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 בזיכרון בפורמט שלם ולהמיר אותם לממשי, והוא יודע גם לכתוב אופרנדים לזיכרון 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 לזיכרון 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 בעתיות, כתיבה של מספר ממשי לזיכרון 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 בזיכרון, שכן בתוך המעבד - הכל ממשי.

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

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 קיצוץ (3.6 ל-3, 3.6- ל-3-). ההבדל בין שני המימושים של idiv_mod בקבצים idiv_mo2.asm ו- idiv_mo3.asm הוא הצורה שהפונקציה דואגת לעיגול קיצוץ. ב- idiv_mo2.asm העיגול היא ע"י הפחתה של 0.499999 כאשר מדובר במספר חיובי והוספה של 0.499999 כאשר מדובר במספר שלילי, בהנחה שהמעבד מעגל לקרוב ביותר. ב-idiv_mo3.asm זוהי הצבה של הערך 11 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 11 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

```
MOV AX,Save_CW,AX
MOV New_CW,AX
OR   New_CW,0000110000000000b
FLDCW New_CW
```

הכופה על ה-RC את הערך 11 ע"י דריסת הערך הנוכחי של התמונה של ה-

RC ב-New_CW ב-11 ולאחר מכן טעינת New_CW ל-Control Word של המעבד.

מימוש האריתמטיקה בשני המימושים

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
```

מעבירות את תוצאות ההשוואה לאוגר הדגלים. משם אנחנו ממשיכים כמו ב-idiv_mol.asm בכל הקשור לטיפול בחלוקה באפס (שכן מעניין 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

ST(0)-ו 2 השלם המספר נכתב ליעד, 2.38636-ב מוחלף מכן לאחר, 44.0
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) ששניהם 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 */
```

```
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
```

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_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
```