

תוכנית זו ממחישה את המשמעות הקלאסית של שימוש במקרו: יצירת קודים נפרדים בעלי משמעות שונה על ידי אותו מקרו.

המקרו Int21h_ah9 יוצר קוד המדפיס מחרוזת המסתיימת ב-'\$' ע"י INT 21h אופציה AH = 9. שם המחרוזת אמור להיות מועבר כפרמטר למקרו. המשמעות של שתי הקריאות

```
Int21h_ah9    First_Msg  
Int21h_ah9    Second_Msg
```

תהיה הפרישה של הפקודות

```
MOV DX, OFFSET First_Msg  
MOV AH,9  
INT 21h  
MOV DX, OFFSET Second_Msg  
MOV AH,9  
INT 21h
```

וזה מסביר את שתי ההדפסות.

יש לשים לב לכך ששתי ההדפסות הן ביצוע יחיד של שני קודים שונים ולא שני ביצועים של אותו קוד, דבר שהיה מתרחש אילו היה מדובר בפרוצדורה. כמו כן אין כאן פעולות הסתעפות לקוד, וגם אין פקודות המישמות פרמטרים. המחיר הוא שהקוד מופיע בתוכנית פעמיים, דבר שבפוטנציה יכול להאריך את התוכנית.

```

;
; dymac1.asm - illustrate print macro with params.
;

.MODEL SMALL
Int21h_ah9 MACRO Msg
    MOV DX,OFFSET Msg
    MOV AH,9      ; INT 21h print-string-until-$ option
    INT 21h      ; print the string
ENDM

.STACK 100h
.DATA
First_Msg      DB 'First message by DOS 21h AH=9.',13,10,'$'
Second_Msg     DB 'Second message by DOS 21h AH=9.',13,10,'$'
                ;

;
.CODE
Main:
    MOV AX,@DATA      ; Standard Program prefix
    MOV DS,AX         ;

    ;
    Int21h_ah9 First_Msg      ; Print First_Msg
    Int21h_ah9 Second_Msg    ; Print Second_Msg
    ;

Done:
    MOV AH,4Ch         ; Set terminate option for INT 21h
    INT 21h           ; Return to DOS (terminate program)
END Main

```

```

E:\>dymac1
First message by DOS 21h AH=9.
Second message by DOS 21h AH=9.

```

```

E:\>

```

התוכנית הזו נועדה להמחיש שימוש אפשרי של המקרו REPT. שימוש
אופייני של REPT הוא להגדיר באסמבלי מבנה נתונים מערכי מאתחל מבלי
לבצע חישובים בזמן ריצה. תוצאת השימוש ב-REPT כאן תהיה:

```
A_Str DB 'A'  
      DB 'B'  
      DB 'C'  
      .....  
      DB 'Z'
```

נוצר כאן מערך תוים של האותיות 'A' עד 'Z' מבלי לבצע את החישוב
בזמן ריצה ומבלי שהתוכניתן יצטרך לעשות זאת ידנית.

```

;
; macstri1.asm - Genrate 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
;               using REPT macro.
;
.MODEL SMALL
Int21h_ah9 MACRO Msg
    MOV DX,OFFSET Msg
    MOV AH,9      ; INT 21h print-string-until-$ option
    INT 21h       ; print the string
ENDM
;
.STACK 100h
.DATA
A_Str DB 'A'
Char_Val = 'B'
REPT 25
    DB Char_Val
    Char_Val = Char_Val+1
ENDM
    DB '$'
;
.CODE
Main:
    MOV AX,@DATA      ; Standard Program prefix
    MOV DS,AX         ;
;
    Int21h_ah9 A_Str   ; Print A_Str
;
Done:
    MOV AH,4Ch         ; Set terminate option for INT 21h
    INT 21h            ; Return to DOS (terminate program)
END Main

```

```

E:\>macstri1
ABCDEFGHIJKLMNOPQRSTUVWXYZ

```

```

E:\>

```

התוכנית הזו ממחישה את האפשרויות בשימוש של פרמטרים במקרו. תפקידו של המקרו Mov_Ax הוא להציב ערך לתוך האוגר AX. לצורך הדוגמא, נניח שעבור המקרה של הצבת אפס לתוך AX המימוש של

```
XOR AX,AX
```

יותר יעיל מאשר המימוש

```
MOV AX,0
```

לפיכך המקרו הזה רוצה לבדוק אם הערך שהוא אמור להציב לתוך AX. אם מדובר באפס הוא ימומש ע"י XOR ועבור כל ערך C אחר הוא יממש ע"י

```
MOV AX,C
```

כך שהקריאות

```
Mov_Ax 0
```

```
Bval = 0  
Mov_Ax Bval
```

ימומש ע"י

```
XOR AX,AX
```

בעוד שהקריאה

```
Aval = 8  
Mov_Ax Aval
```

ימומש ע"י

```
MOV AX,8
```

שים לב שבפרמטר מקרו מן הסוג הזה הפרמטר יכול להיות רק ערך הידוע בזמן ריצה. כלומר או קבוע או משתנה של הפרה-קומפילר שמכיל קבוע, כמו שיש כאן.

```

;
; macmoval.asm - Illustrate pre-compiler variables,
;                  macro with params.
;

.MODEL SMALL
Mov_Ax MACRO IntVal
    IF IntVal EQ 0          ; Is IntVal = 0
        XOR AX,AX          ; Yes, use XOR
    ELSE
        MOV AX,IntVal      ; No, use MOV
    ENDIF
    ENDM                    ; Of Mov_Ax
;
.STACK 100h
.DATA
.CODE
Main:
    MOV AX,@DATA           ; Standard Program prefix
    MOV DS,AX              ;
;
Aval = 8
    Mov_Ax Aval
    Mov_Ax 0
Bval = 7
    Mov_Ax Bval
Bval = 0
    Mov_Ax Bval
Done:
    MOV AH,4Ch             ; Set terminate option for INT 21h
    INT 21h                ; Return to DOS (terminate program)
    END Main

```

התוכנית הזו ממחישה שימוש במקרו לפרישת קוד המחשב ערך כמעין תחליף לפרצדורה, בכפוף להשגות שכבר נסקרו קודם. במקרה הזה נפרש קוד המחשב עצרת של פרמטר שהוא משתנה שערכו ידוע רק בזמן ריצה. אילו היה כאן מדובר בחישוב עצרת של קבוע הידוע בזמן אסמבלי, דבר שהוא אפשרי, המקרו היה נראה לגמרי אחרת. תוצאת הפרישה של קריאת המקרו

Fact Source

בתוכנית הזאת תהיה:

```
MOV AX,1
MOV CX,Source
??0000:
MUL CX
LOOP ??0000
```

בזמן ריצה התוצאה תהיה ב-AX.

הסמל "??0000" מחליף את הסמל Next בשל ההגדרה של Next כ-LOCAL. אילו היתה הקריאה נוספת ל-Fact או כל מקרו אחר שיש לו סמל המוגדר כ-LOCAL היה נוצר הסמל "??0001" וכן הלאה, עד 4 ספרות הקסדצמיליות כלומר עד 65336 סמלים מסוג זה לכל היותר.

; fact1.asm - Local label in a macro.

```
;
        .MODEL SMALL
        .STACK 100h
        .DATA
Source    DW    ?
Result    DW    0
Fact MACRO N
    LOCAL Next
    MOV AX,1
    MOV CX,N
Next:
    MUL CX
    LOOP Next
ENDM
        .CODE
; Standard prefix
    MOV AX,@DATA
    MOV DS,AX
    MOV     Source,8
    Fact    Source
    MOV     Result,AX
    MOV AH,4Ch
    INT 21h
    END
```


תוכנית הדוגמא macregs3.asm

התוכנית הזו נועדה להדגים שימוש ב-IRP ו-IFIDN בתוך מקרו. המקרו המקרו Stack_GP_Regs הוא מעין מקרו שנועד להבטיח ששימור/שיחזור אוגרים יהיה עקבי. הקריאה

Stack_GP_Regs PUSH

תפרוש את הקוד

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ואילו הקריאה

Stack_GP_Regs POP

תפרוש את הקוד

POP DX

POP CX

POP BX

POP AX

הרעיון המתקדם ביותר כאן הוא שניתן ע"י IFIDN לבצע קוד מותנה ע"י השוואה של סמלים (זהה או לא זהה) על ידי IFIDN או IFDIF. הפקודה

IFIDN <Reg_op>,<PUSH>

מתנהל עקרונית כך: מאחר ו-Reg_op זה שם של משתנה (פרמטר) של המקרו, האסמבלר לוקח את תוכנו להשוואה. מאחר ואין משתנה דומה בשם PUSH, להשוואה נלקח הטקסט "PUSH" כמות שהוא. אם הסמלים זהים, נפרש הקוד הזה, ואם לא נפרש המקרה של ה-ELSE.

```

; macregs3.asm - demonstrate macro with IFIDN
;
.MODEL SMALL
Stack_GP_Regs MACRO Reg_Op
    IFIDN <Reg_Op>,<PUSH>
        IRP Reg,<AX,BX,CX,DX> ; Regular order
        PUSH Reg
        ENDM ; Of IRP
    ELSE
        IFIDN <Reg_Op>,<POP>
            IRP Reg,<DX,CX,BX,AX> ; Reverse order
            POP Reg
            ENDM ; Of IRP
        ENDIF
    ENDIF
ENDIF
ENDM ; Of Push_All_Regs
;
.STACK 100h
.DATA
.CODE
Main:
    MOV AX,@DATA ; Standard Program prefix
    MOV DS,AX ;
    ;
    Stack_GP_Regs PUSH
    Stack_GP_Regs POP
    ;
Done:
    MOV AH,4Ch ; Set terminate option for INT 21h
    INT 21h ; Return to DOS (terminate program)
    END Main

```

התוכנית הזו ממחישה שימוש באופרטור ההדבקה & וכן שימוש באופרטור
IRPC והאפשרות שמקרו יקרא למקרו.
הפרישה של הקריאה למקרו

Push_Reg D

הינה

PUSH DX

ואילו הפרישה של הקריאה

Push_GP_Regs

יהיה

PUSH AX

PUSH BX

PUSH CX

PUSH DX

כלומר שהביטוי

&Rletter&X

האסמבלר בשני המקרים מפרש את מה שהוא צריך להדביק בתור התוכן של
המשתנה - סמל. זה לא כל כך ברור שכן נסיון למימוש Push_GP_Regs ע"י
המקרו הפנימי

IRPC Reg_Char,ABCD

PUSH &Reg_Char&X

ENDM

הביא לפרישה של הקוד

PUSH Reg_CharX

PUSH Reg_CharX

PUSH Reg_CharX

PUSH Reg_CharX

```

; macregs4.asm - demonstrate macro with paste (&) and IRP
;
.MODEL SMALL
Push_Reg MACRO RLetter
    PUSH    &RLetter&X          ; Paste operand together with X
ENDM
Push_GP_Regs MACRO
    IRPC Reg_Char,ABCD          ; ABCD is macro char string
    Push_Reg Reg_Char
ENDM ; Of IRP
ENDM ; Of Push_GP_Regs

;
.STACK 100h
.DATA
.CODE
Main:
    MOV    AX,@DATA              ; Standard Program prefix
    MOV    DS,AX                ;
    ;
    Push_Reg    D
    Push_GP_Regs
    ;
Done:
    MOV    AH,4Ch                ; Set terminate option for INT 21h
    INT    21h                  ; Return to DOS (terminate program)
    END    Main

```

התוכנית הזו ממחישה במידה רבה מה, פוטנציאלית, ניתן לעשות בעזרת הכלי מקרו, אבל גם במידה רבה למה כמעט ולא משתמשים בזה. מקרו מהסוג של Fast_Mult מאפשר למתכנת לפרוש בתוכניות קוד שונה לחלוטין בהתאם לתנאים מורכבים למדי, הגם שמדובר בתנאים שחייבים להיות ניתנים לבדיקה בזמן אסמבלי. יחד עם זה תכנות מקרו כזה היא משימה מאד מורכבת, הקוד של המקרו ארוך ומסובך, וקשה מאד לודא את נכונותו. מקרו כזה יכול בקלות רבה להיות מקור לשגיאות תוכנה שקשה לאתר.

התפקיד של Fast_Mult לפרוש קוד המכפיל בקבוע (הידוע בזמן אסמבלי) את תוכן AL הידוע רק בזמן ריצה, ולחשב את התוצאה לתוך AX. הרעיון כאן הוא לנצל את העובדה שאם מדובר בקבוע שהוא חזקה של 2, ניתן לבצע את הכפל ע"י הזזה שמאלה של ביטים, שהוא מהיר יותר מכפל.

לדוגמא, הקריאה

Fast_Mult 16

תביא לפרישה של הקוד

```
MOV AH,0
SHL AX,4
```

כי 4 הוא הלוגריתם של 16 לפי בסיס 2.
הקריאה

Fast_Mult 25

תביא לפרישה של הקוד

```
MOV DL,25
MUL DL
```

מה שנחסך ע"י מימוש מקרו הזה הוא שהמתכנת לא צריך לשים לב או לבדוק אם הקבוע שאת הכפל שלו הוא נדרש לממש הוא חזקה של 2. ברור שבמקרה של 8 ביט זה ממש לא תורם הרבה, אבל אם היה מדובר בקבועים 32 ביט מקרו כזה יכול להיות בעל משמעות מעשית. כאן מומש המקרה של 8 ביט לצורכי פשטות, שכן המקרו מורכב מספיק גם ככה.

השיטה שבה הבדיקה נעשית היא פשוט לבדוק את כל החזקות של 2 לקבוע, ובמקרה של שיוויון לפרוש את קוד ההזה. במקרה של כשלון כל ההשוואות נפרש הכפל. מספר ההשוואות הוא כמספר הביטים של הגודל המירבי של הקבוע פחות 1 (7 במקרה שלנו) כלומר הלוגריתם לפי בסיס 2 של הגודל המירבי של הקבוע, כלומר בהחלט סביר מבחינת הסיבוכיות של הפעולה.

המקרו מתחזק שלושה משתנים: `Power_of_two` המאותחל ב-128 (החזקה הכי גבוהה הרלוונטית) משתנה `Is_power_of_two` דגל המאותחל באפס ו-`Count` המאותחל ב-7 שהוא הלוגריתם לפי בסיס 2 של `Power_of_two`. המקרו מבצע לולאה שבו הוא משווה את `Power_of_two` לקבוע `Fact`. במקרה של שיוויון הוא נותן ערך 1 למשתנה `Is_power_of_two` ויוצא מהלולאה על ידי הפקודה `EXITM`. אחרת הוא מחלק ב-2 את `Power_of_two` ומחסיר 1 מ-`Count`. אם כל ההשוואות האפשריות נכשלות הוא שוב יוצא מהלולאה.

בשלב הזה נבדק המשתנה `Is_power_of_two` ובהתאם לערכו האסמבלר יודע אם `Fact` הוא חזקה של שניים. במקרה שכן נפרש הקוד

```
MOV AL,0
SHL AX,Count
```

אחרת נפרש

```
MOV DL,Fact
MUL DL
```

השורה התחתונה היא שקוד המכיל מקרו כזה (כתוב נכון), השימוש במקרו מאריך את תהליך האסמבלי אבל מקל על המתכנת והתוצאה של האסמבלי, קובץ ה-`exe`, מהיר יותר. במושגים של בית חרושת מדובר באמצעי המאריך את היצור אך מקל על העובדים במפעל והמוצר הסופי המסופק ללקוח מהיר יותר.

```

; multmac5.asm - demonstrate pre-compiler programming
;
.MODEL SMALL
Int21h_ah9 MACRO Msg
    MOV DX,OFFSET Msg
    MOV AH,9      ; INT 21h print-string-until-$ option
    INT 21h      ; print the string
ENDM

Fast_Mult MACRO Fact
    ; Efficient calculation of AX := AL * Fact
    ; Implementation by SHL if Fact is power of two
    ; Otherwise, ordinary mult
    ; AL is assumed to contain first value
    ;
    Is_power_of_two = 0
    Count = 7      ; bits 0..7
    Power_of_two = 80h

;
    REPT 7
        IF Power_of_two EQ Fact      ; Fact is a power of two
            Is_power_of_two = 1
            EXITM      ; Exit REPT
        ENDIF
        Count = Count - 1
        Power_of_two = Power_of_two / 2
    ENDM      ; End of REPT

;
; Expand NOTHING if Fact == 1 (Is_power_of_two == 1, Count == 0)
    IF (Is_power_of_two EQ 1) AND (Count NE 0)
        MOV AH,0
        SHL AX,Count
    ELSE
        IF (Is_power_of_two EQ 0)
            MOV DL,Fact
            MUL DL
        ENDIF
    ENDIF
ENDM      ; End of Fast_Mult

.STACK 100h
.DATA
Print_Str DB      ?
           DB      ?
           DB      '$'

;
.CODE
.386

Main:
    MOV AX,@DATA      ; Standard Program prefix
    MOV DS,AX
    ;
    MOV AL,3
    Fast_Mult 16
    MOV Print_Str,AL
    MOV AL,2
    Fast_Mult 25
    MOV Print_Str+1,AL
    Int21h_ah9 Print_Str
    ;

Done:
    MOV AH,4Ch      ; Set terminate option for INT 21h
    INT 21h      ; Return to DOS (terminate program)
    END Main

```

```

E:\>multmac5
02
E:\>

```

258