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

מבנה מקרו

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

(היפוך ביטים) NOT, XOR, OR, AND.