

תמיכה של DOS לתוכנית האפליקציה

כל תוכנית שאנחנו מקמפלים (באסמבלר או בשפה אחרת) היא תוכנית אפליקציה שרצה תחת מערכת הפעלה (DOS במקרה שלנו). חלק מתפקידיה של מערכת ההפעלה הוא להעביר אינפורמציה לתוכנית האפליקציה, אינפורמציה שהיא לא יכולה לדעת לבד. חלק מהאינפורמציה הזו מועברת לתוכנית על פי בקשותיה דרך קריאה ל-INT 21h, אבל רובה מועברת דרך שטחים מאותחלים של התוכנית (לפני שהיא מתחילה לרוץ). אנחנו נראה חלק מהפרטים של אחד השטחים הללו - ה-Program Segment Prefix או ה-PSP. ה-PSP מכיל, בין השאר, אינפורמציה על הזכרון שעומד לרשות התוכנית, אינפורמציה הנחוצה להחזיר שליטה ל-DOS בדרך הישנה (לפני שישימו את INT 21h, אופציה 4Ch), וכן הפרמטר לתוכנית הראשית.

כאשר תוכנית האפליקציה מתחילה לרוץ, DOS דואג לכך שאוגר ה-DS מכיל את מספר הסגמנט של ה-PSP. זאת הסיבה שה-DS לא מצביע על סגמנט ה-DATA שלה, עד שלא נבצע את הפקודות

```
MOV AX,@DATA
MOV DS,AX
```

או

```
MOV AX,שם הסגמנט
MOV DS,AX
```

ה-Program Segment Prefix

שדות ה-PSP

<u>כתובת PSP+</u>	<u>גודל</u>	<u>הגדרת השדה</u>
0	2	פקודת INT 20h
2	2	סה"כ זכרון ביחידות של 16B
4	1	שמור
5	5	פקודת CALL FAR ל-DOS function dispatcher
0Ah (10)	4	כתובת של פסיקה 22h (סיום התוכנית)

0Eh (14)	4	כתובת של פסיקה 23h (Ctrl-Break Handler)
12h (18)	4	כתובת של פסיקה 24h (Critical Error Handler)
16h (22)	2	מספר הסגמנט של ה-PSP של תהליך האב
18h (24)	20	Handle Table
2Ch (44)	2	מספר הסגמנט של ה-Environment Segment
2Eh (46)	4	שמור
32h (50)	2	Handle Table Size
34h (52)	4	Handle Table Address
38h (56)	24	שמור
50h (80)	2	Int 21h dos call
52h (82)	1	FAR RET
53h (83)	9	שמור
5Ch (92)	16	Default unopened file control bloc #1
6Ch (108)	20	Default unopened file control bloc #2
80h (128)	1	אורך פרמטר
81h (129)	?	פרמטר גם ברירת מחדל ל-DTA

INT 20h היה השיטה הישנה של החזרת השליטה ל-DOS, היחידה שהיתה ב-DOS 1.0 למשל. רק בשלב מאוחר יותר הוסיפו ל-INT 21h את האופציה 4Ch. הצורה הזו היא פחות גמישה מהצורה החדשה - את הפקודה INT 20h חייבים לבצע בדיוק במיקום של PSP+0 - כלומר צריך להסתעף לשם. אני משער שהסיבה לכך היתה שה-DOS הישן היה צריך את האינפורמציה ב-PSP ושה-DOS לא שמר פוינטר ל-PSP אצלו. הדרך הנפוצה לממש את ההסתעפות הזו היא כלהלן:

התוכנית הראשית היתה מוגדרת כ-PROC FAR, ובאופן כללי נראתה כך:

```
Main PROC FAR
    PUSH DS
    MOV AX,0
    PUSH AX
```

וכאשר התוכנית מעונינת לחזור ל-DOS, היא מבצעת את הפקודה RET, שהוא מסוג RET FAR.

הפרמטר לתוכנית הראשית

הפרמטר לתוכנית הראשית הוא מה שממומש ב-C בצורה

```
main(int argc, char *argv[])
```

אלא שבניגוד ל-C, באסמבלר הפרמטר מתקבל ללא שום עיבוד. ב-PSP+80 יש byte בודד עם האורך בתווים של הפרמטר לתוכנית הראשית (אורך ממילא מוגבל ל-128), ומיד אחרי הפרמטר עצמו, כפי שהמשתמש הקיש אותה. היחס לתו הרווח הוא כאל כל תו אחר. הפרמטר מועבר כמחרוזת אחת, כלי קשר מה יש בה, ולא כולל ה-newline שבסופו. הפרמטר כולל את הרווח שמיד אחרי שם התוכנית, כלומר PSP+81 הוא תמיד רווח, אלא אם כן המשתמש לחץ את המקש Enter מיד אחרי שם קובץ ה-.EXE.

לדוגמא, אם ההרצה של התוכנית היא

```
C:\ASM>myprog.exe ABC def G
```

והמקש Enter בא מיד אחרי ה-G, אזי ב-PSP+80 היה הערך 10, וב-PSP+81 יהיה המחרוזת "ABC def G" (ללא הגרשיים, כמובן). אגב, אם היינו משמיטים את ה-.exe" זה היה בדיוק אותו דבר כלומר אם ההרצה היתה

```
C:\ASM>myprog ABC def G
```

זה היה עדיין נכון שב-PSP+80 היה הערך 10, וב-PSP+81 יהיה המחרוזת "ABC def G".

```

/* demo2.c - demonstrate command line arguments */

#include <stdio.h>

void main(int argc, char *argv[])
{
    int i;
    printf("\nI am the powerful %s command!\n", argv[0]);
    if (argc == 1)
        printf("I was called without parameters.\n");
    else
    {
        printf("I was called with %d parameters.\n", argc);
        for(i=0; i < argc; i++)
            printf("My argv[%d] parameter is: %s\n",i,argv[i]);
    }
} /* main */

```

E:\>demo2 11 222 "33 333" 44444

```

I am the powerful E:\DEMO2.EXE command!
I was called with 5 parameters.
My argv[0] parameter is: E:\DEMO2.EXE
My argv[1] parameter is: 11
My argv[2] parameter is: 222
My argv[3] parameter is: 33 333
My argv[4] parameter is: 44444

```

E:\>demo2

```

I am the powerful E:\DEMO2.EXE command!
I was called without parameters.

```

E:\>

272

תוכניות דוגמא repparm2.asm ו-repparm4.asm

התוכניות הללו נועדו להמחיש אספקטים שונים של עבודה עם ה-PSP. מה שהן עושות בפועל הוא להדפיס פעמיים את הפרמטר לתוכנית הראשית פעם שניה ההדפסה כוללת את בית הגודל של הפרמטר. הדרך שהן עושות את זה היא להעתיק את הפרמטר מה-PSP לזכרון הרגיל ולהדפיס אותו משם. זה כמובן לא מחייב המציאות, ניתן היה להדפיס את הפרמטר ישירות מהמיקום שלו ב-PSP. המהלך נעשה ע"י העתקה על מנת להדגים את השימוש בהנחית ה-ASSUME. בתוכניות הללו הנחיות ה-ASSUME נפרשות רק בדיוק באותו מקום שהן מתחילות לשקף באמת את המציאות. שימו לב שבתוכנית repparm2.asm נפרשת ההנחיות

```
ASSUME NOTHING
ASSUME CS:_TEXT
```

ככדי לשקף את המציאות בהתחלה ולאחר הצבת הערך @DATA ל-ES נפרשת ההנחיה

```
ASSUME ES:_DATA
```

מרגע זה ואילך ניתן להתייחס למשתנים בסגמנט המידע דרך ה-ES. לאחר ההעתקה מוצב הערך @DATA לתוך DS ומיד לאחר מכן נפרשת ההנחיה

```
ASSUME DS:_DATA
```

העתקת הפרמטר לתוכנית הראשית נעשית ע"י הפקודות

```
CLD
REP MOVSB
```

שמשמעותו "העתק CX בתים מכתובת DS:[SI] לכתובת ES:[DI]". הפקודה MOVSB היא אחת מפקודות המחרוזת שמטרתם לממש ביעילות פעולות מסוימות על מערכים בגדלים שונים של מקדמים. במקרה הזה זהו מימוש יותר יעיל מהמימוש הכמעט שקול

Loop1:

MOV AL,DS:[SI]

MOV ES:[DI],AL

INC SI

INC DI

LOOP Loop1

מלבד יעילות ההבדל העיקרי שבמימוש של MOVSB האוגר AL לא ישנה את ערכו בעקבות הפקודה. ערכם של האוגרים SI, DI ו-CX בתום הרצת 2 המימושים יהיו זהים.

התוכנית repparm4.asm נבדלת מ-repparm2.asm בעיקר בכך שהיא נכתבה באסמבלי סטנדרטי ולכן לא היה כאן הכרח להשתמש בשמות הסגמנטים המקובלים _DATA, _TEXT וכו'. לפיכך הנחיות ה-ASSUME הן

ASSUME CS:Cseg

ASSUME ES:Dseg

ASSUME ES:Dseg

בתוכנית repparm4.asm לא מופיע הנחיה

ASSUME NOTHING

כי אין צורך בכך, זהו ממילא ברירת המחדל.

```

;
; repparm2.asm - Echo DOS command line parameters.
;

        .MODEL SMALL
        .STACK 100h
        .DATA
Parm      DB  256 DUP (?)          ; Move command line parameters here
; Message Table:
Msg1      DB  'Value of PSP from offset 80h: ',13,10
Msg2      DB  13,10,'Value of PSP from offset 81h: ',13,10
; Error message:
ErrMsg    DB  'ERROR: Empty parameter. '
ErrEndMarker DB 0
;
Dos_Write MACRO BuffName, BuffSize
; Macro to call INT 21h with AH = 40h, BX = 1
; Write to stdout BuffSize chars from buffer BuffName
; Input expected:
; BuffName = Buffer name
; BuffSize = Number of chars to print.
; Registers Destroyed:
; AX, BX, CX, DX
;
        MOV DX,OFFSET BuffName      ; Set INT 21h from parameters
IFDIF <BuffSize>,<CX>                ; If BuffSize is Not CX, move ...
        MOV CX,BuffSize             ; ... BuffSize to CX
ENDIF
        MOV AH,40h                  ; DOS write from handle function #
        MOV BX,1                     ; Standard output handle
        INT 21h                      ; Print the string
        ENDM
        .CODE

Start:
;
; Standard program prologue except retain DS as ptr to PSP
;
ASSUME NOTHING
ASSUME CS:_TEXT
        MOV AX,@DATA                ; Establish extra segment addressability ...
        MOV ES,AX                   ; ... to our data segment
ASSUME ES:_DATA
;
; Fetch command line parameter from PSP
;
        MOV SI,80h                  ; Source string offset (within PSP)
        MOV DI,OFFSET Parm          ; Destination string offset
        MOV CL,DS:[80h]              ; String length to move
        MOV CH,0
        INC CX                       ; Include size byte in length consideration.
        CLD                          ; Write forward
        REP MOVSB                    ; Move parm into our data area
;
; Establish normal data segment addressability
;
        MOV AX,@DATA
        MOV DS,AX
ASSUME DS:_DATA

```

275

```

;
; Check for valid parameter and convert it to numeric value
;
CMP Parm,0                ; Is parameter length = 0 ?
JZ Error                  ; Branch if so
;
Dos_Write Msg1,Msg2-Msg1   ; Display Msg1
MOV CL,Parm                ; Move Msg length to CX
MOV CH,0                   ;
INC CX                     ; Include size byte in length consideration.
Dos_Write Parm,CX          ; Display the Command-line params ...
                           ; ... including size byte
;
Dos_Write Msg2,Errmsg-Msg2 ; Display Msg2
MOV CL,Parm                ; Move Msg length to CX
MOV CH,0                   ;
Dos_Write Parm+1,CX        ; ; Display the Command-line params ...
                           ; ... NOT including size byte
;
JMP SHORT Done
;
; Display error message for invalid parameter:
;
Error:
    Dos_Write ErrMsg,ErrEndMarker-ErrMsg    ; Display Error Message
Done:
    MOV AH,4Ch
    INT 21h                                ; Return to DOS
;
END Start

```

```

E:\>repparm2 123456789 123456789 123456789 123456789 123456789 123456789 1234
Value of PSP from offset 80h:
A 123456789 123456789 123456789 123456789 123456789 123456789 1234
Value of PSP from offset 81h:
123456789 123456789 123456789 123456789 123456789 123456789 1234

E:\>repparm2
Value of PSP from offset 80h:
A
Value of PSP from offset 81h:

E:\>repparm2 123456789
Value of PSP from offset 80h:
123456789
Value of PSP from offset 81h:
123456789

E:\>repparm2
ERROR: Empty parameter.

E:\>

```

2.76


```

; repparm4.asm - Echo DOS command line parameters,
; standard segment directives.
;
Sseg SEGMENT PARA STACK 'STACK'
    DB 100h DUP (?)
Sseg ENDS

;
Dseg SEGMENT WORD 'Dseg'
Parm DB 256 DUP (?) ; Move command line parameters here
; Message Table:
Msg1 DB 'Value of PSP from offset 80h: ',13,10
Msg2 DB 13,10,'Value of PSP from offset 81h: ',13,10
; Error message:
ErrMsg DB 'ERROR: Empty parameter. '
ErrEndMarker DB 0
;
Dseg ENDS
;
Dos_Write MACRO BuffName, BuffSize
; Macro to call INT 21h with AH = 40h, BX = 1
; Write to stdout BuffSize chars from buffer BuffName
; Input expected:
; BuffName = Buffer name
; BuffSize = Number of chars to print.
; Registers Destroyed:
; AX, BX, CX, DX
;
MOV DX,OFFSET BuffName ; Set INT 21h from parameters
IFDIF <BuffSize>,<CX> ; If BuffSize is Not CX, move ...
MOV CX,BuffSize ; ... BuffSize to CX
ENDIF
MOV AH,40h ; DOS write from handle function #
MOV BX,1 ; Standard output handle
INT 21h ; Print the string
ENDM
;
Cseg SEGMENT WORD 'Cseg'
Start PROC FAR
;
; Standard program prologue except retain DS as ptr to PSP
;
ASSUME CS:Cseg
PUSH DS ; Set up DOS return instruction
XOR AX,AX ; AX = 0
PUSH AX
;
MOV AX,Dseg ; Establish extra segment addressability ...
MOV ES,AX ; ... to our data segment
ASSUME ES:Dseg

```

```

;
; Fetch command line parameter from PSP
;
MOV SI,80h          ; Source string offset (within PSP)
MOV DI,OFFSET Parm  ; Destination string offset
MOV CL,DS:[80h]     ; String length to move
MOV CH,0
INC CX              ; Include size byte in length consideration.
CLD                 ; Write forward
REP MOVSB           ; Move parm into our data area
;
; Establish normal data segment addressability
;
MOV AX,Dseg
MOV DS,AX
ASSUME DS:Dseg
;
; Check for valid parameter and convert it to numeric value
;
CMP Parm,0          ; Is parameter length = 0 ?
JZ Error            ; Branch if so
;
Dos_Write Msg1,Msg2-Msg1 ; Display Msg1
MOV CL,Parm         ; Move Msg length to CX
MOV CH,0            ;
INC CX              ; Include size byte in length consideration.
Dos_Write Parm,CX    ; Display the Command-line params ...
; ... including size byte
;
Dos_Write Msg2,Errmsg-Msg2 ; Display Msg2
MOV CL,Parm         ; Move Msg length to CX
MOV CH,0            ;
Dos_Write Parm+1,CX  ; ; Display the Command-line params ...
; ... NOT including size byte
;
JMP SHORT Done
;
; Display error message for invalid parameter:
;
Error:
    Dos_Write ErrMsg,ErrEndMarker-ErrMsg ; Display Error Message
Done:
    RET              ; Far return - Return to DOS
;
Start ENDP
Cseg ENDS
    END Start      ; Set first instruction

```

```

E:\>repparm4 123456789 123456789 123456789 123456789 123456789 123456789 12345
Value of PSP from offset 80h:
B 123456789 123456789 123456789 123456789 123456789 123456789 12345
Value of PSP from offset 81h:
123456789 123456789 123456789 123456789 123456789 123456789 12345
E:\>

```