mov ax,000Dh              ;
    rep                       ;
    stosw                     ;
    jnc short skip_byte       ; if there was a byte left over
    stosb                      ; from shr cx,1 then clear it too
```

```
skip_byte:
```

```
; Now let us turn the cursor off and put a header line across the the top
; of the screen and a status line across the bottom of the screen. For
; speed, we will write directly to the screen memory rather than the
; slower methods.
```

```
    mov ah,01h                ; turn off cursor
    mov ch,20h                 ; bit number 5
    int 10h                    ;
```

```
    mov ax,0B800h              ; point es to screen memory
    mov es,ax                  ;
    xor di,di                  ; first position on the screen
    mov ah,HdrClr              ; our background color
    mov si,offset Header       ; point to our header line
    mov cx,80                  ; 80 chars across the screen
PrntHdr:  lodsb                 ;
    stosw                      ;
    loop PrntHdr               ;
    mov al,20h                 ; clear the middle section
    mov ah,txtclr              ;
    mov cx,1840                ;
    rep                        ;
    stosw                      ;
    mov ah,HdrClr              ; our background color
    mov si,offset Status       ; point to our status line
    mov cx,80                  ; 80 chars across the screen
PrntHdr1: lodsb                 ;
    stosw                      ;
    loop PrntHdr1             ;
```

```
; Now we must set up our user input area and print the actual data to the
; screen. We first calculate where we are in the data file and then
; we print 23 lines of text.
```

```
PrintIt:  mov si,offset Buffer1
    push si
    mov di,160
    mov cx,23
PrtItL1:  push cx
    push di
    push di
    mov cx,80
```

```

        mov ax,0720h
        rep
        stosw
        pop di
        mov cx,81
PrtItL2: lodsb
        cmp al,13
        je short PrtItL2D
        cmp al,32
        jb short PrtItL21
PrtItL3: mov ah,07h
        stosw
PrtItL21: loop PrtItL2
Skip2end: lodsb
        cmp al,13
        jne short Skip2end
PrtItL2D: pop di
        add di,160
        pop cx
        loop PrtItL1
        pop si

```

; Now let's pause for key input by the user and then parse it accordingly.

```

BadKey:  xor ah,ah                ;
        int 16h                ;
        cmp ax,011Bh           ; is it the ESC key?
        je short Done          ;
        cmp ax,5100h           ; page down
        jne short NoPD        ;
        xor bx,bx              ;
        mov cx,22              ; 22 lines at a time
PDownL:  cmp byte [bx+si],00h    ; end of data?
        je short PIsBot        ;
        inc bx                  ;
        cmp byte [bx+si],0Ah    ; if 0Ah skip
        jne short PDownL      ;
        loop PDownL           ;
PIsBot:  add si,bx              ; move to that position
        jmp short PrintIt      ;
NoPD:    cmp ax,4900h          ; page up
        jne short NoPU         ;
        mov bx,si              ;
        mov cx,22              ; 22 lines at a time
PUpLoop: cmp bx,offset Buffer1   ;
        je short PIsTop        ;
        dec bx                  ;
        cmp byte [bx],0Ah      ; skip all 0Ah
        jne short PUpLoop     ;
        loop PUpLoop          ;
PIsTop:  mov si,bx              ; move to that position
        jmp short PrintIt      ;
NoPU:    cmp ax,4F00h          ; end key
        jne short NotEnd       ;
        mov si,offset Buffer1   ;

```

```

        add    si,FileLen            ; move to the end of the file
        jmp    short PrintIt        ;
NotEnd:  cmp    ax,4700h            ; home key
        jne    short NotHome       ;
        mov    si,offset Buffer1    ; move to the first of the file
        jmp    PrintIt            ;
NotHome: cmp    ax,5000h            ; down arrow
        jne    short NotDown      ;
        xor    bx,bx              ;
DownL:  cmp    byte [bx+si],00h    ;
        je     short IsBot        ;
        inc    bx                 ;
        cmp    byte [bx+si],0Ah    ;
        jne    short DownL        ;
IsBot:  add    si,bx              ;
        jmp    PrintIt            ;
NotDown: cmp    ax,4800h            ; up arrow
        jne    short BadKey       ; (must have been a key not needed
        mov    bx,si              ; so just continue.)
UpLoop: cmp    bx,offset Buffer1    ;
        je     short IsTop        ;
        dec    bx                 ;
        cmp    byte [bx],0Ah      ;
        jne    short UpLoop       ;
IsTop:  mov    si,bx              ;
        jmp    PrintIt            ; loop
Done:   mov    ax,0003h            ; set screen back to text 0003h
        int    10h                ; (clear the screen)

        mov    ah,01h             ; turn on cursor
        mov    cx,0607h           ; start = 6   finish = 7
        int    10h                ;

        .exit                      ; exit to dos (user directive/macro)

```

```

; Now we need the actual data and data space used by the program we have
; created.

```

```

HdrClr   db  17h
TxtClr   db  07h
FileLen  dw  00h

Header   db  ' A demo Viewer for the Tutorial included with NBASM   Forever
Young Software
Status   db  ' Press ,  or <esc>                               Line:
'

ParmErrS db  13,10,'Error with command line$'
File1ErrS db 13,10,'Error with Source File.$'
FileName dup 128,?
Buffer1  dup 32767,?

```

```

.end
; ***** end of file

```

```

Assemble with the following:
NBASM V.ASM

```

Run with the following:

V V.ASM

(Remember to add the valid file name as a parameter: V.ASM)

NBasm will return any errors if found.

A few things to think about:

- Notice that I left the LINE: part on the status line we created at the bottom of the screen. You could easily add some code to this program to keep track of the line number and print it in this location.
- Also note that if you load a file larger than 32k, this program will still work OK, but you will get garbage as the last part of the file when it is displayed.
- I'm sure that there are a few more items that I have forgotten or bugs that I didn't find. Please let me know if you see anything that should be added or modified.