

תוכניות דוגמא call_id1.c, idiv_mo4.asm, idiv_mo5.asm

התוכניות הללו משמשות כמובן כדוגמא לקריאה לפרוצדורת אסמבלי מתוך תוכנית C.

ניהול המחסנית של התוכניות תוארו קודם. להלן יתר התיעוד של התוכנית.

מה שהתוכנית המשולבת, call_id1.c עם כל אחד מתוכניות ה-asm, idiv_mo4.asm או idiv_mo5.asm, עושה היא לקבל 2 מספרים שלמים (לאו דווקא חיוביים) ולחשב את החלוקה ללא שארית ושארית החלוקה של 2 המספרים הללו ולהדפיס אותם. התוכנית מוגנת מפני בקשה לחלוקה באפס.

מימוש החלוקה נעשה ע"י פקודת המכונה IDIV, ואפשר לראות מהריצות שמבחינתה של פקודת המכונה הזו היא ששארית החלוקה של 105 ב-44 הוא 17, שארית החלוקה של 105 ב-44 הוא 17 ושארית החלוקה של 105 ב-44 הוא 17. מתכנת הכותב תוכנית המחשבת שארית חלוקה של מספרים שעשויים להיות שליליים חייב לבדוק אם המוסכמות הללו מקובלים עליו.

קימפול תכנית המשלבת קבצי מקור ב-C ואסמבלי ניתן לעשות על ידי תוכנת tcc.exe (בתנאי שיש לך גם את tasm.exe) או bcc.exe. עקרונית כותבים את רשימת הקבצים שממנה מורכבת התוכנית כאשר התוכנית הראשית חייבת להיות ראשונה. במקרה שלנו, קימפול התוכנית המורכבת מהקבצים call_id1.c ו-idiv_mo4.asm יהיה:

```
tcc call_id1.c idiv_mo4.asm
```

תוצאת הקימפול של השורה הזו (בתנאי שאין שגיאות) תהיה לפי השם של הקובץ הראשון, כלומר ייוצר call_id1.exe.

אם רוצים להכין את הקובץ לדיבוג ב-Turbo Debugger אזי פשוט מוסיפים את האופציה "-v" כלומר:

```
tcc -v call_id1.c idiv_mo4.asm
```

התוכניות idiv_mo4.asm ו-idiv_mo5.asm

ההבדל בין שתי הקבצים הללו שב-idiv_mo5.asm אני משתמש באוגרים SI ו-DI, לכן אני חייב לשמר אותם בקובץ הזה ולשחזר אותם לפני החזרה, מה שאין צורך ב-idiv_mo4.asm.

נקודות שיש לשים לב אליהם:

- כל שם המוגדר בתוכניות C או שהתוכנית מתיחסת אליו, באסמבלי חייבים להתייחס אליו עם קו תחתי ("_") מוביל. מהסיבה הזו בקבצי האסמבלי שמה של הרוטינה הנקראת מ-C היא "_idiv_mod".

- `_idiv_mod` צריכה לחלק מספר 16 ביט במספר 16 ביט, אבל הפקודה IDIV מחלקת 32 ביט (DX:AX) ב-16 ביט. על מנת לבצע את המשימה עלינו "להרחיב" את המספר ב-AX לתוך DX. אילו היה מדובר במספרים חסרי סימן היה מדובר כאן בהצבת 0 ל-DX, אבל במספרים עם סימן צריך להביא בחשבון את הסימן של AX: אם הוא חיובי, צריך להציב 0 ל-DX, ואם הוא שלילי, צריך להציב 1 לכל הביטים של DX. הדבר הוא למעשה הצבת ביט הסימן של AX לכל הביטים של DX. יש פקודת מכונה שעושה בשבילנו בדיוק את זה: CWD. גרסאות דומות של הפקודה הזו:

CBW	AL	ל-AX	הרחב את
CWD	AX	ל-DX:AX	הרחב את
CWDE	AX	ל-EAX	הרחב את
CWQ	EAX	ל-EDX:EAX	הרחב את

- שימו לב למבנה:

```
MOV AX,0
JMP Done
```

....

```
MOV AX,1
```

Done:

.....

```
POP BP
RET
```

המבנה הזה מבטיח שעם החזרה לקוד הקורא AX מכיל את הערך הנכון של תוצאת הפונקציה.

```
/* 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("\nEnter 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_mo4.asm
```

```
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
```

```
call_id1.c:
```

```
idiv_mo4.asm:
```

```
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International
```

```
Assembling file:    idiv_mo4.ASM
```

```
Error messages:     None
```

```
Warning messages:   None
```

```
Passes:             1
```

```
Remaining memory:   395k
```

```
Turbo Link Version 5.0 Copyright (c) 1992 Borland International
```

```
    Available memory 4111504
```

```
E:\>call_id1.exe
```

```
Enter Numerator, Denominator
```

```
:105 44
```

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

```
E:\>call_id1.exe
```

```
Enter Numerator, Denominator
```

```
:105 -44
```

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

```
E:\>call_id1.exe
```

```
Enter Numerator, Denominator
```

```
:-105 44
```

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

```
E:\>call_id1.exe
```

```
Enter Numerator, Denominator
```

```
:-105 -44
```

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

```
E:\>
```

```

; idiv_mo4.asm - Assembler implementation of
;
;                                     C-callable function idiv_mod.
;

.MODEL SMALL
.CODE
; 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
MOV CX,[BP+6]    ; CX := Denom
CMP CX,0        ; Denom = 0 ?
JNE Cont        ; No, continue regular operation
                ; Yes, Denom = 0
MOV AX,0        ; Return value := 0
JMP Done        ; Skip following code
Cont:           ; Denom <> 0
MOV AX,[BP+4]    ; AX := Num
CWD             ; DX:AX := AX
IDIV CX         ; AX := DX:AX / CX, DX := MOD(AX,CX)
MOV BX,[BP+8]    ; BX := Offset Q
MOV [BX],AX     ; *Q := AX
MOV BX,[BP+10]   ; BX := Offset Rem
MOV [BX],DX     ; *Rem := DX
MOV AX,1        ; Ensure return value := 1
Done:
POP BP          ; Restore BP register
RET
_idiv_mod ENDP
END

```

```

; idiv_mo5.asm - Assembler implementation of
;                  C-callable function idiv_mod.
;

.MODEL SMALL
.CODE
; 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
PUSH SI          ; Preserve register variables
PUSH DI          ;

;
MOV SI,[BP+6]    ; SI := Denom
CMP SI,0         ; Denom = 0 ?
JNE Cont        ; No, continue regular operation
; Yes, Denom = 0
MOV AX,0         ; Return value := 0
JMP Done         ; Skip following code
Cont:           ; Denom <> 0
MOV AX,[BP+4]    ; AX := Num
CWD             ; DX:AX := AX
IDIV SI         ; AX := DX:AX / SI, DX := MOD(AX,SI)
MOV DI,[BP+8]    ; DI := Offset Q
MOV [DI],AX      ; *Q := AX
MOV DI,[BP+10]   ; DI := Offset Rem
MOV [DI],DX      ; *Rem := DX
MOV AX,1        ; Ensure return value := 1
Done:
;
POP DI          ; Restore register variables
POP SI          ;
POP BP          ; Restore BP register
RET
_idiv_mod ENDP
END

```

תוכניות דוגמא call_id2.c, idiv_mod6.asm

בתוכנית המשולבת call_id1.c, idiv_mod1.asm כל הפרמטרים היו 2 בתים כי זה הגודל של הן int וכן פוינטרים בהקשר הזה של קומפילציה של טורבו C. נראה שזה לא תמיד כך בהמשך הקורס. כאשר מחשבים את המיקום של פרמטרים צריך תמיד להביא בחשבון את גודל הפרמטרים, שכמובן לא יהיו תמיד 2 בתים. זה יקרה למשל אם היינו עובדים עם פרמטרים long int במקום int. זה מה שקורה בדוגמא הבאה, התוכנית המשולבת call_id2.c, idiv_mod6.asm. ההגדרה של idiv_mod32 הינה:

```
extern int idiv_mod32(long int Num, long int Denom,  
    long int *Q, long int *Rem);
```

מאחר ו-Num ו-Denom הם עכשיו 32 ביט, תמונת המחסנית היא עכשיו כזו:

BP ישן	← BP
IP	[BP+2]
תוכן Num	[BP+4]
תוכן Denom	[BP+8]
כתובת Q	[BP+12]
כתובת Rem	[BP+14]

בנוסף לכך שמבנה הפרמטרים במחסנית משתנה, ה-idiv_mod32 צריכה לבצע אריטמיקה של 32 ביט.

```

/* 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("\nEnter 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 */

```

```

E:\>tcc call_id2.c idiv_mo6.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
call_id2.c:
idiv_mo6.asm:
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland
International

Assembling file:    idiv_mo6.ASM
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   431k

Turbo Link Version 5.0 Copyright (c) 1992 Borland International

    Available memory 4150128

```

```

E:\>CALL_ID2.EXE
Enter Numerator, Denominator
:-700000 66666

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

E:\>

```

```

; idiv_mo6.asm - Assembler implementation of
;                  C-callable function idiv_mod32.
;

.MODEL SMALL
.CODE
.386
; Implementation of C callable function ...
; ... int idiv_mod32(long int Num, long int Denom,
;                  [BP+4]          [BP+8]
; long int *Q,      long int *Rem)
; [BP+12]          [BP+14]
; Compute Q := |_ Num / Denom _| , Rem := MOD(Num, Denom)
; function idiv_mod32 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
PUSH SI          ; Preserve register variables
PUSH DI          ;

;
MOV ESI,[BP+8]   ; ESI := Denom
CMP ESI,0        ; Denom = 0 ?
JNE Cont        ; No, continue regular operation
                ; Yes, Denom = 0
MOV AX,0         ; Return value := 0
JMP Done         ; Skip following code
Cont:           ; Denom <> 0
MOV EAX,[BP+4]   ; EAX := Num
CDQ              ; EDX:EAX := EAX
IDIV ESI         ; EAX := EDX:EAX / ESI, EDX := MOD(EAX,ESI)
MOV DI,[BP+12]   ; DI := Offset Q
MOV [DI],EAX     ; *Q := AX
MOV DI,[BP+14]   ; DI := Offset Rem
MOV [DI],EDX     ; *Rem := DX
MOV AX,1        ; Ensure return value := 1
Done:
;
POP DI          ; Restore register variables
POP SI          ;
POP BP          ; Restore BP register
RET
_idiv_mod32 ENDP
END

```