אריתמטיקה של מספרים שלמים במעבד המתמטי

מלבד התמיכה במספרים ממשיים (64, 32 ו-80 ביט) מעבד המתמטי "תומך" באריתמטיקה של מספרים שלמים 16, 32 ו-64 ביט. מאז ה-386 התמיכה במספרים שלמים 16 ו-32 ביט אולי לא כל כך חשוב, אבל עבור מספרים 64 ביט התמיכה כאן היא משמעותית.

ה"תמיכה" במספרים שלמים במעבד המתמטי הוא במובן מסוים מאד. היצוג של מספרים באוגרי ה-ST(i) הוא תמיד ממשי - תמיד. התמיכה של המעבד מספרים שלמים בא לידי ביטוי בשתי צורות:

1. המרות: המעבד יודע לקרוא אופרנדים בזכרון בפורמט שלם ולהמיר אותם לממשי, והוא יודע גם לכתוב אופרנדים לזכרון בפורמט שלם תוך המרה מהפורמט הממשי שמשתמש המעבד.

2. ישנם מספר (קטן) של פקודות מכונה בתוך המעבד המדמות פעולות על מספרים שלמים בפורמט הממשי שהמעבד משתמש. דוגמאות לכך הם FRNDINT, FPREM, FPREM1.

תמיכה מסוג 1. הוא העיקר. המעבד מאפשר לכתוב קוד הקרוא מספר שלם לתוך המעבד, לפעול עליו כמספר ממשי ולכותב את התוצאה לזכרון כמספר שלם. גם 1. וגם 2. מסתמכים למעשה על העובדה שהיצוג של מספר שלם כממשי הוא מדויק. היצוג של של המספר השלם 23 הוא 23.0, ולא 22.9999 ולא 23.0001 וכו'. בעוד שקריאת מספרים שלמים לתוך המעבד היא, איפוא, פעולה נעדרת בעיות, כתיבה של מספר ממשי לזכרון עשויה להיות כרוכה בעיגול המספר, והמתכנת עשוי להיות במצב שעליו לדאוג לכך שיהיה עיגול מהסוג שהוא מעונין בו, אם על ידי הוספה / הפחתה של חצי או מניפולציה של ה-Control Word של המעבד המתמטי. כל הפקודות הללו פועלות על אופרנד בזכרון, שכן בתוך המעבד - הכל ממשי.

הפקודה הקוראת לתוך המעבד מהזכרון מספר בפורמט שלם נקרא FILD (להבדיל מ-FLD שקוראת מהזכרון מספרים בפורמט ממשי). הפקודות הכותבות לזכרון מספר בפורמט שלם נקראים FIST, FISTP (להבדיל מ-FST, FSTP שכותבות מספר בפורמט ממשי). עבור אופרנדים של זכרון 32- 16 ביט ניתן גם לבצע פעולות ארתמטית שאחד האופרנדים (אופרנד מקור בלבד, לא יעד, כלומר אופרנד קריאה בלבד) הוא מספר בזכרון בפורמט ממשי: FIADD, FISUB, FISUBR, FIMUL, FIDIV, FIDIVR. אופרנדי זכרון שלמים בני 64 ביט נתמכים אך ורק בפקודות FILD ו-FISTP. יתר הפקודות תומכות רק באופרנדי זכרון 16 ו-32 ביט בלבד.

לפיכך, אם יש לי 3 משתנים בשם Var1, Var2, ו-Var3 ואני רוצה, נניח, לסכם

את Var1 ו-Var2 כמספרים בפורמט שלם ולהציב את התוצאה בפורמט שלם ב-Var3, הדרך לעשות זאת הרא:

אם Var1, Var2 ו-Var3 הם בגודל 16 או 32 ביט:

```
FILD Var1
FIADD Var2
FISTP  Var3
```

ואם הם בגודל 64 ביט:

```
FILD Var1
FILD Var2
FADD
FISTP Var3
```

שימו לב ש-FADD הוא סכום ממשי.

חיסור וכפל ניתן לפתור באותה צורה משום שהפעולות הללו משמרות את אופי המספרים כמספרים שלמים ביצוג ממשי, דבר שאינו כך בחילוק למשל. במקרה של חילוק המצב יותר מורכב.

תוכניות דוגמא call_idl.c, idiv_mo2.asm, idiv_mo3.asm

מה שיש כאן הוא מימוש חדש של התוכנית call_idl.c שראינו את המימוש שלו בעזרת רוטינת אסמבלי המשתמשת באוגרי המעבד הרגילים. כאן נראה את המימוש של האריתמטיקה של השלמים בעזרת המעבד המתמטי. מטרת תוכניות הדוגמא הללו הוא להמחיש אריתמטיקה של שלמים במעבד המתמטי.

לצורך הדוגמא, נניח שאנחנו מעוניינים שהעיגול של החלוקה ללא שארית תהיה תמיד למינוס אינסוף. ההבדל בין שני המימושים של idiv_mod בקבצים idiv_mo2.asm ו- idiv_mo3.asm הוא הצורה שהפונקציה מודאת עיגול כלפי מטה. ב- idiv_mo2.asm העיגול היא ע"י הפחתה של 0.499999 ואילו ב- idiv_mo3.asm זוהי הצבה של הערך 01 ל-RC שב-Control Word של המעבד.

ניהול ה-Control Word בקובץ idiv_mo3.asm

ניהול ה-Control Word ב-idiv_mo3.asm נעשה על ידי שמירת ערכו המקורי לתוך משתנה מיועד לכך Save_CW ושיחזור של הערך לפני החזרה. הצבת הערך 01 ל-RC נעשה על סמך הערך הנוכחי של ה-Control Word על ידי הצבתו למשתנה נפרד New_CW, ביצוע פעולות ביטיות עליו וטעינתו חזרה ל-Control Word. השימור של ה-Control Word נעשה בפקודה

```
FSTCW Save_CW
```

בתחילת הרוטינה והשיחזור נעשה ע"י הפקודה

```
FLDCW Save_CW
```

השינוי הזמני ב-Control Word נעשה ע"י הפקודות

```
FSTCW New_CW
AND  New_CW,1111001111111111B
OR   New_CW,0000010000000000B
FLDCW New_CW
```

הכופה על ה-RC את הערך 01 ע"י איפוס הערך הנוכחי שלו על ידי פקודת ה-AND וכתיבת 1 לביט הנמוך של ה-RC ע"י פקודת ה-OR.

שתי המימושים מבצעים את משימתם על ידי טעינת המספרים השלמים לאוגרי המעבד ע"י הפקודה FILD וחלוקה ע"י FIDIVR. בתוך המעבד האריתמטיקה היא ממשית אך התמרת התוצאה נעשית ע"י הפקודה FISTP. חישוב שארית החלוקה נעשה ע"י טעינת 2 המספרים למעבד ושימוש בפקודה FPREM.

הפקודה

```
FILD WORD PTR [BP+6]
```

טוען את המכנה. בכדי להתגונן מפני חלוקה באפס, הפקודה

```
FTST
```

משווה את המספר עם אפס. הפקודות

```
FSTSW AX
SAHF
```

מעבירה את תוצאות ההשוואה לאוגר הדגלים. משם אנחנו ממשיכים כמו ב-idiv_mol.asm בכל הקשור לטיפול בחלוקה באפס (שכן מענין אותנו כאן רק שיוויון עם עם אפס או אי שיוויון, וכאן אין הבדל בין חסרי סימן לעם סימן).

במקרה התקין שבו לא נדרשנו לחלק באפס ההמשך הוא ביצוע החלוקה ע"י הפקודה

```
FIDVR WORD PTR [BP+4]
```

בשלב זה התוצאה ממשית, ע"י הפקודות

```
MOV BX,[BP+8]
FISTP WORD PTR [BX]
```

התוכנית מציבה את התוצאה ביעד תוך ביצוע ההמרה והעיגול הדרושים, ו-ST מתרוקן.

בדוגמת ריצה של התוכנית המהלך הזה מתבטא בכך ש-ST(0) מקבל את הערך 44.0, לאחר מכן מוחלף ב-2.38636, ליעד נכתב המספר השלם 2 ו-ST(0) מתרוקן.

חישוב שארית החלוקה נעשה ע"י הפקודות

```
FILD WORD PTR [BP+6]
FILD WORD PTR [BP+4]
FPREM
```

שים לב שהסדר של ה-FILD-ים חשוב. עכשיו ב-ST(0) נמצא השארית ואילו ב-ST(1) נמצא המכנה. עכשיו בעקבות ביצוע הפקודות

```
MOV BX,[BP+10]
FISTP WORD PTR [BX]
```

מוצב התוצאה ליעד ומתרוקן ST(1). ערך המכנה נמצא עכשיו ב-ST(0). חובה לרוקן אותו והדבר נעשה ע"י הפקודה

```
FFREE ST
```

בדוגמת הריצה יש לנו מציאות שבו האוגרים ST(0) ו-ST(1) עוברים שרשרת מצבים ממצב של שניהם ריקים ל-

שלב ראשון, אחרי FILD WORD PTR [BP+6],

ST(0) 44.0
ST(1) ריק

שלב שני, אחרי FILD WORD PTR [BP+4],

ST(0) 105.0
ST(1) 44.0

שלב שלישי, אחרי FPREM,

ST(0) 17.0
ST(1) 44.0

שלב רביעי, אחרי FISTP WORD PTR [BX],
ליעד מוצב הערך השלם 17 ואילו האוגרים של המעבד עוברים למצב

ST(0) 44.0
ST(1) ריק

עד לשלב החמישי (אחרי FFREE ST) שניהם ריקים שוב.

```
/* call_id1.c - call assembler subroutine idiv_mod.asm from C program */

#include <stdio.h>

 extern  int idiv_mod(int Num, int Denom,  int *Q,  int *Rem);

 void main()
    {
      int Num, Denom, Q, Rem, No_Zero_Divide;

      printf("\Enter Numerator, Denominator\n:");
      scanf("%d %d",&Num, &Denom);
      No_Zero_Divide = idiv_mod(Num,Denom,&Q,&Rem);
      if  (No_Zero_Divide)
          printf("\n %d div %d = %d, mod(%d,%d) = %d\n",
                    Num, Denom, Q, Num, Denom, Rem);
      else
         printf("\nError: Zero Divide.\n");

     } /* main */
```

```
E:\>tcc call_id1.c idiv_mo2.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
call_id1.c:
idiv_mo2.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   idiv_mo2.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  418k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4136272

E:\>call_id1.exe
Enter Numerator, Denominator
:105 44

 105 div 44 = 2, mod(105,44) = 17

E:\>
```

```
;  idiv_mo2.asm - Assembler implementation of C-callable function
;                 idiv_mod.
;
  .MODEL SMALL
; Static Variables
;
  .DATA
;
Half  DQ 0.4999999999999
;
    .CODE
    .386
    .387
    ; Implementation of C callable function ...
    ;    ...  int idiv_mod(int Num, int Denom,  int *Q,  int *Rem)
    ;                      [BP+4]    [BP+6]     [BP+8]    [BP+10]
    ;  Compute Q := |_ Num / Denom _|  ,Rem := MOD(Num, Denom)
    ;  function  idiv_mod  returns 0 if Denom = 0 (illegal ..
    ;  ... division by zero), 1 otherwise
    ;
     PUBLIC _idiv_mod
_idiv_mod  PROC NEAR
     PUSH BP         ; Preserve BP
     MOV BP,SP       ; Set BP to point to Parameter area
     FILD WORD PTR [BP+6]   ; ST(0) := Denom
     FTST      ; Denom = 0 ?
     FSTSW  AX ;  AX = Status word
     SAHF      ;  Copy to flags register
     JNZ Cont        ; No, continue regular operation
           ; Yes, Denom = 0
         FFREE ST
         MOV AX,0     ; Return value := 0
         JMP Done              ; Skip following code
Cont:      ; Denom <> 0
     FIDIVR WORD PTR [BP+4]     ; ST = Num / ST, ST = Num / Denom
     FSUB  Half           ; Subtract 1/2 to ensure rounding down
     MOV  BX,[BP+8]       ; BX := Offset Q
     FISTP  WORD PTR [BX]         ; *Q := ST
     FILD  WORD PTR [BP+6]     ; ST = Denom
     FILD  WORD PTR [BP+4]     ; ST = Num, ST(1) = Denom
     FPREM                     ; ST = ST mod ST(1)
     MOV  BX,[BP+10]      ; BX := Offset Rem
     FISTP WORD PTR  [BX]         ; *Rem := ST
     FFREE ST              ;
     MOV  AX,1    ; Ensure return value = 1
Done:
     POP  BP              ; Restore BP register
     RET
_idiv_mod  ENDP
;
;
     END
```

```
;  idiv_mo3.asm - Assembler implementation of C-callable function idiv_mod.
;
;
;  Control Word
;  --------------
;  15  14  13  12  11 10  9  8  7   6    5    4    3   2    1    0
;  ---+---+---+---+--+--+--+--+---+---+----+----+----+----+----+----+
; | R | R | R | I |  RC | PC  | R | R | PM | UM | OM | ZM | DM | IM |
;  ---+---+---+---+--+--+--+--+---+---+----+----+----+----+----+----+
;
  .MODEL SMALL
;
  ; Static Variables
  .DATA
;
Save_CW  DW ?
New_CW   DW ?
;
;
    .CODE
    .386
    .387
    ; Implementation of C callable function ...
    ;     ...  int idiv_mod(int Num, int Denom,  int *Q,  int *Rem)
    ;                          [BP+4]    [BP+6]     [BP+8]    [BP+10]
    ;  Compute Q := |_ Num / Denom _|  ,Rem := MOD(Num, Denom)
    ;  function  idiv_mod  returns 0 if Denom = 0 (illegal ..
    ;  ... division by zero), 1 otherwise
    ;
     PUBLIC _idiv_mod
_idiv_mod  PROC NEAR
     PUSH BP         ; Preserve BP
     MOV BP,SP       ; Set BP to point to Parameter area
;
     FSTCW   Save_CW          ;  Store status in Mem16
     FSTCW   New_CW           ;  Store status in Mem16
     AND  New_CW,1111001111111111B ; Erase existing RC
     OR   New_CW,0000010000000000B ; Set RC to 01
                                   ; (Round towards  -infinity)
     FLDCW New_CW                  ; Set New CW
;
     FILD WORD PTR [BP+6]  ; ST(0) := Denom
     FTST     ; Denom = 0 ?
     FSTSW  AX  ; Transfer SW to AX
     SAHF     ;  Copy to flags register
     JNZ Cont        ; Demon != 0
          ; Yes, Denom = 0
         FFREE ST
         MOV AX,0    ; Return value := 0
         JMP Done             ; Skip following code
Cont:      ; Denom != 0
```

```
      FIDIVR WORD PTR [BP+4]       ; ST = Num / ST, ST = Num / Denom
      MOV  BX,[BP+8]       ; BX := Offset Q
      FISTP  WORD PTR [BX]        ; *Q := ST
      FILD  WORD PTR [BP+6]     ; ST = Denom
      FILD  WORD PTR [BP+4]     ; ST = Num, ST(1) = Denom
      FPREM                     ; ST = ST mod ST(1)
      MOV  BX,[BP+10]      ; BX := Offset Rem
      FISTP WORD PTR  [BX]         ; *Rem := ST
      FFREE ST             ; Free ST(0)
      MOV  AX,1    ; Ensure return value = 1
Done:
;
      FLDCW Save_CW        ; Restore control word to original value
      POP  BP              ; Restore BP register
      RET
_idiv_mod  ENDP
;
;
      END
```

התוכנית idiv_mo9.asm מממשת את idiv_mod בגרסת ה-32 ביט ,
תוך מימוש האריתמטיקה השלמה במעבד המתמטי בדומה ל-idiv_mo2.asm.
מאחר ומדובר בשלמים 32 ביט,

התוכנית של idiv_mo9.asm נבדלת מ-idiv_mo2.asm בכך ש:
- ה-casting לתוך המחסנית הוא DWORD PTR במקום WORD PTR.
- ההסטים בתוך המחסנית של כל הפרמטרים למעט Num גדלים, Denom ב-2 ו-Q ו-Rem ב-4, בדיוק כמו ב-idiv_mo7.asm.

המבנה של פקודות התוכנית ב-idiv_mo9.asm זהה ל-idiv_mo2.asm משום שניתן לבצע פעולות בשלמים עם המעבד ישירות לזיכרון 16 ו-32 ביט. רק ב-64 ביט זה אינו נתמך.

```
/* call_id2.c - call assembler subroutine idiv_mod.asm from C program */

#include <stdio.h>

extern  int idiv_mod(long int Num, long int Denom,  long int *Q,  long
int *Rem);

 void main()
    {
      long int Num, Denom, Q, Rem;
      int  No_Zero_Divide;

      printf("\Enter Numerator, Denominator\n:");
      scanf("%ld %ld",&Num, &Denom);
      No_Zero_Divide = idiv_mod(Num,Denom,&Q,&Rem);
      if  (No_Zero_Divide)
          printf("\n %ld div %ld = %ld, mod(%ld,%ld) = %ld\n",
                    Num, Denom, Q, Num, Denom, Rem);
      else
         printf("\nError: Zero Divide.\n");

    } /* main */
```

```
E:\>tcc -v call_id2.c idiv_mo9.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
call_id2.c:
idiv_mo9.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   idiv_mo9.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  429k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4149256

E:\>CALL_ID2.EXE
Enter Numerator, Denominator
:700065 55000

 700065 div 55000 = 12, mod(700065,55000) = 40065

E:\>
```

```
;  idiv_mo9.asm - Assembler implementation of C-callable function idiv_mod.
;
   .MODEL SMALL
   ; Static Variables
;
   .DATA
;
Half  DQ 0.4999999999999
;
     .CODE
     .386
     .387
     ; Implementation of C callable function ...
     ;     ...  int idiv_mod(long int Num, long int Denom,
     ;                          [BP+4]         [BP+8]
     ;                                                  int *Q,  int *Rem)
     ;                                                  [BP+12]    [BP+14]
     ;  Compute Q := |_ Num / Denom _|   ,Rem := MOD(Num, Denom)
     ;  function  idiv_mod  returns 0 if Denom = 0 (illegal ..
     ;  ... division by zero), 1 otherwise
     ;
      PUBLIC _idiv_mod
_idiv_mod  PROC NEAR
     PUSH BP         ; Preserve BP
     MOV BP,SP       ; Set BP to point to Parameter area
     FILD DWORD PTR [BP+8]   ; ST(0) := Denom
     FTST      ; Denom = 0 ?
     FSTSW  AX ;  AX = Status word
     SAHF      ;  Copy to flags register
     JNZ Cont        ; No, continue regular operation
           ; Yes, Denom = 0
         FFREE ST
         MOV AX,0     ; Return value := 0
         JMP Done              ; Skip following code
Cont:       ; Denom <> 0
     FIDIVR DWORD PTR [BP+4]              ; ST = Num / ST, ST = Num / Denom
     FSUB  Half           ; Subtract 1/2 to ensure rounding down
     MOV  BX,[BP+12]       ; BX := Offset Q
     FISTP  DWORD PTR [BX]        ; *Q := ST
     FILD  DWORD PTR [BP+8]     ; ST = Denom
     FILD  DWORD PTR [BP+4]     ; ST = Num, ST(1) = Denom
     FPREM                     ; ST = ST mod ST(1)
     MOV  BX,[BP+14]      ; BX := Offset Rem
     FISTP DWORD PTR  [BX]         ; *Rem := ST
     FFREE ST             ;
     MOV  AX,1    ; Ensure return value = 1
Done:
     POP  BP              ; Restore BP register
     RET
_idiv_mod  ENDP
;
;
     END
```

תוכניות דוגמא fderiv1.c, fderiv2a.c, fdl.asm

התפקיד העיקרי של התוכניות הללו הינו להמחיש את השימוש בפוינטר לפונקציה כפרמטר, דבר נפוץ במיוחד בחישובים נומריים.

בהנתן פונקציה נתונה נניח ע"י איזה קוד חישובי, אנחנו רוצים לעשות קירוב נומרי של הנגזרת שלה בנקודה. זהו תסריט שמתרחש כאשר יודעים לחשב פונקציה מסוימת אבל לא בהכרח יודעים איך הוא נראה אנליטית, למשל הוא פתרון של משוואה שמחושבת באופן נומרי או חישוב נומרי אחר.

חישוב הקירוב הנומרי מבוסס על כך שהנגזרת הוא הגבול של

$$f'(x) = \lim_{h\to 0} (f(x+h) - f(x-h))/(2h)$$

כלומר אם נחשבאת הקירוב לנגדזרת ע"י הנוסחה (*):

$$f'(x) = (f(x+h) - f(x-h))/(2h) \quad (*)$$

עבור h מספיק קטן, נקבל קירוב לנגזרת. השאלה היא איזה h לבחור. ככל ש-h קטן יותר החישוב יהיה יותר נכון מבחינה מתמטית אולם אם נבחר h קטן מדי יגרום לבעיות של אובדן דיוק . חוץ מזה, h חייב להיות קטן יחסית ל-x ולאו דווקא קטן במושגים מוחלטים. פתרון אפשרי הוא להתחיל אם h = x/2 וכל הזמן להקטין את h ע"י חלוקה ב-2.0 עד שנגיע לכך שהחישובים מתחילים להתכנס, כלומר 2 חישובים עוקבים של הנוסחה (*) ההפרש ביניהם בערכם המוחלט קטן מאפסילון נבחר. בתוכניות הללו נבחר אפסילון כערך המולט של f(x) חלקי 16384.0. אילו החישובים היו בדיוק יותר גבוה מ-float היינו מחלקים ביותר.

השיקולים הועמדים מאחרי האלגוריתם הזה קשורים לנושאים של דיוק חישובי שלא כאן המקום לפרטם.

התוכנית fderiv1.c היא מימוש האלגוריתם בשפת C. התוכנית המשולבת fderiv2a.c ו-fdl.asm ממשים את האלגוריתם הנומרי באסמבלי.

התוכניות fderiv2b.c, fderiv2c.c, fd2.asm, fd3.asm

התוכניות הללו הן המקבילות של fderiv2a.c ו-fd1.asm עבור מספרים double ו-long double.

הקבצים fderiv2b.c ו-fd2.asm מממשים גזירה נומרית עבור מספרים double.

הקבצים fderiv2c.c ו-fd3.asm מממשים גזירה נומרית עבור מספרים long double.

מאחר ואנחנו עובדים כאן בדיוקים יותר גדולים, בחישוב אפסילון אנחנו מחלקים בערכים יותר גדולים, 131072.0 עבור double ו-2097152.0 עבור long double.

```
/* fderiv1.c - approximate derivative function */

#include <stdio.h>
#include <math.h>

float approx_fderiv(float (*f)(float), float x)
{
  float h, fd0, fd1, eps;

  h = fabs(x/2.0);
  fd1 = ((*f)(x+h) - (*f)(x-h))/(2*h);
  eps = x/8192.0;

  do {
      fd0 = fd1;
      h = h/2.0;
      fd1 = ((*f)(x+h) - (*f)(x-h))/(2*h);
     } while(fabs(fd0 - fd1) > eps );

  return fd1;

} /* approx_deriv */


float f(float x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

float real_fderiv(float x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %f\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %f\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
E:\>tcc -v fderiv1.c
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv1.c:
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4103660

E:\>FDERIV1.EXE
approx_deriv(5.0) =  58.001526
real_fderiv(5.0) = 58.000000

E:\>
```

```
/* fderiv2a.c - approximate derivative function */

#include <stdio.h>
#include <math.h>

extern float approx_fderiv(float (*f)(float), float x);

float f(float x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

float real_fderiv(float x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %f\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %f\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
E:\>tcc fderiv2a.c fd1.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv2a.c:
fd1.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   fd1.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  429k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4141520

E:\>FDERIV2A.EXE
approx_deriv(5.0) = 58.001526
real_fderiv(5.0) = 58.000000

E:\>
```

```
;
; fd1.asm - implement numerical differentiation
;
;
      .MODEL SMALL
      .DATA
h   DD ?
two DD 2.0
fd0 DD 0.0
eps_const DD 16384.0
eps   DD 0.0
temp DD 0.0
      .CODE
   ;
   ;     float approx_fderiv (float (*f)(float), float x)
   ;                                [BP+4]           [BP+6]
   ;
         .386
         .387
        PUBLIC _approx_fderiv
_approx_fderiv  PROC    NEAR
         PUSH    BP
         MOV     BP,SP
         FLD DWORD PTR [BP+6]
         FDIV two
         FABS
         FSTP h
   ;
   ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
   ;
         FLD     DWORD PTR [BP+6]
         FADD    h
         FSTP    temp
         PUSH    temp                  ; Push x+h
         CALL    WORD PTR [BP+4]       ; ST(0) = f(x+h)
         ADD     SP,4     ;  Free Parameter
         FLD     DWORD PTR [BP+6]
         FSUB    h
         FSTP    temp
         PUSH    temp                  ; Push x-h
         CALL    WORD PTR [BP+4]       ; ST(0) = f(x-h), ST(1) = f(x+h)
         ADD     SP,4    ;  Free Parameter
         FSUB                          ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
         FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
         FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
         FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
         FSTP fd0
         PUSH DWORD PTR [BP+6]
         CALL WORD PTR [BP+4]
         ADD SP,4
         FDIV eps_const
         FABS
         FSTP eps
```

```
Do1:
        FLD h
        FDIV two
        FSTP h
   ;
   ;      compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
   ;
        FLD     DWORD PTR [BP+6]
        FADD    h
        FSTP    temp
        PUSH    temp                    ; Push x+h
        CALL    WORD PTR [BP+4]         ; ST(0) = f(x+h)
        ADD     SP,4     ;  Free Parameter
        FLD     DWORD PTR [BP+6]
        FSUB    h
        FSTP    temp
        PUSH    temp                    ; Push x-h
        CALL    WORD PTR [BP+4]         ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,4    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FLD ST  ; ST(0) = (f(x+h)-  f(x-h))/2h, ST(1) = (f(x+h)- f(x-h))/2h
        FLD fd0
        FSUB    ; ST(0) = current - fd0
        FABS    ;  ST(0) = | current - fd0 |
        FCOMP eps
        FSTSW AX
        SAHF
        FSTP  fd0
        JAE Do1
   ;
        FLD fd0
        MOV     SP,BP
        POP     BP
        RET
_approx_fderiv  ENDP
   ;
        END
```

```
/* fderiv2b.c - approximate derivative function */

#include <stdio.h>
#include <math.h>

extern double approx_fderiv(double (*f)(double), double x);

double f(double x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

double real_fderiv(double x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %lf\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %lf\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
E:\>tcc fderiv2b.c fd2.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv2b.c:
fd2.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   fd2.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  429k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4141520

E:\>FDERIV2B.EXE
approx_deriv(5.0) = 58.000095
real_fderiv(5.0) = 58.000000

E:\>
```

```
;
; fd2.asm - implement numerical differentiation
;
;
      .MODEL SMALL
      .DATA
h   DQ ?
two DD 2.0
fd0 DQ 0.0
eps_const DQ 131072.0
eps   DQ 0.0
temp DQ 0.0
      .CODE
    ;
    ;     double approx_fderiv (double (*f)(double), double x)
    ;                                [BP+4]              [BP+6]
    ;
          .386
          .387
        PUBLIC _approx_fderiv
_approx_fderiv  PROC    NEAR
         PUSH    BP
         MOV     BP,SP
         FLD QWORD PTR [BP+6]
         FDIV two
         FABS
         FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
         FLD     QWORD PTR [BP+6]
         FADD    h
         FSTP    temp
         PUSH    DWORD PTR temp+4    ; Push x+h
         PUSH    DWORD PTR temp
         CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
         ADD     SP,8     ;  Free Parameter
         FLD     QWORD PTR [BP+6]
         FSUB    h
         FSTP    temp
         PUSH    DWORD PTR temp+4; Push x-h
         PUSH    DWORD PTR temp
         CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
         ADD     SP,8    ;  Free Parameter
         FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
         FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
         FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
         FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
         FSTP fd0
         PUSH DWORD PTR [BP+10]
         PUSH DWORD PTR [BP+6]
         CALL WORD PTR [BP+4]
         ADD SP,8
         FDIV eps_const
         FABS
         FSTP eps
```

```
Do1:
        FLD h
        FDIV two
        FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
        FLD     QWORD PTR [BP+6]
        FADD    h
        FSTP    temp
        PUSH    DWORD PTR temp+4    ; Push x+h
        PUSH    DWORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,8     ;  Free Parameter
        FLD     QWORD PTR [BP+6]
        FSUB    h
        FSTP    temp
        PUSH    DWORD PTR temp+4; Push x-h
        PUSH    DWORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,8    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FLD ST  ; ST(0) = (f(x+h)-  f(x-h))/2h, ST(1) = (f(x+h)- f(x-h))/2h
        FLD fd0
        FSUB    ; ST(0) = current - fd0
        FABS    ;  ST(0) = | current - fd0 |
        FCOMP eps
        FSTSW AX
        SAHF
        FSTP  fd0
        JAE Do1
    ;
        FLD fd0
        MOV     SP,BP
        POP     BP
        RET
_approx_fderiv  ENDP
    ;
        END
```

```
/* fderiv2c.c - approximate derivative function */

#include <stdio.h>
#include <math.h>

extern long double approx_fderiv(long double (*f)(long double),
                                                    long double x);

long double f(long double x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

long double real_fderiv(long double x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %Lf\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %Lf\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
E:\>tcc fderiv2c.c fd3.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv2c.c:
fd3.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   fd3.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  429k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4141520

E:\>FDERIV2C.EXE
approx_deriv(5.0) = 58.000006
real_fderiv(5.0) = 58.000000

E:\>
```

```
;
; fd3.asm - implement numerical differentiation
;
;
      .MODEL SMALL
      .DATA
h   DT ?
two DD 2.0
fd0 DT 0.0
eps_const DT 2097152.0
eps   DT 0.0
temp DT 0.0
      .CODE
    ;
    ; long double approx_fderiv (long double (*f)(double), long  double x)
    ;                                    [BP+4]                  [BP+6]
    ;
         .386
         .387
        PUBLIC _approx_fderiv
_approx_fderiv  PROC    NEAR
         PUSH    BP
         MOV     BP,SP
         FLD  TBYTE PTR [BP+6]
         FDIV two
         FABS
         FSTP h
    ;
    ;      compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
         FLD     TBYTE PTR [BP+6]
         FLD     h
         FADD
         FSTP    temp
         PUSH    DWORD PTR temp+6     ; Push x+h
         PUSH    DWORD PTR temp+2
         PUSH    WORD PTR temp
;
         CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
         ADD     SP,10     ;  Free Parameter
         FLD     TBYTE PTR [BP+6]
         FLD     h
         FSUB
         FSTP    temp
         PUSH    DWORD PTR temp+6     ; Push x-h
         PUSH    DWORD PTR temp+2
         PUSH    WORD PTR temp
;
         CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
         ADD     SP,10    ;  Free Parameter
         FSUB                      ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
         FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
         FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
         FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
         FSTP fd0
```

```
        PUSH DWORD PTR [BP+12]
        PUSH DWORD PTR [BP+8]
        PUSH WORD PTR [BP+6]
        CALL WORD PTR [BP+4]
        ADD SP,10
        FLD eps_const
        FDIV
        FABS
        FSTP eps

Do1:
        FLD h
        FDIV two
        FSTP h
   ;
   ;      compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
   ;
        FLD     TBYTE PTR [BP+6]
        FLD     h
        FADD
        FSTP    temp
        PUSH    DWORD PTR temp+6     ; Push x-h
        PUSH    DWORD PTR temp+2
        PUSH    WORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,10     ;  Free Parameter
        FLD     TBYTE PTR [BP+6]
        FLD     h
        FSUB
        FSTP    temp
        PUSH    DWORD PTR temp+6     ; Push x-h
        PUSH    DWORD PTR temp+2
        PUSH    WORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,10    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FLD ST  ; ST(0) = (f(x+h)-  f(x-h))/2h, ST(1) = (f(x+h)- f(x-h))/2h
        FLD fd0
        FSUB    ; ST(0) = current - fd0
        FABS    ;  ST(0) = | current - fd0 |
        FLD   eps
        FCOMPP
        FSTSW AX
        SAHF
        FSTP  fd0
        JNAE Do1  ; Reversed logic
   ;
        FLD fd0
        MOV     SP,BP
        POP     BP
        RET
_approx_fderiv  ENDP
   ;
        END
```