תוכניות דוגמא call_id2.c, idiv_ml1.asm, idiv_ml2.asm

התוכניות הללו הן גרסאות 32 בית לתוכנית call_id1.c,
idiv_mo2.asm, idiv_mo3.asm בהבדלים המשתמעים: כלומר ה-casting הוא
DWORD PTR ומיקומם של הפרמטרים במחסנית מביא בחשבון את העובדה שעכשיו
Num ו-Denom הם 4 בתים.

```
/* call_id2.c - call assembler subroutine idiv_mod32.asm from C program */

#include <stdio.h>

 extern  int idiv_mod32(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_mod32(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 */
```

```
C:\>tcc call_id2.c idiv_m11.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
call_id2.c:
idiv_m11.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   idiv_m11.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  398k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4116560

C:\>call_id2.exe
Enter Numerator, Denominator
:700000 -66666

 700000 div -66666 = -10, mod(700000,-66666) = 33340

C:\>
```

```
;  idiv_m12.asm - Assembler implementation of C-callable function idiv_mod.
;
; Static Variables
   ASSUME CS:_TEXT, DS:_DATA
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
Save_CW DW ?
New_CW DW ?
_DATA ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
     .386
     .387
     ; Implementation of C callable function ...
     ;     ...  int idiv_mod32(long int Num, long int Denom,
     ;                              [BP+4]          [BP+8]
     ;   long 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_mod32
_idiv_mod32  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
```

```
      FSTCW Save_CW
      MOV AX,Save_CW
      MOV New_CW,AX
      OR    New_CW,0000110000000000B ; Set RC to 11 (Chop)
      FLDCW New_CW                   ; Set New CW
      FIDIVR DWORD PTR [BP+4]              ; ST = Num / ST, ST = Num / Denom
      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
      FLDCW Save_CW         ; Restore control word to original value
Done:
      POP   BP              ; Restore BP register
      RET
_idiv_mod32   ENDP
;
_TEXT ENDS
;
      END
```

```
;  idiv_m11.asm - Assembler implementation of C-callable function idiv_mod.
;
; Static Variables
   ASSUME CS:_TEXT, DS:_DATA
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
Half  DQ 0.499999999
_DATA ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
     .386
     .387
     ; Implementation of C callable function ...
     ;     ...  int idiv_mod32(long int Num, long int Denom,
     ;                              [BP+4]          [BP+8]
     ;   long 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_mod32
_idiv_mod32  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 Cont1        ; No, continue regular operation
            ; Yes, Denom = 0
          FFREE ST
          MOV AX,0     ; Return value := 0
          JMP Done              ; Skip following code
Cont1:       ; Denom <> 0
      FIDIVR DWORD PTR [BP+4]          ; ST = Num / ST, ST = Num / Denom
      FTST   ; Is it positive?
      FSTSW AX
      SAHF
      JB  Neg1  ; Skip if negative
      FSUB  Half           ; Subtract 1/2 to ensure rounding down
      JMP  Cont2
Neg1:
     FADD Half             ; Add 1/2 to ensure rounding down
Cont2:
```

```
      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_mod32   ENDP
;
_TEXT ENDS
;
      END
```

תוכניות דוגמא

fderivl.c,
fderiv2a.c fd2a.asm,
fderiv2b.c fd2b.asm,
fderiv2c.c fd2c.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 היא נראית אנליטית, למשל כאשר היא 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 נחשב את הקירוב לנגדזרת ע"י הנוסחה (*):

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

עבור h מספיק קטן, נקבל (בתאוריה לפחות) 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 כתוצאה משגיאות עיגול. חוץ מזה, h חייב להיות קטן יחסית ל-x ולאו דווקא קטן במושגים מוחלטים. פתרון אפשרי הוא להתחיל אם h = |x/2.0| 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 המוחלט של f(x) חלקי 8192.0. אילו החישובים היו בדיוק יותר גבוה מ-float היינו מחלקים ביותר.

השיקולים העומדים 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-fd2a.asm, fderiv2b.c ו-fd2b.asm ו-fderiv2ca.c ו-fd2ca.asm ממשות את האלגוריתם הנומרי באסמבלי הנקרא מ-C עבור מספרים ממשיים 32, 64 ו-80 ביט בהתאמה.

```
/* 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 */
```

```
C:\>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:  398k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4116560

C:\>call_id1.exe
Enter Numerator, Denominator
:-43 10

 -43 div 10 = -4, mod(-43,10) = -3

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

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

C:\>
```

```
;  idiv_mo2.asm - Assembler implementation of C-callable function idiv_mod.
;
   ; Static Variables
    ASSUME CS:_TEXT, DS:_DATA
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
Half  DQ 0.4999999999
_DATA ENDS
;
_TEXT SEGMENT BYTE PUBLIC '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 Cont1         ; No, continue regular operation
            ; Yes, Denom = 0
          FFREE ST
          MOV AX,0     ; Return value := 0
          JMP Done              ; Skip following code
Cont1:       ; Denom <> 0
      FIDIVR WORD PTR [BP+4]          ; ST = Num / ST, ST = Num / Denom
      FTST   ; Is it positive?
      FSTSW AX
      SAHF
      JBE  Neg1  ; Skip if negative
      FSUB  Half           ; Subtract 1/2 to ensure rounding down
      JMP  Cont2
Neg1:
      FADD Half            ; Add 1/2 to ensure rounding down
Cont2:
;
      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
;
_TEXT ENDS
;
      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 |
;  ---+---+---+---+--+--+--+--+---+----+----+----+----+----+----+----+
;
  ASSUME CS:_TEXT, DS:_DATA
;
  ; Static Variables
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
Save_CW  DW ?
New_CW   DW ?
;
_DATA  ENDS
;
_TEXT SEGMENT BYTE PUBLIC '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 Save_CW
      MOV AX,Save_CW
      MOV New_CW,AX          ;  Store status in New_CW
      OR   New_CW,0000110000000000B ; Set RC to 11 (Chop)
      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
;
_TEXT ENDS
;
      END
```

```
;  idiv_m11.asm - Assembler implementation of C-callable function idiv_mod.
;
; Static Variables
   ASSUME CS:_TEXT, DS:_DATA
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
Half  DQ 0.499999999
_DATA ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
     .386
     .387
     ; Implementation of C callable function ...
     ;    ...  int idiv_mod32(long int Num, long int Denom,
     ;                             [BP+4]          [BP+8]
     ;   long 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_mod32
_idiv_mod32  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 Cont1        ; No, continue regular operation
            ; Yes, Denom = 0
          FFREE ST
          MOV AX,0     ; Return value := 0
          JMP Done              ; Skip following code
Cont1:       ; Denom <> 0
      FIDIVR DWORD PTR [BP+4]          ; ST = Num / ST, ST = Num / Denom
      FTST   ; Is it positive?
      FSTSW AX
      SAHF
      JB  Neg1  ; Skip if negative
      FSUB  Half           ; Subtract 1/2 to ensure rounding down
      JMP  Cont2
Neg1:
      FADD Half            ; Add 1/2 to ensure rounding down
Cont2:
```

```
      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_mod32  ENDP
;
_TEXT ENDS
;
      END
```

```
;  idiv_m12.asm - Assembler implementation of C-callable function idiv_mod.
;
; Static Variables
   ASSUME CS:_TEXT, DS:_DATA
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
Save_CW DW ?
New_CW DW ?
_DATA ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
     .386
     .387
     ; Implementation of C callable function ...
     ;    ...  int idiv_mod32(long int Num, long int Denom,
     ;                             [BP+4]          [BP+8]
     ;   long 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_mod32
_idiv_mod32  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
```

```
      FSTCW Save_CW
      MOV AX,Save_CW
      MOV New_CW,AX
      OR    New_CW,0000110000000000B ; Set RC to 11 (Chop)
      FLDCW New_CW                   ; Set New CW
      FIDIVR DWORD PTR [BP+4]              ; ST = Num / ST, ST = Num / Denom
      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
      FLDCW Save_CW         ; Restore control word to original value
Done:
      POP   BP              ; Restore BP register
      RET
_idiv_mod32   ENDP
;
_TEXT ENDS
;
      END
```