את הערך של ה-Status Word ניתן רק ליצא מהמעבד המתמטי, כלומר תוכנית יכולה רק לכתוב את התוכן שלו לזכרון או לאוגר AX. הדבר נעשה ע"י פקודת המכונה FSTSW. הפקודה FSTSW AX היא הפקודה היחידה של המעבד המתמטי שבה האופרנד מחוץ למעבד הינו אוגר רגיל של ה-CPU. בכל מקרה אחר של אופרנד יעד חיצוני וכל מקרה של אופרנד מקור חיצוני מדובר בזכרון.

ה-Control Word מקבל ערכים מהזכרון ע"י פקודה מיוחדת FLDCW. ניתן גם ליצא את הערך לזכרון ע"י פקודת המכונה FSTCW. אם תוכנית משנה לצרכיה את ה-Control Word, היא בדרך כלל תשחזר אותו לערך המקורי לפני סיום (שימור + שיחזור נוסח טבלת הפסיקות). הערך החדש של ה-Control Word יהיה בדר"כ מבוסס על הערך הקודם, או סטנדרטי.

את ה-Tag Word אפשר רק ליצא לזכרון ע"י פקודות מיוחדות (FSAVE, FSTENV) והתוכנית יכולה לפענח את הערכים שם.

רוטינה שמשתמשת במעבד המתמטי צריכה להחזיר את המעבד המתמטי ריק או עם ערך רק ב-ST(0) במידה שהיא מחזירה תשובת פונקציה ממשית. פונקציות Turbo C מחזירות תוצאה ממשית (float, double, long double) בתוך ה-ST(0) ובאחריות התוכנית הקוראת לקרוא ולרוקן אותו משם. על כך נרחיב בהמשך.

קריאה של ערכים בפורמט ממשי מהזכרון לתוך המעבד המתמטי נעשה ע"י הפקודה FLD ל-ST(0). הפקודה מבצעת פעולת PUSH אוטומטית - אם יש ערך ב-ST(0) הוא נדחף ל-ST(1) וכו'. ה-FLD טוען ערכים בפורמט ממשי 80, 64, 32 ביט (כלומר הוא מפרש את תוכן הזכרון כמספרים בפורמט ממשי), תוך התמרה לפורמט ממשי 80 ביט במקרה של אופרנדי זכרון 32 ו-64 ביט. לדוגמא, אם מצב מחסנית המספרים הוא

ST(0) 1.1
ST(1) 4.4
ST(2) 2.2
ST(3) 3.3
ST(4) ריק
ST(5) ריק
ST(6) ריק
ST(7) ריק

וישנו משתנה בזכרון Var1 מוגדר נניח

```
Var1 DQ 6.753
```

ומתבצעת הפקודה

```
FLD Var1
```

במידה ו-Var1 עדיין מכיל את הערך 6.753, המצב של האוגרים אחרי הפעולה תהיה:

```
ST(0)  6.753
ST(1)  1.1
ST(2)  4.4
ST(3)  2.2
ST(4)  3.3
ST(5)  ריק
ST(6)  ריק
ST(7)  ריק
```

FILD מבצע תהליך דומה ל-FLD, אך הוא קורא <u>ערכים בפורמט שלם</u> (16, 32, 64) מהזכרון לתוך אוגרי ה-ST(0) תוך ביצוע המרה לממשי 80 ביט. כלומר, הוא מפרש את תוכן הזכרון כיצוג של מספר <u>שלם</u>.

לדוגמא הקוד הבא:

```
Var2 DW ?

MOV Var2,45
FLD Var2
```

יגרום ל-ST(0) להכיל את הערך הממשי 45.0.

כתיבת ערכים מתוך המעבד המתמטי לזכרון נעשה ע"י הפקודה FST ליעדי זכרון ממשיים 32, 64 ביט או FSTP ליעדי זכרון ממשיים 32, 64, <u>ו-80</u> ביט. הפקודה FIST כותבת ערכים ליעדע זכרון שלמים 16 ו-32 ביט ו- FISTP ליעדי זכרון שלמים 16, 32, <u>ו-64</u> ביט. להוציא יעדי זכרון ממשיים 80 ביט, הפקודה מבצעת את ההתמרות הדרושות.

לדוגמא, אם ST(0) מכיל את הערך 7.91, ומתבצעת הפקודה

```
FIST Var2
```

לתוך Var2 ערך מעוגל של 7.91 (בדרך כלל 8) תלוי בתוכן שדה ה-RC ב-

.Control Word

ריקון איברים נעשה בדרך כלל על ידי גירסאות ה-POP של הפקודות כמו FADDP, FISTP, FSTP למשל. הפקודות הללו מבצעות POP אוטומטי של מחסנית המספרים עם סיום הפעולה שלהם. המשמעות של POP הוא שהערך של ST(0) חדל להיות מיוצג במעבד ויתר הערכים במחסנית המספרים מקודמים לעבר ST(0), כאשר האחרון שמביניהם שהיה לו ערך "משתחרר" או "מתרוקן" (אגב ישנה גם פקודה שרק משחררת אוגר בשם FFREE).
לדוגמא, אם מצב המחסנית הוא

ST(0) 1.1
ST(1) 4.4
ST(2) 2.2
ST(3) 3.3
ST(4) ריק
ST(5) ריק
ST(6) ריק
ST(7) ריק

ומתבצע הפקודה

```
FSTP Var1
```

ל-Var1 יוצב הערך 1.1, ומצב מחסנית המספרים יהיה:

ST(0) 4.4
ST(1) 2.2
ST(2) 3.3
ST(3) ריק
ST(4) ריק
ST(5) ריק
ST(6) ריק
ST(7) ריק

ניתן לבצע חישובים בין 2 אופרנדים בתוך ה-ST(i). אולם ניתן גם לבצע פעולה אריתמטית עם אוגר ST(0) ואופרנד בזכרון. לשם כך יש גירסאות אוגר - זכרון של פקודות אריתמטיות כמו FADD, FIADD וכו'. יחד עם זאת יש לשים לב שבדומה ל-FST ו-FIST הגירסאות הללו אינן תומכות בגדלים הארוכים ביותר. למשל FADD אינו תומך בחיבור עם משתנה זכרון ממשי 80 ביט, FIADD לא תומך בחיבור עם משתנה זכרון שלם 64 ביט. רק חלק מהפקודות של המעבד המתמטי תומכות בפעולות על יעדי זכרון ממשיים 80 ביט או שלמים 64 ביט, כנראה משיקולי יעילות. למשל, אם צריך לחבר שני משתנים ממשיים 80 ביט או שני משתנים שלמים 64 ביט, יש צורך לקרוא את שני הערכים לתוך שני ST(i) שונים (בדרך כלל ST(0), ST(1)) ולבצע FADD או FADDP עליהם.

לדוגמא הפקודה

```
FDIV Var1
```

תבצע את החישוב ST(0) = ST(0) / Var1. אילו רצינו ST(0) = Var1 / ST(0) היינו משתמשים בפקודה FDIVR, מה שנקרא גירסת ה-reverse, כלומר

```
FDIVR Var1
```

ל-FADD ו-FMUL אין גירסאות reverse כי בחיבור וכפל הסדר אינו משנה.

הפקודות האריתמטיות הרגילות (FADD, FSUB, FSUBR, FMUL, FDIV, FDIVR) כאשר הן נכתבות ללא אופרנדים פועלות על ST(1) ו-ST(0) ומבצעות POP אוטומטי (למרות העדר ה-"P" בפקודה). לדוגמא, הפקודה

```
FSUB
```

שקולה לפקודה

```
FSUBP ST(1),ST(0)
```

כלומר הרצת FSUB ללא אופרנדים תבצע ST(1) = ST(1) - ST(0) ו-POP, כלומר תוכן ST(1) ו-ST(0) יעלמו והם יוחלפו בתוצאת ההפחתה של הערך ST(0) מההערך של ST(1) (אילו רצינו ההפך היינו משתמשים ב-FSUBR). התוצאה של ההפחתה תהיה בסוף הפעולה ב-ST(0). לפחות אחד מאוגרי ה-ST(i) ישתחרר, ולערכים האחרים במחסנית המספרים, במידה וישנם, יקודמו לכיוון ה-ST(0). לדוגמא, אם לפני ביצוע ה- FSUB ללא אופרנדים המצב באוגרים היה:

```
ST(0)  6.4
ST(1)  5.2
ST(2)  2.7
ST(3)  3.4
ST(4)  ריק
ST(5)  ריק
ST(6)  ריק
ST(7)  ריק
```

אחרי ה-FSUB המצב באוגרים יהיה:

```
ST(0)  -1.2
ST(1)  2.7
ST(2)  3.4
ST(3)  ריק
ST(4)  ריק
ST(5)  ריק
ST(6)  ריק
ST(7)  ריק
```

השימוש בפקודות האריתמטיות הללו ללא אופרנדים הוא הנוהל הסטנדרטי כאשר אנחנו מעוניינים לבצע פעולה על שני מספרים ומאותו רגע אין לנו עניין במספרים הללו אלא רק בתוצאה. שים לב שהמבנה הזה אינו תופס עבור הפקודה FCOM. FCOM ללא אופנדים אמנם משווה בין ST(0) ל-ST(1) (בסדר הזה שלא כמו ב-FSUB, כלומר כאילו ST(1) - ST(0)), אך משאיר אותם כמות שהם. אם רוצים לשחרר את תוכן ST(0) ו-ST(1) תוך פעולת ההשוואה משתמשים בפקודה FCOMPP.

לדוגמא, אם יש לנו שלושה משתנים:

```
Var3 DT 100.1
Var4 DT 200.2
Var5 DT  ?
```

ואנחנו מעוניינים לחשב Var5 = Var3 / Var4 אנחנו נכתוב:

```
FLD Var3  ;  ST(0) = 100.1
FLD Var4  ;  ST(0) = 200.2, ST(1) = 100.1
FDIV      ;  ST(0) = 0.5
FSTP Var5 ;  Var5 = 0.5
```

בנוסף למתואר לעיל, המעבד המתמטי תומך בפונקציות מתמטיות שונות (FSQRT, FSIN ברדיאנים וכו'). יש פקודות המדמות אריתמטיקה של מספרים שלמים כמו עיגול לשלם (FRNDINT) וחישוב שארית חלוקה (FPREM, FPREM1).

כמו כן יש לו פקודות בקרה שדיברנו קודם (FLDCW למשל). הם מאפשרות יצוא של תוכן האוגרים של המעבד (FSAVE, FSTENV) וכן פקודות בקרה שמשפיעות על תפקוד המעבד (FINIT, FCLEX ...).

מוכמות העברת פרמטרים ממשיים ותוצאות פונקציות ב-Turbo C

פונקציות Turbo C מקבלות פרמטרים ממשיים כמו כל הפרמטרים אחרים, כלומר הם נדחפים למחסנית בתוך כל הפרמטרים האחרים בסדר של מימין לשמאל. אולם בעוד בהעברת פרמטרים אין חידוש עקרוני יש בפירוש הבדל בכל הקשור להחזרת תוצאות פונקציה ממשיות. כאשר פונקצית Turbo C מחזירה תוצאה ממשית בכל גודל (float, double, long double) היא מוחזרת בתוך אוגר ה-ST(0) (כאשר יתר ה-ST(i) חייבים להיות ריקים). באחריות התוכנית הקוראת להשתמש בערך ולרוקן את ST(0).

אם נניח אנחנו מקמפלים ב-Turbo C קוד נוסף:

```
extern double fun(double);
double x, y;
  ....


 y = fun(x);
```

הקוד שמיצר הקומפילר (במודל SMALL) נראה באסמבלי עקרונית משהו כמו:

```
fld qword ptr [bp-8]                          טוען את x
sub sp,8                                      מקצה מקום במחסנית
fstp qword ptr [bp-324]                       מעתיק את x למחסנית
call near ptr _fun                            קורא לפונקציה
add sp,8                                      משחרר שטח הפרמטר
fstp qword ptr [bp-16]        ST(0) ומרוקן את  מציב את התוצאה ל-y
```

לממשי הקומפילר יכולים לעבוד בצורה הזו כי הם מנהלים מעקב על ניצול המחסנית. למתכנת הכותב תוכנית באסמבלי וקורא לפונקציה כנ"ל הגישה הזו בודאי אינה נוחה. המתכנת מן הסתם יממש את x ו-y כמשתנים סטטיים לפי שם, ויממש מצביע לשטח של הפרמטר המיועד. סכמה אפשרית אחת תהיה:

```
x DQ ?
y DQ ?

  ......

FLD x                                   טוען את x
SUB SP,8                                מקצה מקום במחסנית
MOV BX,SP                               גורם ל-BX להצביע לשטח הפרמטר
FSTP QWORD PTR SS:[BX]                  מעתיק את x לשטח הפרמטר
CALL _fun                               קורא ל-fun
ADD SP,8                                משחרר שטח הפרמטר
FSTP y                                  מעתיק את התוצאה ל-y ומרוקן את ST(0)
```

במודל SMALL לא צריך את הציון המפורש של אוגר הסגמנט ("SS:") אך הוא נחוץ במודלים כלליים יותר.

תוכניות דוגמא math_s2.c, mathsu2a.asm, math_s1.c, mathsula.asm
math_s3.c, mathsu4a.asm,

כל שלושת התוכניות המשלבות הללו עושות אותו דבר: מקבלות 2 מספרים מהמשתמש, מחשבות את ההפרש ביניהם ומדפיסות את התוצאה על המסך. ההבדל היא בדיוק שהם עובדים בו: הראשונה היא ב-float, השניה היא ב-double והשלישי היא ב- long double.

בכל שלושת התוכניות ההפרש מחושב בפונקציה הכתובה בקובץ אסמבלי טהור בשם math_sub, אך הן נבדלות בהגדרת סוג הפרמטרים ובסוג התוצאה. math_sub תמיד מחזיר את תוצאת החיסור ב-ST(0), כי כאשר פונקצית Turbo C מחזירה תוצאה ממשית היא מוחזרת ב-ST(0) בלי קשר לשאלה אם מדובר ב-float, double או long double.

המימושים של math_sub ב-mathsula.asm שהוא המימוש של ההפרש ב-float וזו ב-mathsu2a.asm שהוא המימוש של ההפרש ב-double ממשות את אותו האלגוריתם. בשני הריצות מדובר במהלך של טעינת 325.89 לתוך ST(0) ולאחר החסרת 542.67 יהיה ב-ST(0) המספר 216.78-.

שני המימושים הללו נבדלים רק בשתי שורות:

ב-mathsula.asm

```
FLD  DWORD PTR [BP+4]
FSUB DWORD PTR [BP+8]
```

ב-mathsula.asm

```
FLD  QWORD PTR [BP+4]
FSUB QWORD PTR [BP+12]
```

לפיכך ההבדל בין 2 המימושים הוא בצורת ההתיחסות לפרמטרים והצורך להביא בחשבון את גודלם במחסנית. זאת משום ששיטת החישוב כאן עקרונית זהה: התוכנית קוראת את המספר הראשון מהמחסנית לתוך ST(0) ומחסירה ממנו את המספר השני ממקומו במחסנית. זאת משם שניתן להחסיר מספרים ממשיים 32 ו-64 בזיכרון ביט מ-ST(0).
כל זה לא נכון אם מדובר חישוב הפרש של מספרים long double. במקרה הזה חייבים לקרוא את שני המספרים לתוך אוגרי המעבד המתמטי ולבצע את ההפרש שם. ואכן המימוש של החיסר בקובץ mathsu4a.c הינו

```
FLD TBYTE PTR [BP+4]
FLD TBYTE PTR [BP+14]
FSUB
```

שים לב שחשוב הסדר של טעינת האופרנדים וש-FSUB ללא אופרנדים עושה POP אוטומטי ומשחרר את ST(1) כפי שהדבר נחוץ. לפיכך המהלך המתרחש בזמן הריצה של התוכנית הינו:

שלב ראשון

ST(0) 325.89
ST(1) ריק

שלב שני

ST(0) 542.78
ST(1) 325.89

שלב שלישי

ST(0) -216.78
ST(1) ריק

נקודות נוספות שיש לשים לב אליהן:

- בכדי שהאסמבלר יכיר בפקודות מכונה של המעבד המתמטי יש לציין הנחיה מתאימה כמו .387, .87, .8087 וכו'.

- אי אפשר לציין .387 לבד בלי .386 קודם ובצורה דומה להנחיות דומות (.87 היא יוצא דופן בכלל הזה).

- התוכניות האסמבלי נכתבו באסמבלי סטנדרטי לא משום שיש צורך מיוחד בכך ולא משום שיש שיקול מיוחד בהקשר של תכנות המעבד המתמטי, אלא בכדי להמחיש גם את הנושא הזה בקורס.

- כאשר משלבים תוכנית אסמבלי סטנדרטי בתוכנית Turbo C חייבים לבחור את שמות הסגמנטים ש-Turbo C בוחר, וחלק מאותם המאפיינים. סגמנט הקוד של תוכנית האסמבלי נקרא _TEXT משום שזהו השם ש-Turbo C בוחר, ומאפיינים נוספים שחייבים להבחר הם PUBLIC ו-'CODE'. הנוהג לכנות את הסגמנט הביצועי בשם (הלא מוצלח לדעת רבים) text segment הוא כנראה נוהג מראשית ימי המחשבים, שקשור כנראה לשיטת העבודה של תכנות באותם ימים.

```
/* math_s1.c -
        Call math_sub (assembler, math co-processor)   */

#include <stdio.h>

extern float math_sub(float u, float v );

void main(void)
{
  float x, y, z;

  puts("Enter two reals:");
  scanf("%f %f", &x, &y);
  z = math_sub(x, y );
  printf("\n%f - %f = %f\n", x, y, z);

} /* main */
```

```
E:\>math_s1
Enter two reals:
325.89  542.67

325.890015 - 542.669983 = -216.779968

E:\>
```

```
;    mathsu1a.asm - implement asembler C-callable procedure
;     float math_sub( float u,    float v ).
;                          [BP+4]      [BP+8]
;
; Result returned in ST
;
     ASSUME CS:_TEXT
;
     PUBLIC _math_sub
;
_TEXT  SEGMENT BYTE PUBLIC 'CODE'
     .386
     .387
_math_sub  PROC NEAR
;
      PUSH BP                          ; Preserve BP register
      MOV  BP,SP
;
      FLD  DWORD PTR [BP+4]           ; Read u into ST
      FSUB DWORD PTR [BP+8]           ; ST = ST - v
;
      POP  BP                          ; Restore BP
      RET                              ; Return to caller
_math_sub  ENDP
_TEXT  ENDS
      END
```

```
/* math_s2.c -
        Call math_sub (assembler, math co-processor)  */

#include <stdio.h>

extern double math_sub(double u, double v );

void main(void)
{
  double x, y, z;

  puts("Enter two reals:");
  scanf("%lf %lf", &x, &y);
  z = math_sub(x, y );
  printf("\n%lf - %lf = %lf\n", x, y, z);

} /* main */
```

```
E:\>math_s2
Enter two reals:
325.89  542.67

325.890000 - 542.670000 = -216.780000

E:\>
```

```
;    mathsu2a.asm - implement asembler C-callable procedure
;     float math_sub( double u,    double  v ).
;                          [BP+4]       [BP+12]
;
; Result returned in ST
;
     ASSUME CS:_TEXT
;
     PUBLIC _math_sub
;
_TEXT  SEGMENT BYTE PUBLIC 'CODE'
     .386
     .387
_math_sub  PROC NEAR
;
      PUSH BP                          ; Preserve BP register
      MOV  BP,SP
;
      FLD  QWORD PTR [BP+4]          ; Read u into ST
      FSUB QWORD PTR [BP+12]          ; ST = ST - v
;
      POP  BP                          ; Restore BP
      RET                              ; Return to caller
_math_sub  ENDP
_TEXT  ENDS
      END
```

```
/* math_s3.c -
        Call math_sub (assembler, math co-processor)  */

#include <stdio.h>

extern long double math_sub(long double u, long double v );

void main(void)
{
  long double x, y, z;

  puts("Enter two reals:");
  scanf("%Lf %Lf", &x, &y);
  z = math_sub(x, y );
  printf("\n%Lf - %Lf = %Lf\n", x, y, z);

} /* main */
```

```
E:\>math_s3
Enter two reals:
325.89  542.67

325.890000 - 542.670000 = -216.780000

E:\>
```

```
;    mathsu3a.asm - implement asembler C-callable procedure
;    long double math_sub(
;        long double u, long double v ).
;           [BP+4]          [BP+14]
;
; Result returned in ST
;
    ASSUME CS:_TEXT
;
    PUBLIC _math_sub
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
    .386
    .387
_math_sub  PROC NEAR
;
     PUSH BP                        ; Preserve BP register
     MOV  BP,SP
;
     FLD  TBYTE PTR [BP+4]       ; Read u into ST
     FLD  TBYTE PTR [BP+14]      ; ST = v, ST(1) = u
     FSUBP ST(1),ST             ; ST(1) = ST(1) - ST,
                                ; pop (ST = ST(1), ST(1) = Empty)
     POP  BP                      ; Restore BP
     RET                          ; Return to caller
_math_sub  ENDP
_TEXT  ENDS
     END
```

```
;    mathsu4.asm - implement asembler C-callable procedure
;    long double math_sub(
;        long double u, long double v ).
;           [BP+4]          [BP+14]
;
; Result returned in ST
;
    ASSUME CS:_TEXT
;
    PUBLIC _math_sub
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
    .386
    .387
_math_sub  PROC NEAR
;
     PUSH BP                        ; Preserve BP register
     MOV  BP,SP
;
     FLD  TBYTE PTR [BP+4]      ; Read u into ST
     FLD  TBYTE PTR [BP+14]     ; ST = v, ST(1) = u
     FSUB                       ; ST(1) = ST(1) - ST,
                                ; pop (ST = ST(1), ST(1) = Empty)
     POP  BP                     ; Restore BP
     RET                         ; Return to caller
_math_sub  ENDP
_TEXT  ENDS
     END
```