13. *Dezvoltarea aplicatiilor in limbaj C si asamblare*

13.1. Execuţia programelor pe 32 biţi

O modalitate de a dezvolta programe pe 32 biţi este prin folosirea Visual C++ în care se poate insera cod în limbaj de asamblare inline. Astfel, se beneficiază de toate avantajele folosirii unui limbaj de programare de nivel înalt. În plus, se pot consulta regiştrii pe 32 biţi, aşa cum era posibil în limbajul de nivel scăzut.

Termenul *inline* este un cuvânt cheie în limbajul C și este folosit în declararea funcțiilor. Atunci când în programul sursă compilatorul găsește o funcție declarată ca fiind „inline”, peste tot în programul principal unde va găsi apel al acelei funcţii, va înlocui cu corpul funcției respective. Expresia “inline assembly” se referă la un set de instrucțiuni în limbaj de asamblare, scrise ca funcții inline. ***Inline assembly*** este folosit în general ca o metodă de optimizare utilizată în dezvoltarea programelor.

În programele C/C++ se pot insera instrucțiuni în limbaj de asamblare folosind cuvântul cheie “asm”, precedat de „_” şi apoi între { } se vor scrie respectivele instrucţiuni în asamblare.

13.2. Etape în crearea unui program folosind asamblarea inline

Pas1. Se creează un proiect de tip Win32 Console Application (în exemplele prezentate s-a folosit Visual Studio Express Edition), în care suntem de acord cu toate opţiunile propuse şi selectăm *Finish*, aşa cum se poate observa şi din figura 13.1.

Exemplu de program simplu, tipic în C, care calculează suma a 2 numere preluate de la tastatură:

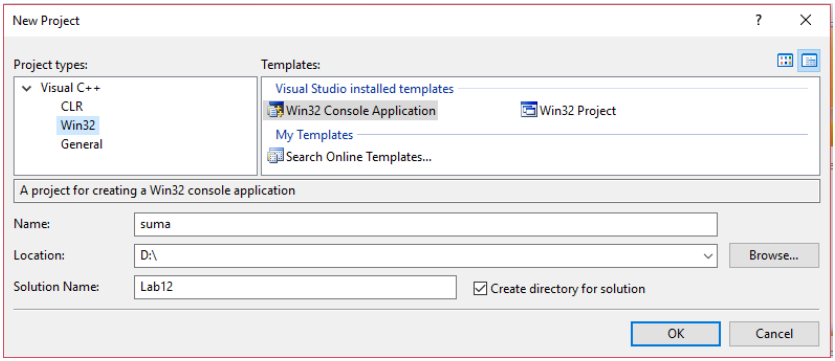

Fig 13.1. Crearea unui program în Visual Studio

Pas 2. Scriem programul de mai jos (în locul şablonului oferit automat de Microsoft Visual Studio C++) şi dăm apoi comanda *Build Solution* sau *F7*, apoi *Build suma*, *Compile*

```
#include "stdafx.h"
#include "stdio.h"
int a, b, sum;      // variabile globale
int main (void)
{
 printf ("Enter the first number: ");
 scanf ("%d", &a);  // preluarea primului numar

 printf ("Enter the second number: ");
 scanf ("%d", &b);  // preluarea celui de-al doilea numar
 // sum = a+b;      // calculeaza suma
 _asm {         MOV eax, a
                MOV ebx, b
                ADD eax, ebx
                MOV sum, eax
       }
 printf ("\n%d plus %d = %d\n", a, b, sum);
 return 0;
}
```

Pas 3. Executăm programul linie cu linie, folosind opţiunea *Step Into*:

Se va selecta *Start Debugging (F5)* şi apoi *Step Into (F11)*; în continuare, la opţiunile *Windows* (din *Debug*) vor apărea mult mai multe opţiuni. Vom parcurge întreg programul cu *F11 (Step Into)* apăsat, pentru fiecare linie din fişier, aşa cum se poate propune în figura 13.2:

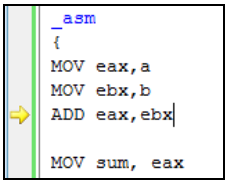

Fig 13.2. Depanare cu Visual Studio: programul propus înainte de execuţia operaţiei de adunare

Regiştrii arată ca în figura 13.3 (după execuţia MOV ebx,b); în exemplul prezentat, utilizatorul a apăsat 5, urmat de Enter şi apoi 11 urmat de Enter; aşa cum se poate urmări şi în codul sursă, valorile sunt preluate ca numere în zecimal.

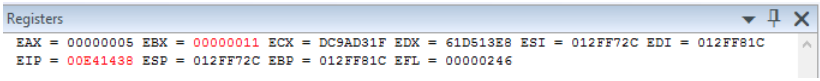

Fig 13.3. Depanare cu Visual Studio: vizualizare regiştri

Mai apăsăm încă o dată ***F11***, şi deci se va executa operaţia de adunare, care va avea ca rezultat coborârea cursorului astfel încât să indice că operaţia următoare este *MOV suma, eax* (aşa cum se prezintă în figura 13.4), iar valorile regiştrilor sunt cele din figura 13.5, unde se poate observa rezultatul în registrul EAX. Subliniem faptul că EFL este registrul de flaguri (Extended FLag).

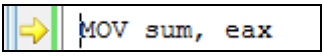

Fig 13.4. Depanare cu Visual Studio: modul cum se mută cursorul pe instrucţiunea curentă

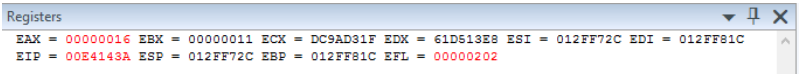

Fig 13.5. Depanare cu Visual Studio: modul cum se prezintă regiştrii după execuţia instrucţiunii *MOV sum, eax*

Pas 4. La sfârşit, când dorim să ne oprim cu depanarea, selectăm ***Stop Debugging.***

Dacă vom selecta opţiunea ***Disassembly*** (disponibilă în meniul *Debug-> Windows* după ce am dat cel puţin o dată pe *F11*, sau folosind *Alt+8*), vom vedea că inclusiv funcţiile din C (precum *printf* şi *scanf*) sunt transformate în codul echivalent în asamblare; important însă pentru noi să urmărim este faptul că tipul variabilelor folosite în program, adică *a,b* şi *sum* sunt precedate acum de construcţia *dword ptr* şi variabila apare între [ ], aşa cum arată figura 13.6.

```
_asm
{
MOV eax,a
00151432  mov         eax,dword ptr [a]
MOV ebx,b
00151435  mov         ebx,dword ptr [b]
ADD eax,ebx
00151438  add         eax,ebx

MOV sum, eax
0015143A  mov         dword ptr [sum],eax
```

Fig 13.6. Depanare cu Visual Studio: adaptarea la tipul variabilelor folosind *dword ptr*

13.3. Când se foloseste inline assembly?

În general, atunci când limbajul de nivel înalt nu are acces la anumite instrucţiuni sau facilităţi (dar care din limbaj de asamblare sunt uşor de obţinut)

1. În limbajul C există suport pentru **operații de deplasare (shiftare) pe biți,** dar nu există implementat suport pentru operații de rotație pe biți (deşi aceste operaţii există la nivelul procesorului).

2. **CPUID**: La nivelul procesoarelor moderne există o instrucțiune simplă, accesibilă doar din limbaj de asamblare, care oferă informații despre procesor; această instrucţiune este cpuid.

Operaţii pe biţi în C: Datele aflate în memoria RAM a sistemului sunt organizate ca o secvenţă de octeţi. Operatorii la nivel de bit sunt necesari atunci când dorim să prelucrăm biţii din interiorul structurii unui octet.

Limbajul C suportă 6 tipuri de operatori pe biţi („bitwise operators").
- **operatorul &** pentru realizarea operaţiei AND logic pe biţi
- **operatorul |** pentru realizarea operaţiei OR logic pe biţi
- **operatorul ^** pentru realizarea operaţiei XOR logic pe biţi
- **operatorul ~** pt realizarea operaţiei NOT logic pe biţi (complement faţă de 1)
- **operatorul <<** de deplasare spre stânga
- **operatorul >>** de deplasare spre dreapta

Exemplul 1. Deplasarea spre stânga a unui număr pe 32 biţi (citit de la tastatură în hexazecimal) cu un nr de poziţii citit de la tastatură (ca nr întreg între 0-31).

```
Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): 1234
Introduceti un nr < 32 ca fiind nr de pozitii cu cat doriti deplasare spre stanga: 8

Valoarea lui n este: 00001234h inainte de deplasarea spre stanga cu 8 pozitii.
Valoarea lui n este: 00123400h dupa deplasarea spre stanga cu 8 pozitii.
```

Fig 13.7. Fereastra execuţiei ex. 1

Program scris în C	Program în C cu inline asm
`/* Program in C - demonstrarea folosirii operatorului de deplasare spre stanga << in C.*/`	`/* Program in C cu asm inline - demonstrarea folosirii instructiunii de deplasare spre stanga SHL in ASM.*/`
`#include <stdafx.h>`	`#include <stdafx.h>`
`#include <stdio.h>`	`#include <stdio.h>`
`int main()`	`int main()`
`{`	`{`
`unsigned int n, poz;`	`unsigned int n, poz;`

```
  printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
  scanf("%X", &n);
  printf("Introduceti un nr < 32 ca fiind nr de pozitii cu cat doriti deplasare spre stanga: ");
  scanf("%d", &poz);
printf("\nValoarea lui n este: %08Xh inainte de deplasarea spre stanga cu %d pozitii.",n,poz);
    /*deplasarea lui n cu poz pozitii spre stanga*/
    n = (n<<poz); /*operatia*/

printf("\nValoarea lui n este: %08Xh dupa deplasarea spre stanga cu %d pozitii. \n\n",n,poz);
    return 0;
}
```

```
    printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
    scanf("%X", &n);
    printf("Introduceti un nr < 32 ca fiind nr de pozitii cu cat doriti deplasare spre stanga: ");
    scanf("%d", &poz);
   printf("\nValoarea lui n este: %08Xh inainte de deplasarea spre stanga cu %d pozitii.",n,poz);
    /*deplasarea lui n cu poz pozitii spre stanga*/
    _asm          /*operatia*/
        {       MOV EAX, n;
                MOV ECX, poz;
                SHL EAX, CL
                MOV n, EAX;
        }
printf("\nValoarea lui n este: %08Xh dupa deplasarea spre stanga cu %d pozitii. \n\n",n,poz);
    return 0;
}
```

Exerciţii propuse:

1. Realizaţi deplasarea spre dreapta (echivalent cu instrucţiunea ***shr***).
2. Propuneţi o metodă/ un algoritm de a implementa în C acelaşi efect ca şi cel obţinut prin execuţia instrucţiunii ***sar***.

Exemplul 2. Setarea/resetarea unui anumit bit specificat prin poziţie (poziţia bitului se preia de la tastatură)

```
Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): 12345678
Introduceti un nr < 32 ca fiind pozitia de pe care vreti sa setati bitul: 7

Valoarea lui n este: 12345678h inainte de setarea bitului de pe pozitia 7.
Valoarea lui n este: 123456F8h dupa setarea bitului de pe pozitia 7.

Introduceti un nr < 32 ca fiind pozitia de pe care vreti sa resetati bitul: 9

Valoarea lui n este: 123456F8h inainte de resetarea bitului de pe pozitia 9.
Valoarea lui n este: 123454F8h dupa resetarea bitului de pe pozitia 9.
```

Fig 13.8. Fereastra execuţiei ex. 2

Program scris în C

```
/* C Program to set/clear some bits.*/
#include <stdafx.h>
#include <stdio.h>
int main()
{     unsigned int n, poz;
printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
scanf("%X", &n);
printf("Introduceti un nr < 32 ca fiind pozitia de pe care vreti sa setati bitul: ");
scanf("%d", &poz);
printf("\nValoarea lui n este: %08Xh inainte de setarea bitului de pe pozitia %d.",n,poz);
   /*seteaza bitul de pe pozitia poz*/
    n |= (1 << poz);

// sau : newN = (1 << poz) | n;

printf("\nValoarea lui n este: %08Xh dupa setarea bitului de pe pozitia %d.\n\n",n,poz);
printf("Introduceti un nr < 32 ca fiind pozitia de pe care vreti sa resetati bitul: ");
scanf("%d", &poz);
printf("\nValoarea lui n este: %08Xh inainte de resetarea bitului de pe pozitia %d.",n,poz);
   /*reseteaza bitul de pe pozitia poz*/
    n &= ~(1 << poz);

printf("\nValoarea lui n este: %08Xh dupa resetarea bitului de pe pozitia %d.\n\n",n,poz);
    return 0;
}
```

Program în C cu inline asm

```
/* C Program to set/clear some bits.*/
#include <stdafx.h>
#include <stdio.h>
int main()
{     unsigned int n, poz;
printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
scanf("%X", &n);
printf("Introduceti un nr < 32 ca fiind pozitia de pe care vreti sa setati bitul: ");
scanf("%d", &poz);
printf("\nValoarea lui n este: %08Xh inainte de setarea bitului de pe pozitia %d.",n,poz);
   /*seteaza bitul de pe pozitia poz*/
    _asm {      MOV EAX, n
                MOV EBX, 1
                MOV ECX, poz
                SHL EBX,CL
                OR EAX,EBX
                MOV n,EAX
          }
printf("\nValoarea lui n este: %08Xh dupa setarea bitului de pe pozitia %d.\n\n",n,poz);
printf("Introduceti un nr < 32 ca fiind pozitia de pe care vreti sa resetati bitul: ");
scanf("%d", &poz);
printf("\nValoarea lui n este: %08Xh inainte de resetarea bitului de pe pozitia %d.",n,poz);
   /*reseteaza bitul de pe pozitia poz*/
_asm    {       MOV EAX, n
                MOV EBX, 1
                MOV ECX, poz
                SHL EBX,CL
                NOT EBX
                AND EAX,EBX
                MOV n,EAX
        }
 printf("\nValoarea lui n este: %08Xh dupa resetarea bitului de pe pozitia %d.\n\n",n,poz);
    return 0;
}
```

Exerciţii propuse:

1. Modificaţi programele astfel încât să se preia de la tastatură poziţia unui bit şi să se seteze/reseteze un număr de 2/3/... biţi pornind de la acea poziţie înspre stânga / dreapta.
2. Să se preia de la tastatură poziţia şi un număr de biţi, de ex. se preia 3 şi 4: de la bitul de pe poziţia 3 vor fi afectaţi 4 biţi.
3. Repetaţi exerciţiile anterioare, dar bitul va fi modificat – adică inversat : newN = n ^ (1 << poz);
4. Sa transforme un caracter din minuscula in majuscula sau invers (se modifica doar bitul 6) – caracterele sunt litere !!! (Bit masking)

Exemplul 3. Program care să numere numărul de biţi de 1 dintr-un număr:

```
Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): 12345

Numarul total de biti de 1 este: 7
```

Fig 13.9. Fereastra execuţiei ex. 3

Program scris în C

```
/*C program to count number of 1's in a number */
#include <stdafx.h>
#include <stdio.h>
int main()
{   unsigned int n;
    unsigned                    char totalBits=sizeof(n)*8;
    int i,count=0;
printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
scanf("%X", &n);
for(i=0;i< totalBits;i++)
    {
        if( n & (1<< i) )
            count++;     //     se parcurge de 32 ori: totalBits=32

    }

    printf("\nNumarul total de biti de 1 este: %d\n",count);
     return 0;
}
```

Program în C cu inline asm

```
/*C program to count number of 1's in a number */
#include <stdafx.h>
#include <stdio.h>
int main()
{   unsigned int n;
    unsigned                    char totalBits=sizeof(n)*8;
    int i,count=0;
printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
scanf("%X", &n);
_asm
    {
      MOV EAX, n
      POPCNT EBX, EAX
      MOV count, EBX
    }
    printf("\nNumarul total de biti de 1 este: %d\n",count);
     return 0;
}
```

Exemplul 4. Inversarea biţilor unui număr.

```
Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): 1234

Numarul inversat este: 2C480000
```

Fig 13.10. Fereastra execuţiei ex. 4

Program scris in C

```
/*C program to reverse bits in
a number */
#include <stdafx.h>
#include <stdio.h>

    int main()
    {   unsigned int n;
        unsigned           char
totalBits = sizeof(n) * 8;
        unsigned           int
reversedNum = 0, i, temp;
printf("Introduceti un nr cu
maxim     8     cifre     hexa
(introduceti doar cifrele lui
hexa): ");
scanf("%X", &n);
for (i = 0; i < totalBits;
i++)
        {
        temp = (n & (1 << i));
         if(temp)
         reversedNum |= (1 <<
((totalBits - 1) - i));
        }




    printf("\nNumarul inversat
este: %X\n", reversedNum);
    return 0;
}
```

Program in C cu inline asm

```
/*C program to reverse bits in
a number */
#include <stdafx.h>
#include <stdio.h>

    int main()
    {   unsigned int n;
        unsigned           char
totalBits = sizeof(n) * 8;
        unsigned           int
reversedNum = 0, i, temp;
printf("Introduceti un nr cu
maxim     8     cifre     hexa
(introduceti doar cifrele lui
hexa): ");
scanf("%X", &n);
_asm
{       MOV ECX,32
        MOV EAX,n
        MOV EBX,0
eti:
        ROL EAX,1
        RCR EBX,1
        loop eti
        MOV reversedNum, EBX
}
    printf("\nNumarul inversat
este: %X\n", reversedNum);
    return 0;
}
```

Exemplul 5. Interschimbarea ultimelor 2 cifre (nibble) ale unui număr.

```
Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): 123456

Numarul introdus de dvs : 00123456
Numarul dupa interschimbarea ultimelor 2 tetrade: 00123465
```

Fig 13.11. Fereastra execuţiei ex. 5

Program scris in C

```
/*C program to swap two nibbles of a given byte*/
#include <stdafx.h>
#include <stdio.h>
int main()
{    unsigned int n;
    unsigned int num;
  printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
scanf("%X", &n);


num= ( (n & 0xFFFFFF00)<<0 |(n & 0x0F)<<4 | (n & 0xF0)>>4 );



printf("\nNumarul introdus de dvs : %08X \nNumarul dupa interschimbarea ultimelor 2 tetrade: %08X \n \n ",n,num);
    return 0;
}
```

Program in C cu inline asm

```
/*C program to swap two nibbles of a given byte*/
#include <stdafx.h>
#include <stdio.h>
int main()
{    unsigned int n;
     unsigned int num;
  printf("Introduceti un nr cu maxim 8 cifre hexa (introduceti doar cifrele lui hexa): ");
scanf("%X", &n);
 _asm
{
        MOV EAX, n
        ROL AL,4
        MOV num, EAX
}


 printf("\nNumarul introdus de dvs : %08X \nNumarul dupa interschimbarea ultimelor 2 tetrade: %08X \n \n ",n,num);
    return 0;
}
```

Exercitii propuse. Analizaţi ce se întâmplă dacă se înlocuieşte cu:
(**(n & 0xFFFF0000) <<0 |** (n<<8) &0xFF00) |((n>>8) &0x00FF).