Laborator APC
Instrucțiuni MMX

Implementarea instrucțiunilor Intel extinse este facilitată de utilizarea “macro-instrucțiunilor” special concepute pentru a fi apelate din limbajul C, astfel încât programatorul este eliberat de sarcina scrierii întregului cod în limbaj de asamblare.

Documentația acestor macro-instrucțiuni se poate consulta la adresa:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/,

pagina punând la dispoziție toate generațiile de instrucțiuni SIMD suplimentare pentru microprocesoarele Intel cu posibilitatea de a selecta categorii specifice de instrucțiuni (artimetice, logice, de transfer etc.).

I. Implementarea produsului scalar a doi vectori ce conțin numere pare, respectiv impare în limbaj C fără utilizarea macro-instrucțiunilor:

```
#include <stdio.h>
#include <malloc.h>

#define NMAX 2000

int main(int argc,char **argv)
{
   short a[NMAX],b[NMAX];
   long S;
   int i;
   //alocare memorie pentru vectorii a si b
   a = (short*)malloc(NMAX*sizeof(short));
   b = (short*)malloc(NMAX*sizeof(short));
   //initializare vectori a si b
   for(i=0;i<NMAX;i++)
   {
         a[i]=2*i;          //numere pare
         b[i]=2*i+1;        //numere impare
   }
```

```
   //produsul scalar al vectorilor a si b
   S=0;
   for(i=0;i<NMAX;i++)
   {
         S+=a[i]*b[i];
   }
   printf("Produs scalar 1 = %ld\n",S);
}
```

Codul C se salvează într-un fișier cu extensia .c: “prodsc1.c”. Programul executabil se obține prin compilarea fișierului cu ajutorul comenzii:

> gcc -o prodsc1 prodsc1.c

În directorul de lucru, în urma compilării cu succes este creat fișierul “prodsc1”, care va fi rulat cu comanda:

> ./prodsc1

Rezultatul pentru un număr de 20.000 elemente este:

> Produs scalar 1 = 10662666000

Se poate vedea codul generat în limbaj de asamblare prin utilizarea opțiunii -S:

> gcc -S prodsc1.c

Rularea acestei comenzi va produce un fișier text cu extensia .s: “prodsc1.s”, al cărui conținut se poate vizualiza cu orice editor. Se pot remarca instrucțiunile utilizate pentru implementarea calculului, care nu conține instrucțiuni MMX.

```
*       *       *       *       *       *       *       *       *       *       *       *       *       *
      .file   "test.c"
      .section        .rodata
.LC0:
      .string "Produs scalar 1 = %ld\n"
      .text
      .globl  main
      .type   main, @function
main:
.LFB129:
      .cfi_startproc
      pushq   %rbp
      .cfi_def_cfa_offset 16
      .cfi_offset 6, -16
      movq    %rsp, %rbp
      .cfi_def_cfa_register 6
      subq    $16048, %rsp
      movl    %edi, -16036(%rbp)
      movq    %rsi, -16048(%rbp)
```

```
        movl    $4000, %edi
        call    malloc
        movq    %rax, -24(%rbp)
        movl    $4000, %edi
        call    malloc
        movq    %rax, -32(%rbp)
        movl    $0, -12(%rbp)
        jmp     .L2
.L3:
        movl    -12(%rbp), %eax
        cltq
        leaq    (%rax,%rax), %rdx
            .       .       .       .       .       .       .       .       .

            .       .       .       .       .       .       .       .       .
*       *       *       *       *       *       *       *       *       *       *       *       *       *
```

Listing parțial al fișierului “prodsc1.s”

II. Implementarea produsului scalar a doi vectori ce conțin numere pare, respectiv impare în limbaj C cu utilizarea macro-instrucțiunilor “*Intel intrinsics*”:

```
#include <stdio.h>
#include <mmintrin.h>
#include <malloc.h>

#define NMAX 2000

int main(int argc,char **argv)
{
   //variabile speciale de tipul __m64 pentru incarcarea in registrii MMX
   __m64 mmxa1,mmxa2,mmxb1,mmxb2,mmxs,mmxprod1,mmxprod2;
   short *a,*b;
   long s[NMAX],S;
   int i;

   a = (short*)malloc(NMAX*sizeof(short));
   b = (short*)malloc(NMAX*sizeof(short));

   for(i=0;i<NMAX;i++)
   {
         a[i]=2*i;          //numere pare
         b[i]=2*i+1;        //numere impare
   }
   //suma vectorilor a si b
   S=0;
   for(i=0;i<=NMAX-8;i+=8)                                  //se vor executa de 8 ori mai puține iterații
   {
         mmxa1 = _mm_set_pi16(a[i+3],a[i+2],a[i+1],a[i]);       //incarcarea variabilei de 64 biti mmxa1
```

```
        mmxb1 = _mm_set_pi16(b[i+3],b[i+2],b[i+1],b[i]);       //incarcarea variabilei de 64 biti mmxb1

        mmxa2 = _mm_set_pi16(a[i+7],a[i+6],a[i+5],a[i+4]);    //incarcarea variabilei de 64 biti mmxa2
        mmxb2 = _mm_set_pi16(b[i+7],b[i+6],b[i+5],b[i+4]);   //incarcarea variabilei de 64 biti mmxb2

        mmxprod1 = _m_pmaddwd(mmxa1,mmxb1);    //calculul produs+suma dintre mmxa1 si mmxb1
        mmxprod2 = _m_pmaddwd(mxa2,mmxb2);     //calculul produs+suma dintre mmxa2 si mmxb2

        //calculul sumei dintre rezultatele partiale mmxprod1 si mmxprod2
        mmxs = _mm_add_pi32(mmxprod1,mmxprod2);
        //suma finala obtinuta ca suma celor doua sume partiale din variabila 'mmxs'
        S += _mm_cvtsi64_si32(mmxs);
        S += _mm_cvtsi64_si32(_mm_srli_si64(mmxs,32));
    }
    printf("Produs scalar 2 = %ld\n",S);
    free(a);
    free(b);

    return 0;
}
```

Codul C prezentat anterior se salvează într-un fișier cu numele “prodsc2.c”, urmând a fi compilat și executat la fel ca “prodsc1.c”.

Se poate consulta fișierul rezultat în urma compilării cu opțiunea de asamblare (-S), remarcându-se utilizarea instrucțiunilor MMX:

```
*       *       *       *       *       *       *       *       *       *       *       *       *       *
.       .       .       .       .       .       .       .
        movq    %rax, -64(%rbp)
        movq    -40(%rbp), %rax
        movq    %rax, -128(%rbp)
        movq    -48(%rbp), %rax
        movq    %rax, -136(%rbp)
        movq    -128(%rbp), %rax
        movq    %rax, -144(%rbp)
        movq    -136(%rbp), %rax
        movq    %rax, -152(%rbp)
        movq    -152(%rbp), %mm0
        movq    -144(%rbp), %mm1
        pmaddwd         %mm1, %mm0
        movq    %mm0, -16264(%rbp)
        movq    -16264(%rbp), %rax
        movq    %rax, -72(%rbp)
        movq    -56(%rbp), %rax
        movq    %rax, -160(%rbp)
        movq    -64(%rbp), %rax
        movq    %rax, -168(%rbp)
        movq    -160(%rbp), %rax
        movq    %rax, -176(%rbp)
```

```
        movq    -168(%rbp), %rax
        movq    %rax, -184(%rbp)
        movq    -184(%rbp), %mm0
        movq    -176(%rbp), %mm1
        pmaddwd         %mm1, %mm0
        movq    %mm0, -16264(%rbp)
        movq    -16264(%rbp), %rax
        movq    %rax, -80(%rbp)
        movq    -72(%rbp), %rax
        movq    %rax, -192(%rbp)
        movq    -80(%rbp), %rax
        movq    %rax, -200(%rbp)
        movq    -200(%rbp), %mm0
        movq    -192(%rbp), %mm1
        paddd   %mm1, %mm0
        movq    %mm0, -16264(%rbp)
        movq    -16264(%rbp), %rax
        movq    %rax, -88(%rbp)
        movq    -88(%rbp), %rax
        movq    %rax, -208(%rbp)
        movq    -208(%rbp), %xmm0
        movd    %xmm0, -16264(%rbp)
        movl    -16264(%rbp), %eax
        cltq
.       .       .       .       .       .       .       .       .       .       .       .       .
*       *       *       *       *       *       *       *       *       *       *       *       *
```

Listing parțial al codului rezultat în urma compilării cu opțiunea

de asamblare pentru fișierul “prodsc2”

III. EXERCIȚIU

Utilizând documentația pusă la dispoziție de Intel pe pagina menționată, realizați un program C/macro-instrucțiuni care să implementeze suma a doi vectori:

s[NMAX] = a[NMAX] + b[NMAX]