

תוכניות דוגמא math_s2.c, maths2a.asm, math_s1.c, maths1a.asm
math_s3.c, maths4a.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, השנייה 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 פונקציית Turbo C מחזירה תוצאה ממשית היא מוחזרת ב-ST(0) בלי קשר לשאלה אם מדובר ב-double, float או long double.

המימושים של math_sub ב-maths1a.asm והוא המימוש של ההפרש ב-float וזו ב-maths2a.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-maths1a.asm

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

ב-maths2a.asm

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

לפיכך ההבדל בין 2 המימושים הוא בצורת ההתייחסות 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 בזכרון ביט מ-ST(0). כל זה לא נכון אם מדובר בחישוב הפרש של מספרים long double. במקרה הזה חייבים לקרוא את שני המספרים לתוך אוגרי המעבד המתמטי ולבצע את ההפרש שם. ואכן המימוש של החיסור בקובץ maths4a.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 לצין הנחיה מתאימה כמו 387, .87, .8087. וכו'.

- אי אפשר לציין 387. לכד 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 המאפינים. סגמנט הקוד של תוכנית האסמבלי נקרא TEXT_ משום שזהו השם ש-Turbo C בוחר, ומאפינים 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:\>

```

```

; mathsul.asm - implement assembler C-callable procedure
;   float math_sub( float u,   float v ).
;                   [BP+4]     [BP+8]
;
; Result returned in ST
;
; .MODEL SMALL
; .STACK 100h
; PUBLIC _math_sub
;
; .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
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:\>

```

```

;   maths2.asm - implement assembler C-callable procedure
;   float math_sub( double u,   double v ).
;                   [BP+4]      [BP+12]
;
; Result returned in ST
;
; .MODEL SMALL
;
; PUBLIC _math_sub
;
; .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
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:\>

```

```

; maths3.asm - implement assembler C-callable procedure
; long double math_sub(
;     long double u, long double v ).
;     [BP+4]         [BP+14]
;
; Result returned in ST
;
; .MODEL SMALL
;
; PUBLIC _math_sub
;
; .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
END

```



```

; maths31.asm - implement assembler C-callable procedure
; long double math_sub(
;     long double u, long double v ).
;     [BP+4]         [BP+14]
;
; Result returned in ST
;
; .MODEL SMALL
;
; PUBLIC _math_sub
;
; .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
END

```

ביצוע השוואות במעבד המתמטי

אחד התפקידים של המעבד המתימטי הוא לתמוך בהשוואות בין מספרים ממשיים: כלומר לדעת בין שני מספרים (שלפחות אחד מהם, מן הסתם, ממשי) האם הם שווים, או האם הראשון גדול מהשני וכו'. התהליך אינו לחלוטין פשוט.

לפני שנכנס לפרטים של איך זה עובד ולמה זה נכון, להלן מרשם איך לבצע את השוואה בין שני מספרים.

הדרך המקובלת לבצע השוואה בין שני מספרים הוא כלהלן:

1. הבא לפחות את אחד המספרים לתוך אחד מאוגרי ה-ST(i).

2. השתמש באחד מפקודות ההשוואה (FCOMx, FICOMx)

3. בצע את הפקודת המכונה FSTSW AX

4. בצע את פקודת המכונה SAHF

5. במידה וקיימת אפשרות שהאופרנדים אינם ניתנים להשוואה בדוק זאת על ידי בדיקת ה-PF (על ידי הפקודות JP או JNP).

6. המשך כאילו בצעת את פקודת המכונה CMP על אופרנדים שלמים, אותם אתה מעוניין לפרש כמספרים שלמים חסרי סימן.

לדוגמא, הקוד הבא משווה בין שני משתנים ממשיים בזכרון Var1 ו-Var2 (32 או 64 ביט, נניח), כאשר הוא מציב לתוך CX את הערך 1 אם $Var1 > Var2$. לצידו ישנו קוד מקביל למצב שבו Var1 ו-Var2 היו משתנים שלמים 32 ביט:

MOV CX,1

FLD Var1
FCOMP Var2
FSTSW AX
SAHF

JA Skipl
MOV CX,0
Skipl:

MOV CX,1

MOV DX,Var1
CMP DX,Var2

JA Skipl
MOV CX,0
Skipl:

שים לב בשימוש ב-JA ולא ב-JG.

בתוך ה-Status Word קיימים ארבע ביטים הקרויים C0, C1, C2, C3.

כאשר מתבצע השוואה, הביטים C0, C2, C3 הם שמכילים את תוצאת ההשוואה.

C2 משקף מצב שבו האיברים שנעשו בהם השוואה אינם ניתנים להשוואה (ערכים מיוחדים של NaN, אינסוף)

C0 ו-C3 מכילים את תוצאות ההשוואה (בהנחה שהאופרנדים היו תקינים).

C3 הוא למעשה ה-Zero Flag של המעבד המתימטי: $C3 = 1$ פירושו שהאופרנדים היו שווים.

$C0 = 1$ אם האופרנד השני (Var2 בדוגמא) גדול מהראשון. האופרנד הראשון הוא בדרך כלל ST(0).

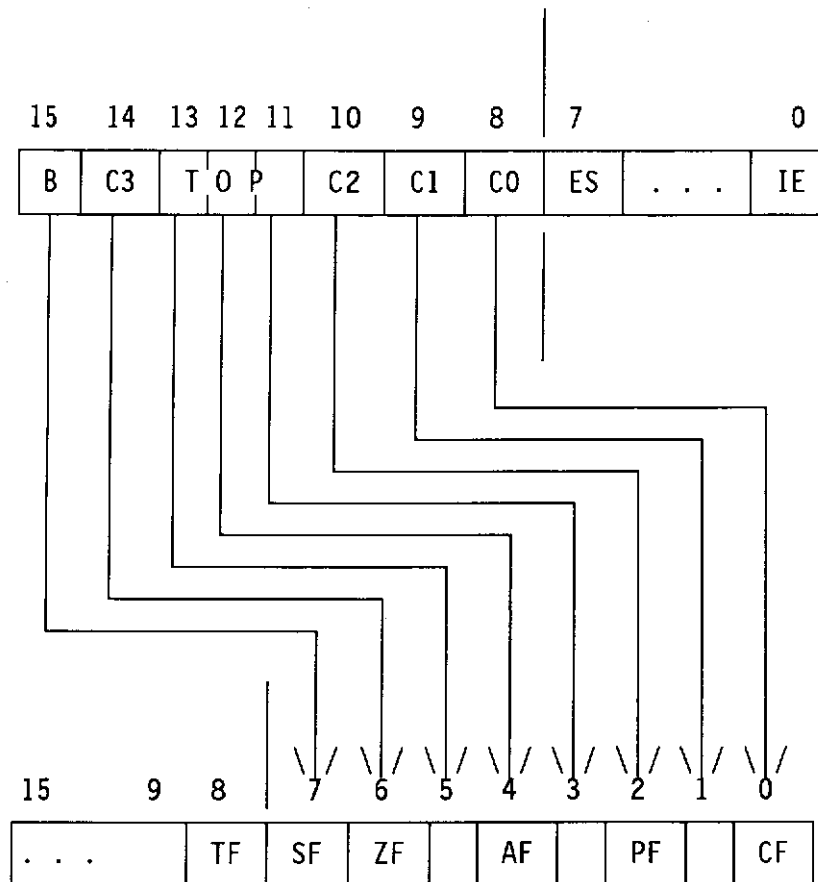
לפיכך אפשר לסכם את משמעות הביטים C0, C2, C3 בטבלה הבאה:

C3	C2	C0	משמעות
0	0	0	אופרנד ראשון < אופרנד שני
0	0	1	אופרנד ראשון > אופרנד שני
1	0	0	אופרנד ראשון == אופרנד שני
1	1	1	אופרנדים בלתי ניתנים להשוואה

הפקודה FSTSW AX מעביר את תוכן ה-Status Word לתוך AX.

הפקודה SAHF מעתיקה את AH לתוך החלק התחתון של אוגר הדגלים. במידה ו-AX מכיל את תוכן ה-Status Word, משמעות הדבר יהיה ש-PF יקבל את הערך של C2, ZF יקבל את הערך של C3, ו-CF יקבל את הערך של C0.

ה-Status Word של המעבד המחמטי



אוגר הדגלים

כהערות אגב נוסף את הפרטים הבאים על C0 - C3:

ל-C1 אין תפקיד בהשוואה בין מספרים.

ל-C1 יש מספר תפקידים. במקרים של ביצוע פעולה חוקית שמחייב עיגול (למשל חישוב 1/3) הערך שלו משקף את סוג העיגול שהתבצע בפועל (עיגול כלפי מעלה C1 = 1, כלפי מטה C1 = 0).

ל-C1 יש גם תפקיד בפענוח מקרה של Stack Fault (0 במקרה של Underflow, 1 במקרה של Overflow). כלומר תפקידו לאפשר הבדלה בין מצב שניסינו להכניס איבר תשיעי למחסנית (Stack Overflow) נניח ע"י FLD לבין מצב שניסינו לרוקן מהמחסנית יותר איברים משהיו בה (Stack Underflow) נניח ע"י FCOMPP

כאשר היה רק ערך אחד במחסנית.

ל-C0, C1, C3 יש גם תפקיד מיוחד בפקודות FPREM ו-FPREM1: הם מכילים את הביטים הכי פחות משמעותיים של התוצאה (C2 מקבל 0 בפקודות הללו). לא נכנס לזה כאן.

סיכום

המנגנון מתנהל כך: ביצוע אחת מפקודות ה-FCOM קובע את ערכי ה-C0, C2, C3, הפקודה FSTSW AX מעבירה את הערכים ל-AX והפקודה SAHF מעבירה אותם משם לדגלים $ZF = C3$, $CF = C0$, $PF = C2$. בדיקת PF מאפשר לנו לגלות שהבדיקה היתה בין אופרנדים שאינם ניתנים להשוואה. במידה שזה לא היה המקרה (או שמראש אין אפשרות כזו) $ZF = 1$ משקף מצב של שיוון ובמידה ו- $ZF = 0$ אזי CF משקף מי מהאופרנדים גדול יותר: $CF = 0$ הראשון, $CF = 1$ השני. קצת אנליזה תראה שזה בדיוק המצב של השוואות בין מספרים חסרי סימן, כאשר CF משקף אפשרות של Borrow של חיסור חסרי סימן.

כמובן שאפשר גם לבדוק את ערכי הביטים ישירות לאחר הפקודה FSTSW AX ע"י הפקודות TEST או AND. אחת מתוכניות הדוגמא שבהמשך מממשת את האפשרות הזו.

תוכניות דוגמא ccomp1.c, mc1.asm, ccomp2.c mc2.asm, ccomp3.c mc3.asm
mc3.asm

מטרת התוכניות המשולבות הזו להמחיש את נושא השוואת 2 מספרים ממשיים במעבד המתמטי.

התוכניות הראשית הכתובה ב-C מקבלת 2 מספרים מהמשתמש כפרמטרים לתוכנית ראשית וקוראת לפונקציה האסמבלי comp להשוות ביניהם. comp מחזירה תוצאה שלמה 0 אם שניהם שווים, 1 אם הפרמטר הראשון גדול מהשני ו-1- אם הפרמטר השני גדול מהראשון. ההנחה היא ששני המספרים הממשיים תקינים, לא מביאים בחשבון את האפשרות של "לא ניתנים להשוואה". הזוג mc2.asm ו-ccomp2.c, float-מישם את החישוב ב-mc1.asm ו-ccomp1.c, double-מישם את החישוב ב-mc3.asm ו-ccomp3.c, long double-ב-long double.

אשר למימוש של עצמם, הם נעשים לפי המרשם של השוואות מספרים במעבד המתמטי שתואר קודם. התוכניות mc1.asm ו-mc2.asm מנצלות את העובדה שניתן בפקודה FCOMP להשוות מספרים ממשיים 32 ו-64 ביט ישירות מהזכרון ואילו mc3.asm, מאחר והיא משווה 2 מספרים 80 ביט, חייבת לטעון את שניהם לאוגרי המעבד ומבצעת שם את הפקודה FCOMPP. שימו לב לסדר הטעינה של המספרים ב-mc3.asm. על מנת לשמר ב-mc3.asm את אותה פקודת הסתעפות (JBE) כמו ב-mc1.asm ו-mc2.asm, התוכנית טוענת למעבד קודם את המספר השני ורק אחריו את המספר הראשון. בתוכנית הדבר בא לידי ביטוי בשורות

```
FLD TBYTE PTR [BP+14] ; ST(0) = p1
FLD TBYTE PTR [BP+4]
FCOMPP ; Compare p1 with p2, pop twice
```

זאת משום שהפקודה FCOMPP משווה את ST(0) לעומת ST(1).

מעניין לעשות אנלוגיה למצב שבו נניח היינו רוצים לממש רוטינה דומה המחזירה אותה תשובה נניח אם היינו משווים 2 מספרים אי שליליים 32 ביט.

FLD DWORD PTR [BP+4]	MOV EAX,[BP+4]
FCOMP DWORD PTR [BP+8]	CMP EAX,[BP+8]
FSTSW AX	
SAHF	
JBE Not_Above	JBE Not_Above
MOV AX,1	MOV AX,1
JMP SHORT Finish	JMP SHORT Finish
Not_Above:	Not_Above:
JE Equal	JE Equal
MOV AX,-1	MOV AX,-1
JMP SHORT Finish	JMP SHORT Finish
Equal:	Equal:
MOV AX,0	MOV AX,0
Finish:	Finish:
POP BP	POP BP
RET	RET

הערה: "JMP SHORT" מנחה את האסמבלר לבחור את הגרסה של -128 + 127 של JMP. במקרה של הסתעפות קדימה זה מיעל במשהו את הקובץ הבינארי שנוצר. זה לא קשור בהכרח למעבד המתמטי. לא נכנס לזה כאן.

```

/* ccomp1.c - compare real numbers */

#include <stdio.h>
#include <stdlib.h>

extern int comp( float p1, float p2 );

void main(int argc, char *argv[])
{
    float x, y;
    int cmp_flag;

    if (argc < 3)
    {
        fprintf(stderr, "Usage: ccomp1 real1 real2\n");
        exit(1);
    }

    sscanf(argv[1], "%f", &x);
    sscanf(argv[2], "%f", &y);

    cmp_flag = comp(x,y);

    if ( cmp_flag == 0 )
        printf("%f == %f\n\n", x, y);
    else
        if ( cmp_flag > 0 )
            printf("%f > %f\n\n", x, y);
        else
            if ( cmp_flag < 0 )
                printf("%f < %f\n\n", x, y);
} /* main */

```

```

E:\>tcc ccomp1.c mc1.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
ccomp1.c:
mc1.asm:
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland
International

```

```

Assembling file:    mc1.ASM
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   398k

```

```

Turbo Link Version 5.0 Copyright (c) 1992 Borland International

```

```

    Available memory 4104476

```

```

E:\>CCOMP1.EXE 432.65 988.22
432.649994 < 988.219971

```

```

E:\>

```

220


```

; mcl.asm - Compare within the math coprocessor
;
;      int _comp( float p1, float p2)
;                  [BP+4]    [BP+8]
;
; If p1 = p2,      Return 0
; If p1 > p2,      Return 1
; If p1 < p2,      Return -1
;
; FCOMP = Compare and Pop
; Return  C3  C2  C1  C0  Meaning
;         0   0   ?   0   ST > Source
;         0   0   ?   1   ST < Source
;         1   0   ?   0   ST == Source
;         1   1   ?   1   ST is not comparable to source
;
; SAHF - Store AH into flags
;
; Status Word
; -----
; 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
; ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
; | B | C3 | TOP | C2 | C1 | C0 | ES | SF | PE | UE | OE | ZE | DE | IE |
; ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
;
PUBLIC _comp
;
.MODEL SMALL
;
.CODE
.386
.387
_comp PROC NEAR
    PUSH    BP
    MOV     BP, SP
    FLD     DWORD PTR [BP+4] ; ST(0) = p1
    FCOMP   DWORD PTR [BP+8] ; Compare ST with p2, pop
    FSTSW   AX               ; Store comparison result in AX
;
    SAHF    ; Store C3, C2, C0 in Flags register (ZF, PF, CF)
;
; Proceed as we did CMP on two unsigned integer operands
;
;      ; Was ST > ST(1) ( p1 > p2 ) ?
JBE     Not_Above ; No, more tests
    MOV     AX, 1 ; Yes, return 1
    JMP     SHORT Finish
Not_Above: ; ST <= ST(1)
;      ; Was ST < ST(1) ( p1 < p2 ) ?
JE      Equal ; No, ST == ST(1)
    MOV     AX, -1 ; Yes, return 0
    JMP     SHORT Finish
;
Equal:
    MOV     AX, 0 ; ST == ST(1) or incomparable
;
Finish:
    POP     BP
    RET
ENDP _comp
;
;
END

```

```

/* ccomp2.c - compare real numbers */

#include <stdio.h>
#include <stdlib.h>

extern int comp( double p1, double p2 );

void main(int argc, char *argv[])
{
    double x, y;
    int cmp_flag;

    if (argc < 3)
    {
        fprintf(stderr, "Usage: ccomp1 real1 real2\n");
        exit(1);
    }

    sscanf(argv[1], "%lf", &x);
    sscanf(argv[2], "%lf", &y);

    cmp_flag = comp(x,y);

    if ( cmp_flag == 0 )
        printf("%lf == %lf\n\n", x, y);
    else
        if ( cmp_flag > 0 )
            printf("%lf > %lf\n\n", x, y);
        else
            if ( cmp_flag < 0 )
                printf("%lf < %lf\n\n", x, y);
} /* main */

```

```

E:\>tcc ccomp2.c mc2.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
ccomp2.c:

```

```

mc2.asm:
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland
International

```

```

Assembling file:    mc2.ASM
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   398k

```

```

Turbo Link Version 5.0 Copyright (c) 1992 Borland International

```

```

    Available memory 4103660

```

```

E:\>ccomp2.exe 321.2 -400.9
321.200000 > -400.900000

```

```

E:\>

```

222

```

; mc2.asm - Compare within the math coprocessor
;
;      int _comp( double p1, double p2)
;                  [BP+4]      [BP+12]
;
; If p1 = p2,      Return 0
; If p1 > p2,      Return 1
; If p1 < p2,      Return -1
;
;FCOMP = Compare and Pop
;Return   C3  C2  C1  C0  Meaning
;          0   0   ?   0   ST > Source
;          0   0   ?   1   ST < Source
;          1   0   ?   0   ST == Source
;          1   1   ?   1   ST is not comparable to source
;
; SAHF - Store AH into flags
;
; Status Word
; -----
; 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
; ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
; | B | C3 | TOP | C2 | C1 | C0 | ES | SF | PE | UE | OE | ZE | DE | IE |
; ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
;
;
.MODEL SMALL
.CODE
.386
.387
_comp PROC NEAR
PUBLIC _comp
PUSH BP
MOV BP,SP
FLD QWORD PTR [BP+4] ; ST(0) = p1
FCOMP QWORD PTR [BP+12] ; Compare ST with p2, pop
FSTSW AX ; Store comparison result in AX
;
SAHF ; Store C3, C2, C0 in Flags register (ZF, PF, CF)
;
; Proceed as we did CMP on two unsigned integer operands
;
; Was ST > ST(1) ( p1 > p2 ) ?
JBE Not_Above ; No, more tests
MOV AX,1 ; Yes, return 1
JMP SHORT Finish
Not_Above: ; ST <= ST(1)
; Was ST < ST(1) ( p1 < p2 )?
JE Equal ; No, ST == ST(1)
MOV AX,-1 ; Yes, return 0
JMP SHORT Finish
;
Equal:
MOV AX,0 ; ST == ST(1) or incomparable
;
Finish:
POP BP
RET
ENDP _comp
;
;
END

```

```

; mc2.asm - Compare within the math coprocessor
;
;      int _comp( double p1, double p2)
;                  [BP+4]    [BP+12]
;
;  If p1 = p2,      Return  0
;  If p1 > p2,      Return  1
;  If p1 < p2,      Return -1
;
;FCOMP = Compare and Pop
;Return   C3  C2  C1  C0  Meaning
;         0   0   ?   0   ST > Source
;         0   0   ?   1   ST < Source
;         1   0   ?   0   ST == Source
;         1   1   ?   1   ST is not comparable to source
;
; SAHF - Store AH into flags
;
;  Status Word
;  -----
;  15  14  13 12 11  10   9   8   7   6   5   4   3   2   1   0
;  ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
;  | B | C3 | TOP | C2 | C1 | C0 | ES | SF | PE | UE | OE | ZE | DE | IE |
;  ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
;
; .MODEL SMALL
; .CODE
; .386
; .387
_comp PROC NEAR
PUBLIC _comp
PUSH BP
MOV BP,SP
FLD QWORD PTR [BP+4] ; ST(0) = p1
FCOMP QWORD PTR [BP+12] ; Compare ST with p2, pop
FSTSW AX ; Store comparison result in AX
;
SAHF ; Store C3, C2, C0 in Flags register (ZF, PF, CF)
;
; Proceed as we did CMP on two unsigned integer operands
;
;      ; Was ST > ST(1) ( p1 > p2 ) ?
JBE Not_Above ; No, more tests
MOV AX,1 ; Yes, return 1
JMP SHORT Finish
Not_Above: ; ST <= ST(1)
;      ; Was ST < ST(1) ( p1 < p2 )?
JE Equal ; No, ST == ST(1)
MOV AX,-1 ; Yes, return 0
JMP SHORT Finish
;
Equal:
MOV AX,0 ; ST == ST(1) or incomparable
;
Finish:
POP BP
RET
ENDP _comp
;
;
END

```

```

/* ccomp3.c - compare real numbers */

#include <stdio.h>
#include <stdlib.h>

extern int comp( long double p1, long double p2 );

void main(int argc, char *argv[])
{
    long double x, y;
    int cmp_flag;

    if (argc < 3)
    {
        fprintf(stderr, "Usage: ccomp1 real1 real2\n");
        exit(1);
    }

    sscanf(argv[1], "%Lf", &x);
    sscanf(argv[2], "%Lf", &y);

    cmp_flag = comp(x, y);

    if ( cmp_flag == 0 )
        printf("%Lf == %Lf\n\n", x, y);
    else
        if ( cmp_flag > 0 )
            printf("%Lf > %Lf\n\n", x, y);
        else
            if ( cmp_flag < 0 )
                printf("%Lf < %Lf\n\n", x, y);
} /* main */

```

```

E:\users\eytan\asm\braude>tcc -v ccomp3.c mc3.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
ccomp3.c:
mc3.asm:
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland
International

Assembling file:    mc3.ASM
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   429k

Turbo Link Version 5.0 Copyright (c) 1992 Borland International

    Available memory 4138012

E:\>CCOMP3.EXE 654.6 32.87
654.600000 > 32.870000

```

E:\>

225

```

; mc3.asm - Compare within the math coprocessor
;
;      int _comp( long double p1, long double p2)
;                  [BP+4]          [BP+14]
;
; If p1 = p2,      Return 0
; If p1 > p2,      Return 1
; If p1 < p2,      Return -1
;
; FCOMP = Compare and Pop
; Return  C3  C2  C1  C0  Meaning
;         0   0   ?   0   ST > Source
;         0   0   ?   1   ST < Source
;         1   0   ?   0   ST == Source
;         1   1   ?   1   ST is not comparable to source
;
; SAHF - Store AH into flags
;
; Status Word
; -----
; 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
; ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
; | B | C3 | TOP | C2 | C1 | C0 | ES | SF | PE | UE | OE | ZE | DE | IE |
; ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
;
;
; .MODEL SMALL
; .CODE
; .386
; .387
; _comp PROC NEAR
; PUBLIC _comp
; PUSH BP
; MOV BP, SP
; FLD TBYTE PTR [BP+14] ; ST(0) = p1
; FLD TBYTE PTR [BP+4]
; FCOMPP ; Compare p1 with p2, pop twice
; FSTSW AX ; Store comparison result in AX
;
; SAHF ; Store C3, C2, C0 in Flags register (ZF, PF, CF)
;
; ; Proceed as we did CMP on two unsigned integer operands
;
; ; Was ST > ST(1) ( p1 > p2 ) ?
; JBE Not_Above ; No, more tests
; MOV AX, 1 ; Yes, return 1
; JMP SHORT Finish
;
; Not_Above: ; ST <= ST(1)
; ; Was ST < ST(1) ( p1 < p2 ) ?
; JE Equal ; No, ST == ST(1)
; MOV AX, -1 ; Yes, return 0
; JMP SHORT Finish
;
;
; Equal:
; MOV AX, 0 ; ST == ST(1) or incomparable
;
;
; Finish:
; POP BP
; RET
; ENDP _comp
;
;
; END

```