תקציר מספר 8

אסמבלי מותנה ומקרו

אסמבלי מותנה ומקרו הם אמצעי תכנות שעומדים לרשותינו תוך שימוש בשלב מיוחד של הידור תוכניות הנקרא פרה-קומפילציה pre-compilation. ברוב השפות המודרניות היום, וכמעט כל האסמבלרים בעבר, ההידור של תוכנית עובר שלב מוקדם של המרה. לצורך ההמחשה נניח תחילה שמדובר בתוכנית C. ההמרה שבה מדובר היא של קובץ תוכנית הקלט ב-C, לקובץ תוכנית שגם היא בשפת C - אבל מצומצמת.

על מנת באמת להבין את מושג הפרה-קומפילציה צריך להיות מודעים לשתי עובדות:

1. הפרה קומפילציה היא תהליך המרה של קובץ טקסט לקובץ טקסט קצת אחר.
2. ההמרה הזו נעשית כשלב ראשון של תהליך הקומפילציה. החלטות שנעשות בשלב זה הם לפי מידע שידוע בשלב המוקדם הזה. אי אפשר לקבל החלטות לפי מידע שידוע רק בזמן ריצה, למשל.

אסמבלי מותנה (או קוד מותנה) הינו אפשרות מתוחכמת שמספק האסמבלר למתכנת כדי לבצע אסמבלי של קטעי קוד, רק אם מתקיימים תנאים מסוימים. אסמבלי מותנה יגרום רק לחלקים הדרושים מהתוכנית לעבור אסמבלי ולהכלל בקוד של שפת מכונה.

אמצעי התכנות העיקרי - יכולת לבדוק קיום תנאים מסוימים לגבי סמלים, למשל,

```
IF  condition
 ... to be assembled if condition is true
ENDIF
```

יתרונות

- מאפשר מימוש גירסה אחת של הקוד למטרות שונות דבר המהווה הקלה על תחזוקה עדכון ותיעוד.

- מאפשר למתכנת להגדיר בקלות מבנה (כגון טבלאות) החוזרים על עצמם.

כפי שאנחנו נראה, התנאים הנבדקים הם כמעט תמיד תנאים על סמלים. התנאים

האופיניים הם האם סמל הינו מוגדר, או יש לו ערך מסוים.

מקרו

מקרו הוא אמצעי נוסף שמספק האסמבלר לשיפור המבניות וחלוקה לקטעים קטנים ועצמאיים.

קטע קוד המוגדר כמקרו מקבל שם ומופעל באמצעות הזכרת שמו - בדומה לפרוצדורה.

במקום בו מוזכר שם המקרו הוא מוחלף בקטע קוד (במקום הסתעפות).
המקרו מתפתח בזמן אסמבלי.
אם השם מופיע יותר מפעם אחת - הקוד יופיע כמספר הפעמים.
שימוש מקרו במקום בפרוצדורה - מהיר יותר בביצוע - בזבזני יותר במקום.

המושגים בשפת C

קוד מותנה

בשפת C ניתן לממש קוד מותנה בעזרת ההנחיה #if. מקרו-ים מוגדרים על ידי ההנחיה #define.

שימוש שהיה נפוץ בעבר (לפני עידן ה-debugger-ים) של קוד מותנה היה לצורך ניפוי שגיאות. הרעיון היה לשתול בתוכנית הדפסות ביניים של משתנים על מנת לעקוב אחרי ביצוע התוכנית. על מנת שהדפסות הביניים לא יופיעו בגירסה הסופית ועל מנת לחסוך את הצורך להוציא את הדפסות הביניים (או להפוך אותם להערות) השתמשו באמצעי נוסח הבא:

```
#ifdef MODE_DEBUG
  printf("x = %f, i = %d\n" x, i);
#endif
```

במידה ורצו ליצור קובץ ביצועי שמכיל את הדפסות הביניים, קימפלו את התוכנית, (נניח ששם הקובץ myprog.c, נניח ב-TURBO C):

```
tcc -DMODE_DEBUG myprog.c
```

ה- D- גורם לכך שהסמל הצמוד לו (MODE_DEBUG במקרה הזה) יהיה מוגדר

(defined). במקרה כזה, ה-printf הנ"ל ימומש בקובץ הביצועי הנוצר. לעומת זאת, במידה ופקודת הקימפול תהיה:

```
tcc  myprog.c
```

ה-printf לא יממומש בקובץ הביצועי.

מקרו

אשר למקרו-ים, בשפת C הם ממומשים ע"י ההנחיה #define. דוגמא קלאסית היא המקרו sqr הבא, המממש חישוב ריבוע של פרמטר:

```
#define sqr(x)  ((x)*(x))
```

לדוגמא אם בתוכנית כלשהיא נכתב:

```
y = sqr(z+2.0);
```

לאחר ההמרה התוכנית תהיה:

```
y = ((z+2.0)*(z+2.0));
```

שים לב: הקוד שיווצר ע"י מרבית הקומפילרים, ה-z+2.0 יחושב פעמיים. מקרו-ים יכולים בצורה כזו לגרום לקוד לא יעיל ולבעיות. לדוגמא, תאר לעצמך שנכתוב בתוכנית את הפקודה sqr(++x). יתבצע קידום כפול של המשתנה x, בעוד שיש להניח שהכוונה היתה לקידום בודד.

במילים אחרות, למרות הדמיון, sqr איננה פונקציה. היא הנחיה לקומפילר לפרוש טקסט מסוים. כל פעם שנקרא ל-sqr יפרש טקסט נפרד. מסיבה זו אין זה משנה אם הפרמטר שלה הוא שלם או ממשי. sqr יכול לקבל פרמטר מכל סוג מספרי אפשרי, משום שקוד ה-C של הכפלת מספר בעצמו - ((x)*(x)) - אינו תלוי בסוג המספר. כל זה בניגוד לפונקציה, שהקוד שלה מופיע בתוכנית פעם אחת, וכל קריאה שלה בתוכנית פורשת קוד המסתעף אליה והיא מצפה לפרמטר מסוג מסוים אחד.

מקרו לעומת פרוצדורות

שימוש במקרו לעומת שימוש בפרוצדורה יוצרת קובץ EXE מהיר יותר, במחיר של הגדלתו. כל זה במקרה שהמקרו נקרא יותר מפעם אחת. מה שנחסך כאן הם הפעולות של יצירת הפרמטרים ופקודות ההסתעפות והחזרה.

הבדל נוסף הוא שמקרו-ים הם דינמיים. הם מסוגלים ליצור קוד שונה במידה מסוימת לפי ערכי פרמטרים.

מבחינה זו הם יותר גמישים מפרוצדורות.

המושגים באסמבלי

אסמבלי מותנה

אילו רצינו לממש באסמבלי את הרעיון של קוד מותנה לצורכי דיבוג שתואר קודם לכן עבור שפת C, זה היה נראה בערך כך:

```
IFDEF MODE_DEBUG
   ..... קוד לצורכי דיבוג
ENDIF
```

הדרך להגדיר את הסמל MODE_DEBUG לצורך פרישת הקוד יהיה ב-command line:

```
tasm /DMODE_DEBUG myprog.asm
```

/D היא האופציה שגורמת לסמל להיות מוגדר.

אמצעי תכנות לאסמבלי מותנה

ביטוי IF - מבצע אסמבלי אם הביטוי <> 0.

ביטוי IFE - מבצע אסמבלי אם הביטוי = 0.

סמל IFB - מבצע אסמבלי אם הסמל = blank.

סמל IFNB - מבצע אסמבלי אם הסמל <> blank.

סמל IFDEF - מבצע אסמבלי אם הסמל מוגדר.

סמל IFNDEF - מבצע אסמבלי אם הסמל לא מוגדר.

IF1 - לבצע במעבר אסמבלי ראשון.

IF2 - לבצע במעבר אסמבלי שני.

IFIDN<arg1,arg2> לבצע אם arg1 זהה ל-arg2.

IFDIF<arg1,arg2> לבצע אם arg1 שונה ל-arg2.

ELSE - לבצע במקרה שה-IFx נכשל.

ENDIF - סוגר את בלוק ה-IFx.

IFx

...
...

ELSE אופציונלי

...
...

ENDIF

<u>מבנה מקרו</u>

רשימת פרמטרים MACRO שם מקרו

.
.
.

גוף המקרו, שעשוי להכיל, מלבד

פקודות אסמבלר גם התיחסיות לפרמטרים

ליבלים לוקליים וכו'

ENDM

לדוגמא, מקרו הפורש קריאה ל- INT 21h, AH = 9

```
Int21h_ah9  MACRO
            MOV AH,9
            INT 21h
            ENDM
```

המקרו הזה יחסוך קריאה לשגרה או כתיבה חוזרת.

בדיקת פריסה של מקרו

יש לבצע tasm יחד עם אופציה /la. ה-tasm יצור, בנוסף לקובץ ה-obj, גם קובץ listing עם סיומת lst שיראה את הפריסה של המקרו. את הקובץ הזה אפשר להדפיס או לקרוא בכל test editor.

לדוגמא,

```
E:\>tasm /la myprog.asm
```

יצור קבצים myprog.obj ו- myprog.lst.

פרמטרים למקרו

פרמטרים למקרו הם בעיקרו של דבר סמלים.

הם עשויים לשמש כסמלים, מספרים או משתנים של הפרה-קומפילר.

המידע מועבר בזמן אסמבלי, וזה יותר יעיל מפרמטרים לפרוצדורה הדורשים תקורה בהצבת ערכים לאוגרים או זכרון.

דוגמא:

```
Int21_ah9  MACRO  Msg
           MOV    DX,OFFSET Msg
           MOV    AH,9
           INT    21h
           ENDM
```

הנחיות מקרו

באמצעי התכנות של אסמבלי מותנה ניתן להשתמש גם במקרו.

ENDM - מסים את הטווח של הנחיות REPT, MACRO, IRP, IRPC.

EXITM - מהווה פקודת עצירה של הרחבה של REPT, MACRO, IRP, IRPC.

לעיתים בדיקת תנאי מסוים מוליך למסקנה של עצירת הפריסה.

IRP - מבצע חזרה על פריסה מתוך רשימה של אופרנדים.

מבנה הפקודה הינה IRP dummy,<oprandlist>.

האופרנדים מופרדים ע"י פסיקים.

IRPC - כמו IRP רק שבמקום רשימת אופרנדים יש מחרוזת והפריסה נעשית על כל תו במחרוזת.

מבנה הפקודה הינו IRPC dummy,string

REPT - גורם לביצוע חוזר של פריסה של קטע ממנו עד ל-ENDM המתאים לו.

שם מקרו PURGE - מוחק מקרו.

הפעלתו על שם מקרו גורם למחיקתו מהטבלאות של האסמבלר.

אם עושים זאת לאחר המופע האחרון של המקרו מקלים על הקומפילציה.

- LOCAL label

ליבלים המוגדרים בתוך מקרו הם ליבלים לכל דבר ומוכרים כפשוטם גם מחוץ למקרו אלא אם כן הם מוגדרים בשורה השניה של המקרו כליבלים מקומיים ע"י ההנחיה LOCAL. אם מקרו מגדיר ליבל שאינו מוגדר כ-LOCAL, לא יהיה ניתן לקרוא למקרו פעמיים, משום שאז יהיה בתוכנית ליבל זהה שיוגדר פעמיים.

ליבל שמוגדר LOCAL עובר המרה. ההמרה תהיה לליבל ??xxxx כאשר xxxx מספר הקסה דצימל בין 0000h ל-FFFFh.

יש להמנע מלהגדיר ליבלים מהצורה הזו.

דוגמא:

```
Wait  MACRO Count
      LOCAL Next                    <---------- חייב להיות כאן
      MOV CX,Count
Next: LOOP Next
      ENDM
```

באמצעי התכנות של אסמבלי מותנה ניתן להשתמש גם במקרו. לדוגמא, המקרו הבא, המישם הדפסה באמצעות INT 21h אופציה 40h, משתמש ב-IFDIF בכדי להמנע מלפרוש את הפקודה MOV CX,CX:

```
    MACRO BuffName, BuffSize
        ; Macro to call INT 21h with AH = 40h, BX = 1
        ; Write to stndout 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
    ;
```

קבועים ומשתנים של הפרה-קומפילר.

הפרה קומפילר יכול לעבוד עם קבועים ומשתנים שלמים.

ראינו כבר שימושים של הנחית האסמבלר EQU, שיוצרת קבועים של הפרה-קומפילר.

למשל
ההנחיה

```
Str_Size EQU 80
        .
        .
        .
        MOV CX,Str_Size
```

גורמת לפריסה של

```
        MOV CX,80
```

כאשר התוכנית מגיעה לאסמבלר.

אפשר בצורה דומה להגדיר משתני פרה-קומפלר ע"י ההנחיה =.

למשל

```
IntVal = 80
   ...
      MOV CX,IntVal
```

יוצר גורם המוחלף המניב אותה תוצאה כמו קודם רק שאת IntVal ניתן לשנות אם רוצים.

אפשר בשלב כלשהוא לכתוב

```
  IntVal = 20
```

ומאותו מקום ואילך IntVal יוחלף ב-20.

אופרטורים על משתני קבועי פרה-קומפילר

אריתמטיים:

+, -, *, /, MOD, SHL, SHR

רציונליים (פעולות ביטיות):

EQ, NE, LT, GT, LE, GE

לוגיים

AND, OR, XOR, NOT (היפוך ביטים).

תוכנית דוגמא dymacl.asm

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 המסתימת 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 macstril.asm

התוכנית הזו נועדה להמחיש שימוש אפשרי של המקרו REPT. שימוש אופייני של REPT הוא להגדיר באסמבלי מבנה נתונים מערכי מאותחל 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 macmoval.asm

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 יממוש ע"י

```
MOV AX,C
```

כך שהקריאות

```
Mov_Ax 0
```

```
Bval = 0
Mov_Ax Bval
```

יממומש 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
```

תוכנית הדוגמא fact1.asm

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 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
               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 macregs4.asm

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 - demostrate macro with paste (&) and IRP
;
    .MODEL SMALL
Push_Reg    MACRO  RLetter
              PUSH     &RLetter&X          ; Paste oprand 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 multmac5.asm

התוכנית הזו ממחישה ב[illegible]ידה רבה מה, פוטנציאלית, ניתן לעשות בעזרת הכלי מקרו, אבל גם במידה ר[illegible]ה למה כמעט ולא משתמשים בזה. מקרו מהסוג של Fast_Mult מאפשר למתכנת לפרוש בתוכניות קוד שונה לחלוטין בהתאם לתנאים מורכבים למדי, הגם [illegible]מדובר בתנאים שחייבים להיות ניתנים לבדיקה בזמן **אסמבלי**. יחד עם זה [illegible]כנות מקרו כזה היא משימה מאד מורכבת, הקוד של המקרו ארוך ומסובך, [illegible]קשה מאד לודא את נכונותו. מקרו כזה יכול בקלות רבה להיות מקור לשגי[illegible]ות תוכנה שקשה לאתר.

התפקיד של Fast_Mult לפר[illegible]ש קוד המכפיל בקבוע (הידוע בזמן אסמבלי) את תוכן AL הידוע רק בזמ[illegible]ריצה, ולחשב את התוצאה לתוך AX. הרעיון כאן הוא לנצל את העובדה שא[illegible]מדובר בקבוע שהוא חזקה של 2, ניתן לבצע את הכפל ע"י הזזה שמאלה של [illegible]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 [illegible]פי בסיס 2.

הקריאה

```
Fast_Mult 25
```

תביא לפרישה של הקוד

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

מה שנחסך ע"י מימוש מקרו [illegible]הזה הוא שהמתכנת לא צריך לשים לב או לבדוק אם הקבוע שאת הכפל שלו הוא[illegible]נדרש לממש הוא חזקה של 2. ברור שבמקרה של 8 ביט זה ממש לא תורם ה[illegible]בה, אבל אם היה מדובר בקבועים 32 ביט מקרו כזה יכול להיות בעל משמעות[illegible]מעשית. כאן מומש המקרה של 8 ביט לצורכי פשטות, שכן המקרו מורכב מכ[illegible]פיק 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 הרלונטית) משתנה דגל Is_power_of_two המאותחל באפס ו-Count המאתחל 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 דגל 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:\>
```