

Irvine eljárások

Ahogy azt már év elején is említettük, a 32 bites környezet lehetővé tette számunkra, hogy több, már előre létrehozott eljárást használhassunk. Ehhez csupán telepítenünk kellett az Irvine függvénycsomagot, majd a programkódunk elején importálni a megfelelő könyvtárat. Ez jelen esetünkben az *Irvine32.inc*. Tehát az alábbi sort kell elhelyeznünk a program elején:

```
INCLUDE Irvine32.inc
```

Az adott függvény meghívásához a *CALL* parancsot kell használni. Röviden bemutatnánk néhány főbb eljárást és azok használatát, egy-egy példával, a teljesség igénye nélkül.

ClrScr - Törli a képernyő tartalmát és a kurzort a képernyő bal felső sarkába helyezi.

```
call ClrScr
```

CrLf - A következő sor elejére helyezi a kurzort (kocsi vissza + új sor).

```
call CrLf
```

ReadChar - Beolvas egy karaktert a standard outputról az AL regiszterbe. A karakter nem kerül kiírásra a képernyőre.

```
.data
```

```
char BYTE    ?
```

```
.code
```

```
    call ReadChar
```

```
    mov  char, al
```

ReadInt - Beolvas egy 32 bites integer értéket a standard outputról az EAX regiszterbe. Az opcionális jel után csak számjegyek szerepelhetnek.

```
.data
```

```
intVal  DWORD    ?
```

```
.code
```

```
    call ReadInt
```

```
    mov  intVal, eax
```

ReadString - Beolvas egy karaktorsorozatot a standard bemenetről, az Enter billentyű leütéséig. A beolvasott karakterek számát az EAX regiszterbe tárolja le, magát a sztringnek az eltolását (offset) pedig mindig az EAX regiszternek kell tartalmaznia. Az ECX regiszterben kell megadni a maximálisan beolvasható karakterek számát.

```
.data
buffer          BYTE          50 DUP(?)
byteCount       DWORD        ?

.code

    mov  edx, OFFSET buffer ; a sztring pointere
    mov  ecx, (SIZEOF buffer)-1 ; Max hossz
    call ReadString          ; beolvasás
    mov  byteCount, eax      ; karakterek számának lementése
```

WriteDec - Kiír egy 32 bites, előjelnélküli egész értéket az standard kimenetre az EAX regiszterből.

```
mov  eax, 2957FFFFh
call WriteDec ; "295" fog megjelenni a képernyőn
```

WriteInt - Kiír egy 32 bites, előjeles egész értéket a standard kimenetre az EAX regiszterből.

```
mov  eax, 216543
call WriteInt ; "+216543" fog megjelenni a képernyőn
```

WriteString - Kiír egy sztringet a standard kimenetre. A sztring eltolás értékének az EDX regiszterben kell lennie.

```
.data
prompt         BYTE          "Add meg a neved: ", 0

.code

    mov  edx, OFFSET prompt
    call WriteString
```

GoToXY - A kurzort a megadott oszlop és sor pozícióra mozgatja. Az oszlop szám (X koordináta) a DL regiszter tartalma, a sor szám (Y koordináta) pedig a DH regiszter tartalma lesz.

```
mov dh, 10      ; 10. sor
mov dl, 20      ; 20. oszlop
call GoToXY     ; A kurzos átmozgatása
```

A további eljárásokról részletes leírás található az alábbi címen, az *Irvine Library* fül alatt:

<http://programming.msjc.edu/asm/help/index.html?page=source%2Fabout.htm>

Invoke direktíva

Az **INVOKE** direktíva csak a 32 bites módban érhető el. Argumentumokat helyez el a veremben, majd meghív egy eljárást. Az INVOKE egy kényelmes helyettesítése a CALL utasításnak, mivel lehetőséget ad több argumentum elhelyezésére egyetlen programkód soron belül. Az alap szintaxis:

```
INVOKE eljárásNév [, paraméterLista]
```

A *paraméterLista* egy opcionális, vesszővel elválasztott listája az argumentumoknak, melyeket átadunk az eljárásnak.

PROC direktíva: A PROC direktíva használatával hozhatunk létre saját eljárásokat.

```
név PROC [attribútumok] [USES regiszterLista], paraméterLista  
...  
név ENDP
```

A megadható attribútumokkal most nem foglalkozunk.

USES direktíva: Az adott eljárás fejlécében a USES direktíva használatával lehetőségünk van megadni regiszterek egy listáját (vessző nélkül, egymás után felsorolva), melyek az eljárás meghívásakor lementésre kerülnek a verembe, végrehajtódik az eljárás, majd visszatéréskor a veremből visszaállításra kerülnek. Természetesen azon regiszter(ek) lementése nem szükséges, amely(ek) eredményt ad(nak) vissza.

A paraméterlistában megadhatunk tetszőleges számú és típusú paramétert vesszővel elválasztva, melyeknek az eljárás hívásakor van lehetőségünk értéket adni és az eljárás futásakor felhasználni. Az alap szintaxis:

```
..., elsőParamNév:típus1, másodikParamNév:típus2, ...
```

PROTO direktíva: A PROTO direktíva készíti el egy létező eljárás prototípusát. A prototípus deklarálja az eljárás nevét és paraméterlistáját. Lehetővé teszi, hogy az adott eljárást a definiálása előtt meghívjuk és leellenőrizzük a paramétereinek nevét és típusát, hogy megegyeznek-e az eljárás definíciójával. Meglévő eljáráshoz prototípust készíteni úgy lehet a legegyszerűbben, hogy lemásoljuk az eljárás fejlécét és a PROC kulcsszót lecseréljük a PROTO kulcsszóra, valamint ha használtuk a USES direktívát, akkor az csak az eljárás fejlécében szerepelhet, a prototípusban nem.

Egy egyszerű példa, melyben egy összeadó eljárást valósítottunk meg:

```
INCLUDE Irvine32.inc
.data
    A_val DWORD 10
    B_val DWORD 5
.code
main PROC
    ; NINCS SZÜKSÉG PROTO-ra HA AZ ELJÁRÁS A HÍVÁSA ELŐTT VAN MEGVALÓSÍTVA!
    Osszeg PROTO, param1:DWORD, param2:DWORD ; Az "Osszeg" nevű eljárás
prototípusa
    MOV EAX, 123 ; beletesszük az 123 értéket, hogy az eljáráshívás után
                ; leellenőrizzük, hogy valóban kimentettük-e az értékét
    INVOKE Osszeg, A_val, B_val ; Az "Osszeg" nevű eljárás meghívása.
                                ; Két paramétere van: az összeadandó két szám
    Call WriteInt ; ellenőrző kiíratás
    Call Crlf ; újsor beszúrása
    INVOKE ExitProcess,0
main ENDP

; Egy összeadó eljárás, melynek két bemeneti paramétere van: az összeadandó két
szám
; Ezeket a paramétereket jelöltük param1 és param2 nevekkkel, és típusuk DWORD
; USES direktíva használatával az EAX regisztert menjük ki hívás után
Osszeg PROC USES EAX, param1:DWORD, param2:DWORD
    MOV EAX, param1 ; az első paraméter regiszterbe helyezése
    ADD EAX, param2 ; a második paraméter hozzáadása
    Call WriteInt ; az eredmény kiíratása
    Call Crlf ; újsor beszúrása
    ret ; visszatérés az eljáráshíváshoz
Osszeg ENDP
end main
```

Lehetőségünk van assembly nyelven írt kódot (eljárást) meghívni C/C++ kódon belül. Pl.:

A C++ kód, melyben egy *addem* nevű eljárást hívunk, mely az assembly kódon belül van megvalósítva:

```
// Addem Main Program (AddMain.cpp)

#include <iostream>
using namespace std;

extern "C" int addem(int p1, int p2, int p3);

int main(){
    int total = addem( 10, 15, 25 );
    cout << "Total = " << total << endl;
    return 0;
}
```

Az assembly kód, melyben egy összeadó eljárás van megvalósítva:

```

; The addem Subroutine    (addem.asm)
; This subroutine links to Visual C++.

.386P
.model flat
public _addem

.code
_addem proc near
    push    ebp
    mov     ebp,esp
    mov     eax,[ebp+16]    ; first argument
    add     eax,[ebp+12]    ; second argument
    add     eax,[ebp+8]     ; third argument
    pop     ebp
    ret
_addem endp
end

```

Mikor a C++ kódból meghívjuk az eljárást, a bementi paraméterek bele kerülnek a verembe (plusz a visszatérési cím). Így az eljárás futásakor már benne vannak a veremben, csak megfelelő címmel el kell érni őket. A verem tartalma:

25	ebp+16
15	ebp+12
10	ebp+8
ret	ebp+4
ebp	

További példák az INVOKE direktíva használatára. MessageBox megjelenítése:

```

;---ASM Hello World Win32 MessageBox
.386
.model flat, stdcall

include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc

.data
Titlestr db 'Win32', 0
Msg db 'Hello World', 0

.code
Main:
INVOKE MessageBoxA, 0, offset titlestr, offset msg, 0
INVOKE ExitProcess, eax
End Main

```

Egy kis érdekesség: A következő program (ShowTime.asm) lekérdezi az aktuális időpontot, majd megjeleníti azt az általunk választott képernyő pozíción.

```

INCLUDE Irvine32.inc
.data
    sysTime SYSTEMTIME <>
    XYPos COORD <10,5>
    consoleHandle DWORD ?
    colonStr BYTE ":",0
.code
main PROC
    ; Get the standard output handle for the Win32 Console.
    INVOKE GetStdHandle, STD_OUTPUT_HANDLE
    mov consoleHandle,eax

    ; Set the cursor position and get the system time.
    INVOKE SetConsoleCursorPosition, consoleHandle, XYPos
    INVOKE GetLocalTime, ADDR sysTime

    ; Display the system time (hh:mm:ss).
    movzx eax,sysTime.wHour ; hours
    call WriteDec
    mov edx,OFFSET colonStr ; ":"
    call WriteString
    movzx eax,sysTime.wMinute ; minutes
    call WriteDec
    call WriteString
    movzx eax,sysTime.wSecond ; seconds
    call WriteDec
    call Crlf
    exit
main ENDP
END main

```

Feladatok

1. Írj egy eljárást, melynek legyen két DWORD típusú bemeneti paramétere. Az eljárásan belül pedig kerüljön kiszámításra a két paraméter minimuma (maximuma). Az eljárást az INVOKE direktíva segítségével hívd meg. Használd a USES direktívát az eljárásan belül felhasznált regiszterek kimentésére.
2. Írj egy eljárást, melynek legyen két bemeneti paramétere. Az egyik típusa DWORD legyen, mely egy karaktersorozat eltolása, a másik típusa BYTE legyen, amely pedig a karaktersorozatban keresendő karakter legyen. Az eljárás tehát a paraméterben kapott karaktersorozatban számolja meg hányszor szerepel az úgyszintén paraméterként kapott karakter. Az eljárást az INVOKE direktíva segítségével hívd meg.