2° curso / 2° cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

### Arquitectura de Computadores (AC)

Cuaderno de prácticas. Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

**Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo):** Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

Sistema operativo utilizado: Ubuntu 16.04 LTS

**Versión de gcc utilizada:** *gcc version* 5.4.0 20160609 (*Ubuntu* 5.4.0-6*ubuntu*1~16.04.9)

Volcado de pantalla que muestre lo que devuelve 1scpu en la máquina en la que ha tomado las medidas

```
Arquitectura: x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes: Little Endias
CPU(s):
On-line CPU(s) list:
                                                          0-7
Hilo(s) de procesamiento por núcleo:2
Núcleo(s) por «socket»:4
Socket(s):
Modo(s) NUMA:
ID de fabricante:
                                                           GenuineIntel
 Familia de CPU:
Modelo:
Model name:
Revisión:
                                                          Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
 CPU MHz:
                                                          800.033
 CPU max MHz:
                                                           3500,0000
                                                          800,0000
5184.00
 CPU min MHz:
BogoMIPS:
Virtualización:
                                                        VT-x
 Caché L1d:
                                                        32K
  Caché L1i:
                                                         32K
 Caché L2:
                                                        256K
 Caché L3:
                                                        6144K
NUMA node0 CPU(s):
                                                           0 - 7
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm consta nt_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc _known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand lahf_lm abm 3dnowprefet ch cpuid_fault epb invpcid_single pti retpoline intel_pt rsb_ctxsw tpr_shadow vnmi flexpr iority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed a dx smap clflushopt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify
hwp_act_window hwp_epp
```

- 1. Para el núcleo que se muestra en el Figura 1, y para un programa que implemente la multiplicación de matrices (use variables globales):
  - 1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use -O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.
  - 1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.
  - 1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

Figura 1 . Código C++ que suma dos vectores

```
struct {
    int a;
    int b;
} s[5000];

main()
{
    ...
    for (ii=0; ii<40000;ii++) {
        X1=0; X2=0;
        for(i=0; i<5000;i++) X1+=2*s[i].a+ii;
        for(i=0; i<5000;i++) X2+=3*s[i].b-ii;

        if (X1<X2) R[ii]=X1 else R[ii]=X2;
    }
    ...
}</pre>
```

A) MULTIPLICACIÓN DE MATRICES: CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char **argv)
       unsigned i, j, k;
       if(argc < 2)
             fprintf(stderr, "falta size\n");
exit(-1);
       unsigned int N = atoi(argv[1]); // M\dot{a}ximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
      int **ma, **mb, **mc;
ma = (int **) malloc(N*sizeof(int*));
mb = (int **) malloc(N*sizeof(int*));
mc = (int **) malloc(N*sizeof(int*));
       for (i=0; i<N; i++){
    ma[i] = (int *) malloc(N*sizeof(int));
    mb[i] = (int *) malloc(N*sizeof(int));
    mc[i] = (int *) malloc(N*sizeof(int));</pre>
       }
      // Inicialización

for (i=0; i<N; i++){
    for (j=0; j<N; j++){
        ma[i][j] = 0; //aquí es donde almacenaremos el resultado
        mb[i][j] = 6;
        mc[i][j] = 4;
              }
       struct timespec cgt1,cgt2; double ncgt;
       clock_gettime(CLOCK_REALTIME,&cgt1);
      for (i=0; i<N; i++){
    for (j=0; j<N; j++){
        for (k=0; k<N; k++){
            ma[i][j] += mb[i][k] * mc[k][j];
       clock_gettime(CLOCK_REALTIME,&cgt2);
       ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

```
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

// Pitamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);

// Liberamos la memoria
for (i=0; i<N; i++)
{
    free(ma[i]);
    free(mb[i]);
    free(mc[i]);
}

free(mc);

return 0;

// Company for the foundation of the following prints of the foundation of the
```

Cuaderno de practicas de Arquitectura de Computadores, Grado en Ingenieria Informatica
1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):
Modificación a) –explicación-:
He invertido las variables j y k en el bucle, al estar así los datos más cerca en
memoria
Modificación b) –explicación-:
He desenrollado el bucle en bloques de 8
1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de pmm-secuencial-modificado\_a.c

```
#include <stdlib.l
#include <time.h>
int main(int argc, char **argv)
       unsigned i, j, k;
        if(argc < 2)
              fprintf(stderr, "falta size\n");
exit(-1);
       unsigned int N = atoi(argv[1]); // Máximo N = 2^32 - 1 = 4294967295 (sizeof(unsigned int) = 4 B)
       int **ma, **mb, **mc;
ma = (int **) malloc(N*sizeof(int*));
mb = (int **) malloc(N*sizeof(int*));
mc = (int **) malloc(N*sizeof(int*));
       //reserva de memoria
for (i=0; i<N; i++){
    ma[i] = (int *) malloc(N*sizeof(int));
    mb[i] = (int *) malloc(N*sizeof(int));
    mc[i] = (int *) malloc(N*sizeof(int));
}</pre>
       // Inicialización

for (i=0; i<N; i++){
    for (j=0; j<N; j++){
        ma[i][j] = 0; //aquí es donde almacenaremos el resultado
        mb[i][j] = 6;
        mc[i][j] = 4;
              }
       struct timespec cgt1,cgt2; double ncgt;
       clock_gettime(CLOCK_REALTIME,&cgt1);
       for (i=0; i<N; i++){
    for (k=0; k<N; k++){
        for (j=0; j<N; j|++){
            ma[i][j] += mb[i][k] * mc[k][j];
}</pre>
       clock_gettime(CLOCK_REALTIME,&cgt2);
       ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
       // Pitamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);
        // Pitamos la primera y la ultima linea de la matriz resultante
printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);
         for (i=0; i<N; i++)
               free(ma[i]);
free(mb[i]);
free(mc[i]);
         free(ma);
        free(mb);
free(mc);
         return 0;
```

#### Capturas de pantalla (que muestren la compilación y que el resultado es correcto):

#### b) Captura de pmm-secuencial-modificado b.c

```
#Include <stdio.h>
#Include <stdio.h>
#Include <stdio.h>
#Include <tinc.h>

int matn(int argc, char **argv)

{
    unsigned i, j, k;
    int total = 0;
    int h;
    int sl,s2,s3,s4,s5,s6,s7,s8;
    sl=s2=3=s4=55=s6=57=s8=6;

if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }

unsigned int N = atoi(argv[i]); // Maximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 8)

int **ma, **mb, **mc;
    ma = (int **) malloc(N*sizeof(int*));
    mb = (int **) malloc(N*sizeof(int*));
    mc = (int **) malloc(N*sizeof(int*));
    mc = (int **) malloc(N*sizeof(int*));
    mb[i] = (int *) malloc(N*sizeof(int));
    mb[i] = (int *) malloc(N*sizeof(int));
    mb[i] = (int *) malloc(N*sizeof(int));
    mc[i] = 0; //aqui es donde almacenaremos el resultado
    mb[ii][i] = 0;
    mc[ii][i] = 0;
```

Capturas de pantalla (que muestren la compilación y que el resultado es correcto):

```
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial.c -o pm
guillesiesta@guillesiesta:~$ ./pm 2000
Tiempo = 48.565257990
                         Primera = 48000
                                                 Ultima=48000
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial-modA.c -o pmA
guillesiesta@guillesiesta:~$ ./pmA 2000
Tiempo = 5.317406115
                         Primera = 48000
                                                 Ultima=48000
guillesiesta@guillesiesta:~$ gcc -02 pmm-secuencial-modB.c -o pmB
guillesiesta@guillesiesta:~$ ./pmB 2000
                         Primera = 48000
Tiempo = 4.630081184
                                                 Ultima=48000
guillesiesta@guillesiesta:~$
```

#### **1.1. TIEMPOS:**

Modificación	-O2
Sin modificar	48.565257990
Modificación a)	5.317406115
Modificación b)	4.630081184
•••	

#### 1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Es increíble como con una modificación tan sencilla en el bucle se puede llegar a tener algo casi unas 10 veces más rápido. Hay que tener en cuenta la arquitectura del computador que ejecuta el programa para poder realizar estas modificaciones, sin embargo, merece la pena pues se aumenta bastante el rendimiento.

# 1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES : (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s		nm-secuencial.s pmm-secuencial- modificado_b.s		pmm-secuencial- modificado_c.s	
.L16:	movq	call	clock_gettime	call	clock_gettime movl
	0(%rbp,%rbx,8),		movq		80(%rsp), %eax
%r11	0(201 50),201 52,0),		0(%rbp,%rbx,8),		movq
701 11	movq	%rdx	0(701 bp, 701 bx, 0),		\$0, 16(%rsp)
	(%r12,%rbx,8),	701 UX	movq		shrl
%rdi	(701 12,701 57,0),		(%r12,%rbx,8),		\$3, %eax
701 U.I.	xorl	%r11	(%112,%15x,0),		leal
	%r10d, %r10d	/01 11	xorl		0(,%rax,8),
	,		%r10d, %r10d	%ebx	υ(, // ax, σ),
	.p2align 4,,10		,	/₀enx	movl
140.	.p2align 3		.p2align 4,,10		
.L12:	maurl	112.	.p2align 3		%eax, 60(%rsp)
	movl	.L12:	maurl		subl
0/	(%r11,%r10,4),		movl		\$1, %eax
%ecx	1	04 - 41	(%r11,%r10,4),		addq
	leaq	%edi			\$1, %rax
04 *	0(,%r10,4),		movq		salq
%rsi	1	0,	(%r14,%r10,8),		\$5, %rax
	xorl	%rsi			mov1
	%eax, %eax		xorl		%ebx, 88(%rsp)
	.p2align 4,,10		%eax, %eax		movq
	.p2align 3		.p2align 4,,10		%rax, 64(%rsp)
.L10:			.p2align 3	.L19:	
	movq	.L10:			movq
	(%r14,%rax,8),		movl		72(%rsp), %rax
%rdx			(%rsi,%rax,4),		movq
	movl	%ecx			16(%rsp), %rbx
	(%rdx,%rsi),		imull		xorl
%edx			%edi, %ecx		%r13d, %r13d
	imull		addl		movq
	(%rdi,%rax,4),		%ecx, (%rdx,		\$0, 8(%rsp)
%edx		%rax,4)			movq
	addq		addq		(%rax,%rbx,8),
	\$1, %rax		\$1, %rax	%rax	
	addl		cmpl		movq
	%edx, %ecx		%eax, %r15d		%rax, 48(%rsp)
	cmpl		ja		.p2align 4,,10
	%eax, %r15d		.L10		.p2align 3
	ja		addq	.L15:	
	.L10		\$1, %r10		movl
	movl		cmp1		60(%rsp), %ed>
	%ecx,		%r10d, %r15d		testl
(%r11,%r10,4)	/		ja		%edx, %edx
, , ==-, -,	addq		.L12		je
	\$1, %r10		addq		.L21
	cmpl		\$1, %rbx		movq
	%r10d, %r15d		cmpl		16(%rsp), %rb>
	ja		%ebx, %r15d		movq
	.L12		ja		24(%rsp), %rax
	addq		.L16		xorl
	\$1, %rbx	.L9:			%r12d, %r12d
	cmpl		leag		movq
	%ebx, %r15d		48(%rsp), %rsi		· ·
					8(%rsp), %rcx
	ja . 16		xorl		xorl
10.	.L16		%edi, %edi		%ebp, %ebp
.L9:	1000		call		xorl
	leaq		clock_gettime	I	%r11d, %r11d

	48(%rsp), %rsi			xorl
	xorl			%r10d, %r10d
	%edi, %edi			xorl
	call			%r9d, %r9d
	clock_gettime			xorl
	mov1			%r8d, %r8d
	28(%rsp), %ebx			movq
	pxor			(%rax,%rbx,8),
	%xmm0, %xmm0		%rax	(10. 427)0. 527/5//
	movq		701 471	movq
	0(%rbp), %rdx			32(%rsp), %rbx
	mov1			movq
	\$.LC2, %esi			%rbx, %rdi
	movl			movq
	\$1, %edi			(%rbx,%rcx,8),
	movq		%rbx	(101 00, 101 00, 0),
			701 DX	xorl
%rax	0(%rbp,%rbx,8),			
%I ax	movil			%ecx, %ecx
	mov1			movq
	(%rdx), %edx			%rbx, 40(%rsp)
	leaq			leaq
0/512	8(,%rbx,8),			4(%rbx), %rdx
%r13	maurī			movq
	mov1			64(%rsp), %rbx
	(%rax,%rbx,4),			leaq
%ecx				(%rax,%rbx),
	movq		%r14	_
	56(%rsp), %rax			xorl
	xorl			%ebx, %ebx
	%ebx, %ebx			.p2align 4,,10
	subq			.p2align 3
	40(%rsp), %rax		.L10:	
	cvtsi2sdq			movq
	%rax, %xmm0			(%rdi), %rsi
	movq			addq
	48(%rsp), %rax			\$32, %rax
	subq			addq
	32(%rsp), %rax			\$64, %rdi
	movapd			addq
	%xmm0, %xmm1			\$32, %rdx
	pxor			movl
	%xmm0, %xmm0			(%rsi,%r13),
	divsd		%esi	( / //
	.LC1(%rip),			imull
%xmm1	(   -   -   / /			-32(%rax), %esi
	cvtsi2sdq			addl
	%rax, %xmm0			%esi, %ecx
	movl			movl
	\$1, %eax			-28(%rax), %esi
	addsd			imull
	%xmm1, %xmm0			
	call			-32(%rdx), %esi addl
	printf_chk			%esi, %r8d
	printi_cnk			movl
	16(%rsp), %eax			-24(%rax), %esi
	testl			imull
	%eax, %eax			-28(%rdx), %esi
	je			addl
140.	. L15			%esi, %r9d
.L18:				movl
	movq			-20(%rax), %esi
	0(%rbp,%rbx),			imull
%rdi				-24(%rdx), %esi
	call			addl
	free			%esi, %r10d
	movq			movl
	(%r12,%rbx),			-16(%rax), %esi
%rdi				imull
	call			-20(%rdx), %esi
	free			addl
	movq			%esi, %r11d
	(%r14,%rbx),			movl
%rdi				-12(%rax), %esi
	addq			imull
	\$8, %rbx			-16(%rdx), %esi
	call			addl
	free			%esi, %ebx
	cmpq			movl
	%r13, %rbx			-8(%rax), %esi
	,	I .		, ,

	jne			imull
	.L18			-12(%rdx), %esi
.L15:				addl
	movq			%esi, %ebp
	%rbp, %rdi			movl
	call			-4(%rax), %esi
	free			imull
	movq			-8(%rdx), %esi
	%r12, %rdi			addl
	call			%esi, %r12d
	free			cmpq
	movq			%r14, %rax
	%r14, %rdi			jne
	call			.L10
	free			addl
	xorl			%r8d, %ecx
	%eax, %eax			addl
	movq			%r9d, %ecx addl
	72(%rsp), %rbx			
	xorq			%r10d, %ecx addl
	%fs:40, %rbx			
	jne			%ecx, %r11d
	. L30			addl %r11d %eby
	addq ¢oo %rcn			%r11d, %ebx
	\$88, %rsp			addl
tata	.cfi_remember_s			%ebx, %ebp
tate	ofi dof -f- of			addl
foot 50	.cfi_def_cfa_of		114.	%ebp, %r12d
fset 56	nong		.L14:	mov1
	popq			movl
	%rbx			88(%rsp), %eax
5	.cfi_def_cfa_of			cmpl
fset 48				%eax, %r15d
	popq			jbe
	%rbp			. L11
	.cfi_def_cfa_of			movq
fset 40				24(%rsp), %rbx
	popq			movq
	%r12			16(%rsp), %rcx
	.cfi_def_cfa_of			movq
fset 32				(%rbx,%rcx,8),
	popq		%rsi	
	%r13			movq
	.cfi_def_cfa_of			32(%rsp), %rbx
fset 24				movq
	popq			8(%rsp), %rcx
	%r14			movq
	.cfi_def_cfa_of			(%rbx,%rcx,8),
fset 16			%rcx	
	popq			.p2align 4,,10
	%r15			.p2align 3
	.cfi_def_cfa_of		.L12:	
fset 8				movl
	ret			%eax, %edi
.L3:				addl
	.cfi_restore_st			\$1, %eax
ate				movl
	leaq		L	(%rsi,%rdi,4),
	32(%rsp), %rsi		%edx	
	xorl			imull
	%edi, %edi			(%rcx,%rdi,4),
			%edx	
				addl
				%edx, %r12d
				cmpl
				%eax, %r15d
				jne
				.L12
			.L11:	
				movq
				48(%rsp), %rax
				addq
				\$1, 8(%rsp)
				movl
				%r12d, (%rax,
			%r13)	• ,
				addq
				\$4, %r13
				movq
		I.	L	*

```
8(%rsp), %rax
                 cmpl
                 %eax, %r15d
                 .L15
                 addq
                 $1, 16(%rsp)
                 movq
                 16(%rsp), %rax
                 cmpl
                 %eax, %r15d
                 ja
                 .L19
.L9:
                 leaq
                 112(%rsp), %rsi
                 xorl
                 %edi, %edi
                 call
                 clock_gettime
```

#### **B) CÓDIGO FIGURA 1:**

CAPTURA CÓDIGO FUENTE: figura1-original.c

```
#include <stdio.h>
#include <time.h>
struct
     int a;
int b;
s[5000];
int main(int argc, char **argv)
     int ii, i, X1, X2;
int R[40000];
     struct timespec cgt1,cgt2; double ncgt;
     clock_gettime(CLOCK_REALTIME,&cgt1);
     for (ii = 1; ii <= 40000; ii++)</pre>
          X1 = 0; X2 = 0;
          for (i = 0; i < 5000; i++)
    X1 += 2 * s[i].a + ii;</pre>
          for (i = 0; i < 5000; i++)
X2 += 3 * s[i].b - ii;
          if ( X1 < X2 )
    R[ii] = X1;</pre>
          else
               R[ii] = X2;
     clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
     return 0;
```

### 1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones): Modificación a) –explicación-:

Se han agrupado los dos for en uno solo, pues hacen los mismo. Y se ha usado el operador ternario en lugar del if/else.

#### Modificación b) -explicación-:

También he agrupado los dos for en uno, y he sustituido el if/else por el operador ternario. A esta modificación le añado un desenrrollado de bucle de 5.

#### 1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1-modificado\_a.c

Capturas de pantalla (que muestren la compilación y que el resultado es correcto):

```
guillesiesta@guillesiesta:~$ gcc -02 figura1-original.c -o f
guillesiesta@guillesiesta:~$ ./f
R[0] = 0, R[39999] = -199995000

[Tiempo (seg.) = 0.205815694
guillesiesta@guillesiesta:~$ gcc -02 figura1-modA.c -o fA -w
iguillesiesta@guillesiesta:~$ ./fA
iR[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.157116599
guillesiesta@guillesiesta:~$
```

#### b) Captura figura1-modificado\_a.c

```
pmm-secuencial-modB.c × pmm-secuencial.s × figura1-original.c × figura1-modA.c × figura1-mod
  #include <stdio.h>
  #include <time.h>
  struct
        int a;
       int b;
s[5000];
   int main(int argc, char **argv)
        int ii, i, X1, X2;
int R[40000];
        struct timespec cgt1,cgt2; double ncgt;
        clock_gettime(CLOCK_REALTIME,&cgt1);
        for (ii = 1; ii <= 40000; ii++)
             X1 = 0; X2 = 0;
               for (i = 0; i < 5000; i+=4)
                  X1 += 2*s[i].a+ii;

X2 += 3*s[i].b-ii;

X1 += 2*s[i+1].a+ii;

X2 += 3*s[i+1].b-ii;

X1 += 2*s[i+2].a+ii;

X2 += 3*s[i+2].b-ii;

X1 += 2*s[i+3].a+ii;

X2 += 3*s[i+3].b-ii;
               R[ii] = (X1 < X2) ? X1 : X2;
        clock_gettime(CLOCK_REALTIME,&cgt2);
        ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
        printf("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
printf("\nTiempo (seg.) = %11.9f\n", ncgt);
        return 0;
```

#### Capturas de pantalla (que muestren la compilación y que el resultado es correcto):

```
guillesiesta@guillesiesta:~$ gcc -02 figura1-original.c -o f
guillesiesta@guillesiesta:~$ ./f
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.205815694
guillesiesta@guillesiesta:~$ gcc -02 figura1-modA.c -o fA -w
guillesiesta@guillesiesta:~$ ./fA
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.157116599
guillesiesta@guillesiesta:~$ gcc -02 figura1-modB.c -o fB -w
guillesiesta@guillesiesta:~$ ./fB
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.125431234
guillesiesta@guillesiesta:~$
```

#### **1.1. TIEMPOS:**

Modificación	-O2
Sin modificar	0.205815694
Modificación a)	0.157116599
Modificación b)	0.125431234

#### 1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Reducir el número de bucles se nota de gran manera, es evidente, pues se reduce a la mitad la ejecución. El desenrollado de bucle aún funciona mejor.

# 1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES: (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

Figura1-	original.s	figura1-	modificado_a.s	figura1	-modificado_b.s
call	clock_gettime	call	clock_gettime	call	clock_gettime
	leaq		leaq		leaq
	52(%rsp), %r9		52(%rsp), %r9		52(%rsp), %r10
	movl		movl		movl
	\$1, %ecx		<b>\$1</b> , %esi		\$1, %ecx
	movl		movl		movl
	\$s+40004, %r8d		\$s+40000, %r8d		\$s+40000, %r9d
	.p2align 4,,10		.p2align 4,,10		.p2align 4,,10
	.p2align 3		.p2align 3		.p2align 3
.L2:		.L2:		.L2:	
	movl		movl		movl
	\$s, %eax		\$s, %eax		\$s, %eax
	xorl		xorl		xorl
	%esi, %esi		%edi, %edi		%esi, %esi
	.p2align 4,,10		xorl		xorl
	.p2align 3		%ecx, %ecx		%edx, %edx
.L3:	, ,		.p2align 4,,10		.p2align 4,,10
	movl		.p2align 3		.p2align 3
	(%rax), %edx	.L3:	.pzarry c	.L3:	.p=a=19 0
	addg		movl		movl
	'				
	\$8, %rax		(%rax), %edx		(%rax), %edi
	leal		addq		addq
	(%rcx,%rdx,2),		\$8, %rax		\$32, %rax
%edx			leal		leal
	addl		(%rsi,%rdx,2),		(%rcx,%rdi,2),
	%edx, %esi	%edx		%edi	
	cmpq		addl		addl
	\$s+40000, %rax		%edx, %ecx		%edi, %edx
	jne .		movl		movl
	.L3		-4(%rax), %edx		-28(%rax), %edi
	movl		leal		leal
	\$s+4, %eax		(%rdx,%rdx,2),		(%rdi,%rdi,2),
	xorl	%edx	(% ux, % ux, z),	%r8d	(701 41,701 41,2),
	%edi, %edi	70CUX	subl	701 OU	movl
	•				
	.p2align 4,,10		%esi, %edx		-24(%rax), %edi
	.p2align 3		addl		sub1
.L4:	_		%edx, %edi		%ecx, %r8d
	movl		cmpq		leal
	(%rax), %edx		%rax, %r8		(%rcx,%rdi,2),
	addq		jne	%edi	
	\$8, %rax		.L3		addl
	leal		cmpl		%r8d, %esi
	(%rdx,%rdx,2),		%edi, %ecx		addl <sup>'</sup>
%edx	, , , , , , , , , , , , , , , , , , , ,		cmovg		%edi, %edx
	subl		%edi, %ecx		mov1
	%ecx, %edx		addl		-20(%rax), %edi
	addl		\$1, %esi		leal
			,		
	%edx, %edi		addq	0/ = =1 =	(%rdi,%rdi,2),
	cmpq		\$4, <sub>_</sub> %r9	%edi	
	%rax, %r8		movl		subl
	jne		%ecx, -4(%r9)		%ecx, %edi
		1	1	1	1 1
	. L4		cmpl		leal
	.L4 cmpl		стрі \$40001, %esi		leal (%rdi,%rsi),

Jige   1.12   mov1   1.16   mov1   32(%rsp), %rsi   1eal   32(%rsp), %rsi   wesi, (%rd)   wedi, %edi   call   wesi, %edx   mov1   1.2(%rax), %esi   leal   (%rsi, %rsi, 2), wesi   wesi, %edx   wesi, %edx   wesi, %edx   wesi   wesi, %edx   wesi   wesi, %edx   wesi   w				T	
Leal		jge	.L2		movl
Leal			leag		-16(%rax), %esi
Nesi					
Addi					
addl \$1, %ecx addq \$4, %r9 cup1 \$49001, %ecx jne 1.2 leaq 32(%rsp), %rsi xorl %edi call clock_gettime  %esi  %esi  movl -2(%rax), %esi leal (%rsi, %rsi, 2), %esi  movl -6(%rax), %esi leal (%rcx, %rsi, 2), %esi  subl %ecx, %esi leal (%rcx, %rsi, 2), %esi  dddl %esi, %edx movl -4(%rax), %esi leal (%rcx, %rsi, 2), %esi  subl %esi, %edx movl -4(%rax), %esi leal (%rcx, %rsi, 2), %esi  subl %esi, %edx movl -4(%rax), %esi leal (%rcx, %rsi, 2), %esi  subl %ecx, %esi addl %esi, %edx movl -6(%rax), %esi leal (%rcx, %rsi, 2), %esi  subl %ecx, %esi addl %esi, %edx cmpq %rax, %r9 jne .1.3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl %edx, %ecx addq \$2(%rsp), %rsi xorl %edi, %edi all	161	WC3T, (WI3)		%oci	(101 61, 101 51, 4),
\$1, %ecx addq \$4, %r9 cmpl \$4,0001, %ecx jne 1.2 leaq 32(%rsp), %rsi xorl %edi call clock_gettime della clock_gettime della clock_gettime della complete della call call call call call call call	.LO:			%esi	. 4.43
addq \$4, %F9 cnp1 \$40001, %ecx jne .12 leaq 32(%rsp), %rsi xorl %edi, %edi call clock_gettime  movl .2					
\$4, %r9 cmpl \$40001, %ecx jne 1.2 leaq 32(%rsp), %rsi xorl %edi, %edi call clock_gettime		<b>\$1,</b> %ecx	clock_gettime		%esi, %edx
Campl		addq			mov1
Campl		\$4, %r9			-12(%rax), %esi
\$40001, %ecx jne .12 leaq .32(%rsp), %rsi xor1 %edi, %edi call clock_gettime  **esi  **mov1 -6(%rax), %esi leal (%rcx,%rsi,2),  **esi  **addl **mov1 -4(%rax), %esi leal (%rsi,%rsi,2),  **esi  **subl **mov1 -6(%rax), %esi leal (%rsi,%rsi,2),  **esi  **subl **mesi, %edx mov1 -4(%rax), %esi leal (%rsi,%rsi,2),  **subl **ecx, %esi addl **edi, %esi cmpq %rax, %rs jne .13 cmpl **esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1  **edx, -4(%r10) cmpl \$40001, %ecx jne .12 leaq 32(%rsp), %rsi xor1 %edi, %edi cmory %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1  **edx, -4(%r10) cmpl \$40001, %ecx jne .12 leaq 32(%rsp), %rsi xor1 %edi, %edi call					
jne					
Leaq   32(%rsp), %rsi   %ccx, %esi   leal   %ccx, %esi   leal   (%rsi,%r8),   wedi   call   clock_gettime   wedi   call   (%rcx,%rsi,2),   wesi   add   (%rsi,%rsi,2),   wesi   wedi   wesi, %edx   movl   -4(%rax), %esi   leal   (%rcx,%rsi,2),   wesi   wesi, %edx   wesi   wesi, %edx   wesi   wesi, %edx   wesi   wesi   wesi, %esi   wesi, %edx   wesi   wesi, %edx   wesi, %edx   wesi   wesi   wesi, %edx   wesi   wesi, %edx   wesi   wesi				9/00i	(701 31)701 31)2))
leaq 32(%rsp), %rsi xorl %edi, %edi call clock_gettime  movl -8(%rax), %esi leal (%rcx,%rsi,2), %esi  addl %esi, %edx movl -4(%rax), %esi leal (%rcx,%rsi,2), %esi  subl %ecx, %esi addl %esi, %edx movl -4(%rax), %esi leal (%rsi,%rsi,2), %esi  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx addl \$1, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx addl \$1, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx addl \$1, %exi addl \$1, %exi addl \$2, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call				%es1	b.1
leal (%rs), %rsi xorl					
Xerdi					
Wedi   Call   Clock_gettime   Wedi   movl   -8(%rax), %esi   leal   (%rcx,%rsi,2), wesi   addl   Wesi, %edx   movl   -4(%rax), %esi   leal   (%rsi,%rsi,2), wesi   wesi					
Call   Clock_gettime		xorl			(%rsi,%r8),
Call   Clock_gettime		%edi, %edi		%edi	
clock_gettime  -8(%rax), %esi leal (%rcx,%rsi,2),  %esi  addl %esi, %edx mov1 -4(%rax), %esi leal (%rsi,%rsi,2),  %esi  subl %ecx, %esi addl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .13 cmpl %esi, %edx cmovg %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1 %edx, -4(%r10) cmpl \$40001, %ecx jne .12 leaq 32(%rsp), %rsi xorl %edi, %edi call					mov1
leal (%rcx,%rsi,2),					
(%rcx,%rsi,2),  addl %esi, %edx mov1 -4(%rax), %esi leal (%rsi,%rsi,2),  %esi  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .1.3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1 %edx, -4(%r10) cmpl \$40001, %ecx jne .1.2 leaq 32(%rsp), %rsi xor1 %edi, %edi call		orook_gerrime			
## ## ## ## ## ## ## ## ## ## ## ## ##					
addl %esi, %edx mov1 -4(%rax), %esi leal (%rsi,%rsi,2), %esi  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ex addl \$1, %ex addd \$4, %r10 mov1 %edx, -4(%r10) cmpl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xor1 %edi, %edi call					(%rcx,%rs1,2),
## ## ## ## ## ## ## ## ## ## ## ## ##				%esi	
mov1					addl
-4(%rax), %esi leal (%rsi,%rsi,2),  %esi  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1 %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xor1 %edi, %edi call					%esi, %edx
-4(%rax), %esi leal (%rsi,%rsi,2),  %esi  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1 %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xor1 %edi, %edi call					movl
leal (%rsi,%rsi,2),  %esi  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xor1 %edi, %edi call					
(%rsi, %rsi, 2),  subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .l3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .l2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
## subl ## ## ## ## ## ## ## ## ## ## ## ## ##					
subl %ecx, %esi addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call				0/	(%151,%151,2),
## ## ## ## ## ## ## ## ## ## ## ## ##				%esi	
addl %edi, %esi cmpq %rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					subl
### ### ##############################					%ecx, %esi
cmpq %rax, %r9 jne .l3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					addl
cmpq %rax, %r9 jne .l3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					%edi, %esi
%rax, %r9 jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 mov1 %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
jne .L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
.L3 cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
cmpl %esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
%esi, %edx cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
cmovg %esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
<pre>%esi, %edx addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call</pre>					%esi, %edx
addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					cmovg
addl \$1, %ecx addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					%esi, %edx
\$1, %ecx addq \$4, %r10 mov1 %edx, -4(%r10) cmp1 \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xor1 %edi, %edi call					
addq \$4, %r10 movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
\$4, %r10 mov1 %edx, -4(%r10) cmp1 \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xor1 %edi, %edi call					
movl %edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
%edx, -4(%r10) cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
cmpl \$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
\$40001, %ecx jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					cmpl
jne .L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					\$40001, %ecx
.L2 leaq 32(%rsp), %rsi xorl %edi, %edi call					
leaq 32(%rsp), %rsi xorl %edi, %edi call					
32(%rsp), %rsi xorl %edi, %edi call					
xorl %edi, %edi call					
%edi, %edi call					
call					
					%edi, %edi
clock_gettime					call
					clock_gettime

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada DAXPY (*Double precision- real Alpha X Plus Y*) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

for 
$$(i=1; i \le N, i++)$$
  $y[i] = a*x[i] + y[i];$ 

2.1. Genere los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan

en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CAPTURA CÓDIGO FUENTE: daxpy.c

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
void daxpy(int *y, int *x, int a, unsigned n, struct timespec *cgt1, struct timespec *cgt2)
     clock gettime(CLOCK REALTIME,cgt1);
     for (i=0; i<n; i++)
    y[i] += a*x[i];
clock_gettime(CLOCK_REALTIME,cgt2);</pre>
}
int main(int argc, char *argv[])
      if (argc < 3)
           fprintf(stderr, "ERROR: falta tam del vector y constante\n");
     unsigned n = strtol(argv[1], NULL, 10);
int a = strtol(argv[2], NULL, 10);
int *y, *x;
y = (int*) malloc(n*sizeof(int));
x = (int*) malloc(n*sizeof(int));
     unsigned i;
for (i=0; i<n; i++)
{
           y[i] = i+2;
x[i] = i*2;
     struct timespec cgt1,cgt2; double ncgt;
     daxpy(y, x, a, n, &cgt1, &cgt2);
     ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
     printf("y[0] = %i, y[%i] = %i\n", y[0], n-1, y[n-1]);
printf("\nTiempo (seg.) = %11.9f\n", ncgt);
      free(y);
free(x);
     return 0;
```

	-O0	-Os	-02	-O3
Tiempos ejec.	0.023195	0.00925	0.008748	0.010607
	608	9249	321	655

CAPTURAS DE PANTALLA (que muestren la compilación y que el resultado es correcto):

```
guillesiesta@guillesiesta:~$ gcc -00 daxpy.c -o d00
guillesiesta@guillesiesta:~$ gcc -02 daxpy.c -o d02
quillesiesta@quillesiesta:~$ qcc -03 daxpy.c -o d03
guillesiesta@guillesiesta:~$ ./d00 10000000 10000000
y[0] = 2, y[9999999] = 542894465
Tiempo (seg.) = 0.023195608
guillesiesta@guillesiesta:~$ ./dO2 10000000 10000000
v[0] = 2. v[9999999] = 542894465
Tiempo (seg.) = 0.008748321
guillesiesta@guillesiesta:~$ ./dO3 10000000 10000000
y[0] = 2, y[9999999] = 542894465
Tiempo (seg.) = 0.010607655
guillesiesta@guillesiesta:~$ gcc -Os daxpy.c -o dOs
guillesiesta@guillesiesta:~$ ./dOs 10000000 10000000
y[0] = 2, y[9999999] = 542894465
Tiempo (seg.) = 0.009259249
guillesiesta@guillesiesta:~$
```

#### COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Con O0 usamos direcciones relativas a la pila y con O2, registros de la arquitectura. Así ahorramos muchas operaciones move innecesarias y como se puede ver abajo obtenemos un código mucho más reducido para O2 que para O0, donde estamos moviendo a registros de la arquitectura direcciones relativas de la pila y operando con estas dimensiones

Por último, en el O3, el compilador ha hecho un desenrollado del bucle, dándonos un código más largo, y en este caso más lento que O2.

CÓDIGO EN ENSAMBLADOR (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón): (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

dax	py00.s	dax	py0s.s	da	xpy02.s	daxp	y03.s
movl	\$0,	xorl	%eax,	xorl	%eax, %eax	call	
-4(%rbp)		%eax			testl		
	jmp	.L2:			%ebp, %ebp	clock_getti	ne
	.L2		cmpl		je		testl
.L3:			%eax,		.L4		%r14d,
	movl	%ebp			.p2align	%r14d	
	-4(%rbp),		jbe	4,,10			je
%eax			. L6		.p2align 3		.L10
	leaq		movl	.L5:			leaq
	0(,				movl		16(%r12),
%rax,4), %	rdx	(%r12,%rax	,4), %edx			%rax	
	movq		imull	0(%r13,%r	ax,4), %esi		cmpq
	-		%r13d,		imull		%rax,
24(%rbp),	%rax	%edx			%r12d,	%rbx	
	addq		addl	%esi			leaq
	%rax,		%edx,		addl		16(%rbx),

%rdx		(%rbx,%rax,4	1)		%esi,	%rax	
701 47	movl	(701 57,701 47,7	incq	(%rbx,%rax,4)		701 <b>G</b> X	setnb
	-4(%rbp),		%rax	(701 271 701 427 7 1 )	addq		%dl
%eax	-(/oi bb)/		jmp		\$1, %rax		cmpq
70CUX	leaq		.L2		cmpl		%rax,
	0(,	.L6:	. L Z		%eax, %ebp	%r12	/01 αλ,
%rax,4), %rc>			addq			/01 12	setnb
% (ax, 4), % (C)					ja		
	movq	0/15 0 10	\$24,	1.4.	. L5		%al
0.4/0/mbm) 0/m	-	%rsp	مد: مامد	.L4:			orb
24(%rbp), %ra			.cfi_def		popq		%al, %dl
	addq	_cfa_offset			%rbx		je
	%rcx,		movq	5 55 1 10	.cfi_def_c		.L3
%rax	-		%r9,	fa_offset 40			cmpl
	movl	%rsi	_		movq		\$6, %r14d
	(%rax),		xorl		%r14, %rsi		jbe
%ecx			%edi,		xorl		. L3
	movl	%edi			%edi, %edi		movq
	-4(%rbp),		popq		popq		%rbx,
%eax			%rbx		%rbp	%rax	
	leaq		.cfi_def		.cfi_def_c		andl
	0(,	_cfa_offset	32	fa_offset 32			\$15, %eax
%rax,4), %rsi			popq		popq		shrq
	movq		%rbp		%r12		\$2, %rax
	-		.cfi_def		.cfi_def_c		negq
32(%rbp), %ra	ax	_cfa_offset		fa_offset 24			%rax
	addq		popq		popq		andl
	%rsi,		%r12		%r13		\$3, %eax
%rax	/		.cfi_def		.cfi_def_c		cmpl
701 47	movl	_cfa_offset		fa_offset 16	1011_001_0		%r14d,
	(%rax),		popq	14_011300 10	popq	%eax	701 1447
%eax	(701 47),		%r13		%r14	70CUX	cmova
70CUX	imull		.cfi_def		.cfi_def_c		%r14,
	IIIUII	_cfa_offset		fa_offset 8	.cri_uer_c	%rax	701 ±4,
36(%rbp), %ea	-	_cra_orrset	0	Ta_UTTSet o	jmp	701 ax	xorl
30(%i bh), %ea	addl				Jilib		
				alaak gattima		0/ody	%edx,
0/	%ecx,			clock_gettime	;	%edx	£ £ 1
%eax							testl
	movl					04	%eax,
45. 4 3	%eax,					%eax	
(%rdx)							je
	addl						. L4
	\$1,						movl
-4(%rbp)							(%r12),
.L2:						%edx	
	movl						imull
	-4(%rbp),						%r13d,
%eax						%edx	
	cmpl						addl
	-						%edx,
40(%rbp), %ea	ax					(%rbx)	•
	jb						cmpl
	.L3						\$1, %eax
	movq						movl
	-						\$1, %edx
56(%rbp), %ra							je
- ( 2   )   /31 0	movq						.L4
	%rax,						movl
%rsi	, or any						4(%r12),
/01 J±	movl					%edx	T(/01 14),
	\$0, %edi					/0CUA	imull
	ΨU, ⁄0CUI						%r13d,
						%edx	/01 ±3U,
						/0€UX	

addl %edx, cmp1 \$3, %eax				
### Add   Cmpl   Cmpl   S3, %eax   mov1   S2, %edx   jne   .14   mov1   S3, %eax   mov1   S3, %eax   mov1   S3, %edx   mov1   S3, %edx   mov1   S3, %edx   mov1   S1, %edx   mov1   %r13d, %edi   mov1   %r13d, %edi   mov1   %r13d, %ecx   sub1   %eax, %ecx   sub1   %eax, %edi   mov4   12(%rsp)   %xmm4   saiq   S2, %rax   saiq   S2, %rax   saiq   S4, %rax   saiq   S4, %rax   saiq   S6, %r9d   pshufd   %9, %xmm4, %xmm2   %xmm4, %xmm2   %xmm2, %				addl
A(%rbx)   Cmpl   33, %eax   mov1   \$2, %eax   mov1   %ea				
Cmp1			4 / 0/ m la v / \	weak,
\$3, %eax mov1 \$2, %edx inul1 %ri3d, %edx add1 %edx, mov1 \$3, %eax  add1 %edx  mov1 %ri3d, %edi %eax, %edx subl %eax, %exi leal -4(%rib), %smm4 %xmm4 slq \$2, %rax leal -4(%rib), %rb, %rb, %rb, %rb, %rb, %rb, %rb, %rb			4(%rbx)	
\$3, %eax mov1 \$2, %edx inul1 %ri3d, %edx add1 %edx, mov1 \$3, %eax  add1 %edx  mov1 %ri3d, %edi %eax, %edx subl %eax, %exi leal -4(%rib), %smm4 %xmm4 slq \$2, %rax leal -4(%rib), %rb, %rb, %rb, %rb, %rb, %rb, %rb, %rb				cmpl
mov1				
\$2, %ext				
jne				
jne				\$2, %edx
1.4 movol 8(%r12), %edx				
Medx				
S(%-12), %edx				
Sedx				movl
Sedx				8(%r12).
imull %edx addl %edx addl %edx, 8(%rbx) mov1 \$3, %edx  """""""""""""""""""""""""""""""""""			%ody	- ( - //
### ##################################			⁄₀eux	
## ## ## ## ## ## ## ## ## ## ## ## ##				
## ## ## ## ## ## ## ## ## ## ## ## ##				%r13d,
### ### ##############################			%ody	
## Sedx, ## Sedx ## Se			%eux	
8(%rbx)				addl
8(%rbx)				%edx.
S3, %edx   S4, %edi			9/%rhv\	,,
S3, %edx   Nedi			o(winx)	_
1.4:   mov1   %r14d, %edi   mov1   %r13d,   12(%rsp)   xor1   %ecx, %ecx   sub1   %eax, %edi   movd   12(%rsp), %xmm4   salq   \$2, %rax   leal   -4(%rdi), %esi   leaq (%rbx, %rax), %r10   xor1   %r9d, %r9d, %r9d, %r9d, %r9d, %r12, %rax   shr1   \$2, %esi   add   %r12, %rax   shr1   \$2, %esi   add   \$1, %esi   movdqa   %xmm2, %xmm2   %xmm3   leal   0(,				
1.4:   mov1   %r14d, %edi   mov1   %r13d,   12(%rsp)   xor1   %ecx, %ecx   sub1   %eax, %edi   movd   12(%rsp), %xmm4   salq   \$2, %rax   leal   -4(%rdi), %esi   leaq (%rbx, %rax), %r10   xor1   %r9d, %r9d, %r9d, %r9d, %r9d, %r12, %rax   shr1   \$2, %esi   add   %r12, %rax   shr1   \$2, %esi   add   \$1, %esi   movdqa   %xmm2, %xmm2   %xmm3   leal   0(,				\$3, %edx
mov1			14.	,
%r14d, %edi			.L4.	_
%r14d, %edi				
Wedi				
movl %r13d, 12(%rsp) xor1 %ecx, %ecx subl %eax, %edi movd 12(%rsp), %xmm4  \$alq \$2, %rax leal -4(%rdi), %esi leaq (%rbx, %rax), %r10 xor1 %r9d, %r9d pshufd \$0, xor1 %r9d, %rax  \$heli \$1, %esi add1 \$1, %esi movdqa %xmm2, %xmm2  %xmm3  leal 0(,			%adi	,
### ##################################			∕0€UI	
12(%rsp)				movl
12(%rsp)				%r13d.
xorl %ecx, %ecx subl %eax, %edi movd 12(%rsp), %xmm4 salq \$2, %rax leal -4(%rdi), %esi leaq (%rbx, %rax), %r10 xorl %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shrl \$2, %esi addl \$1, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,			12/0/rcn)	70. 20 4. /
%ecx subl %eax, %edi movd 12(%rsp), %xmm4 salq \$2, %rax leal -4(%rdi), %esi leaq (%rbx, %rax), %r10 xorl %red, %red, %red pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,			12(%i Sp)	_
%ecx subl %eax, %edi movd 12(%rsp), %xmm4 salq \$2, %rax leal -4(%rdi), %esi leaq (%rbx, %rax), %r10 xorl %r9d, %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,				xorl
%ecx subl %eax, %edi movd 12(%rsp), %xmm4 salq \$2, %rax leal -4(%rdi), %esi leaq (%rbx, %rax), %r10 xorl %r9d, %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,				
Subl   %eax,   %edi   movd   12(%rsp),   %xmm4   salq   \$2, %rax   leal   -4(%rdi),   %esi   leaq   (%rbx,   %r9d,   %r9d,   %r9d   pshufd   \$0,   %xmm4,   %xmm2   addq   %r12,   %rax   shrl   \$2, %esi   addl   \$1, %esi   movdqa   %xmm2,   %xmm3   leal   0(,			%acv	,
%edi movd 12(%rsp), %xmm4  \$alq \$2, %rax leal -4(%rdi), %esi leaq (%rbx, %rax), %r10  xorl %r9d pshufd \$0, %xmm4, %xmm2  addq %r12, %rax  \$hr1 \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm2, %xmm2  %xmm3  leal 0(,			<b>⊘</b> ピしX	
%edi				subl
%edi				%eax
movd 12(%rsp), %xmm4  salq \$2, %rax leal -4(%rdi), %esi  leaq (%rbx, %rax), %r10  xor1 %r9d, %r9d  pshufd \$0, %xmm4, %xmm2  addq %r12, %rax  shr1 \$2, %esi add1 \$1, %esi movdqa %xmm2, %xmm2, %xmm3  leal 0(,			0/odi	ποαπη
12(%rsp), %xmm4   salq			%ea1	
%xmm4				movd
%xmm4				12(%rsn).
salq \$2, %rax leal			0/2 cmm 4	(/oi op / /
\$2, %rax leal			70XIIIII14	
leal				salq
leal				\$2. %rax
-4(%rdi), %esi leaq (%rbx, %rax), %r10 xorl %r9d, %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shr1 \$2, %esi add1 \$1, %esi movdqa %xmm2, %xmm2, %xmm3 leal 0(,				
%esi leaq (%rbx, %rax), %r10 xorl xr9d, %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shr1 \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,				Tear
%esi leaq (%rbx, %rax), %r10 xorl xr9d, %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shr1 \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,				-4(%rdi),
leaq (%rbx, %rax), %r10			%esi	
(%rbx, %rax), %r10 xorl xord, %r9d pshufd \$0, %xmm4, %xmm2 addq %r12, %rax shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,				lead
%rax), %r10				
%rax), %r10				(%rbx,
xorl %r9d, %r9d  pshufd \$0,  %xmm4, %xmm2  addq %r12,  %rax  shrl \$2, %esi addl \$1, %esi movdqa %xmm2,  %xmm3  leal 0(,			%rax), %r10	
%r9d  pshufd \$0,  %xmm4, %xmm2  addq %r12,  %rax  shr1 \$2, %esi add1 \$1, %esi movdqa %xmm2,  %xmm3  leal 0(,			,,,	vorl
%r9d  pshufd \$0,  %xmm4, %xmm2  addq %r12,  %rax  shr1 \$2, %esi add1 \$1, %esi movdqa %xmm2,  %xmm3  leal 0(,				
%r9d  pshufd \$0,  %xmm4, %xmm2  addq %r12,  %rax  shr1 \$2, %esi add1 \$1, %esi movdqa %xmm2,  %xmm3  leal 0(,				%r9d,
pshufd \$0, %xmm4, %xmm2  addq %r12, %rax  shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3  leal 0(,			%r9d	
\$0, %xmm4, %xmm2 addq %r12, %rax shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3 leal 0(,				nchufd
%xmm4, %xmm2				
%xmm4, %xmm2				\$0,
addq %r12, %rax  shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3  leal 0(,			%xmm4, %xmm2	
%r12, %rax  shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3  leal 0(,			,	adda
%rax  shrl \$2, %esi addl \$1, %esi movdqa %xmm2,  %xmm3  leal 0(,				
%rax  shrl \$2, %esi addl \$1, %esi movdqa %xmm2,  %xmm3  leal 0(,				%r12,
shrl \$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3  leal 0(,			%rax	
\$2, %esi addl \$1, %esi movdqa %xmm2, %xmm3				chr1
addl \$1, %esi movdqa %xmm2, %xmm3				
addl \$1, %esi movdqa %xmm2, %xmm3				
\$1, %esi movdqa %xmm2, %xmm3				
movdqa %xmm2, %xmm3 leal 0(,				
%xmm2, %xmm3 leal 0(,				
%xmm2, %xmm3 leal 0(,				movdqa
%xmm3 leal 0(,				
leal 0(,			0/vmm2	. 3/
0(,			%XIIIIIIろ	_
0(,				leal
Ponto Arquitactura y Teopología da Computadorea				

Srsi, 4), %r8d		la	\ 0( \ 0 \ 1	
\$32, %xmm3 .L6: movdqu (%rax, %rox), %xmm0 add1 \$xmm1, %rod movdqa %xmm1, \$xmm3  \$xmm3		%rs1,4		oorla
Scumm3				
Note   Note   Note		9/ymm2	1	p32,
(%rax), %xmm0			n	novdau
### ##################################				
addl		%rcv)		(701 ax,
S.1, %r9d		701 GX ) ,		Ibhe
movdqa   3xmm1   prlq   \$32   %xmm0   prlq   \$32   %xmm0   pmuludq   %xmm3, %xmm0   pshufd   \$8, %xmm1, %xmm1   pshufd   \$8, %xmm1, %xmm1   punpckldq   %xmm1, %xmm1   movdqa   (%r10,%rcx), %xmm0   paddd   %xmm1, %xmm0   movaps   %xmm0, (%r10,%rcx)   addq   \$16, %rcx   cmp1   %esi, %red, %edx   cmp1   %r				
3xmm1				
Sxmm1				
psrlq   \$32, %xmm0   pmuludq   %xmm3, %xmm0   pshufd   \$8, %xmm0, %xmm0   pmuludq   %xmm2, %xmm1   pupckldq   \$8, %xmm1, %xmm1   movdqa   (%r10,%rcx), %xmme   paddd   %xmm1, %xmm0   movaps   %xmm0, (%r10,%rcx)   addq   \$16, %rcx   xmm0, (%r10,%rcx)   xmm0   xmm0, (%r10,%rcx)   xmm0   xmm0, (%r10,%rcx)		%xmm1	ŕ	o,
\$32, %xmm0  pmuludq %xmm3, pshufd \$8, %xmm1, %xmm1  pshufd \$8, %xmm1, %xmm1  pupckldq %xmm0, %xmm0  %xmm1  movdqa (%r10,%rcx), %xmm0  movaps %xmm0  %xmm0  movaps %xmm1, %xmm0  movaps %xmm0  %xmm0  movaps %xmm0, %xmm0  movaps %xmm0  %xmm0  movaps %xmm0  movaps %xmm0  %xmm0  movaps %xmm1  %xmm0  movaps %xmm0  %xmm1  movaps %xmm0  %xmm0  movaps %xmm0  %xmm1  movaps %xmm1  %xmm0  movaps %xmm0  %xmm0  movaps %xmm0  %xmm1  movaps %xmm0  %xmm0  movaps %xmm0  m		· · · · · · · · · · · · · · · · · · ·	r	osrla
%xmm0				
muludq		%×mm0	•	,
%xmm0   %xmm0   pshufd			r	omuluda
Sxmm0				
pshufd		%×mm0		,
\$8, %xmm0, %xmm0 pmuludq %xmm2, %xmm1 pshufd \$8, %xmm1, %xmm1 punpckldq %xmm0, %xmm1 movdqa  (%r10,%rcx), %xmm0 paddd %xmm1, %xmm0 movaps %xmm0, (%r10,%rcx) addq \$16, %rcx cmp1 %esi, %r9d jb .16 addll %r8d, %edx cmpl %r8d, %edx mov1 (%r12,%rax,4), %ecx			ŗ	shufd
%xmm0, %xmm0				
pmuludq		%xmm0,	%xmm0	
%xmm1			ŗ	omuludq
pshufd				
\$8, %xmm1, %xmm1 punpckldq %xmm0, %xmm1 movdqa  (%r10,%rcx), %xmm0 paddd %xmm1, %xmm0 movaps %xmmo, (%r10,%rcx) addq \$16,%rcx cmpl %esi, %r9d jb .L6 addl %r8d, %edx cmpl %r8d, %edx		%xmm1		
%xmm1, %xmm1 punpckldq %xmm0, %xmm1 movdqa  (%r10,%rcx), %xmm0 paddd %xmm1, %xmm0 movaps %xmm0, (%r10,%rcx) addq \$16,%rcx cmp1 %esi, %r9d jb .16 addl %r8d, %edx cmpl %r8d, %edx				
punpckldq %xmm0,				\$8,
%xmm1 movdqa  (%r10,%rcx), %xmm0 paddd %xmm1,  %xmm0 movaps %xmm0, (%r10,%rcx) addq \$16, %rcx cmp1 %esi,  %r9d jb .L6 addl addl %r8d, %edx cmp1 %r8d, %edx cmp1 %r8d, %edi je .L10 mov1 %edx, %eax mov1 (%r12,%rax,4), %ecx		%xmm1,		
%xmm1				
movdqa   (%r10,%rcx), %xmm0			9	%xmm0,
(%r10,%rcx), %xmm0		%xmm1		
paddd   %xmm1,   %xmm0			n	novdqa
paddd   %xmm1,   %xmm0		(0) 10	04	6
%xmm0		(%r10,		
%xmm0				
movaps %xmm0, (%r10,%rcx)  addq \$16, %rcx cmp1 %esi,  %r9d  jb .L6 addl %r8d,  %edx  cmpl %r8d,  %edi  je .L10 movl %edx, %edx  movl (%r12,%rax,4), %ecx		9/ymm0	7	%ΧΙΙΙΙΙ <b>Ι</b> Ι,
%xmm0, (%r10,%rcx)  addq \$16, %rcx cmp1 %esi,  %r9d  jb .L6 addl %r8d,  %edx  cmp1 %r8d,  %edi  je .L10 mov1 %edx, %eax mov1 (%r12,%rax,4), %ecx		%XIIIIIU	n	001000
(%r10,%rcx) addq \$16, %rcx cmpl %esi,  %r9d  jb .L6 addl %r8d,  %edx  cmpl %r8d,  %edi  je .L10 movl %edx, %eax  movl (%r12,%rax,4), %ecx				
addq \$16, %rcx cmpl %esi, %r9d  jb .L6 addl %r8d, %edx  cmpl %r8d, %edx  ige .L10 movl %edx, %eax  movl (%r12,%rax,4), %ecx		(%r10		oxiiiiio,
\$16, %rcx cmpl %esi, %r9d  jb		(701 10)	701 CA )	nhhe
cmpl %esi, %r9d  jb .L6 addl %r8d, %edx  cmpl %r8d, %edi  je .L10 movl %edx, %eax  movl (%r12,%rax,4), %ecx				\$16. %rcx
%esi, %r9d  jb .L6 addl %r8d, %edx  cmpl %r8d, %edi  je .L10 movl %edx, %eax  movl (%r12,%rax,4), %ecx			4	cmpl
%r9d  jb .L6 addl %r8d,  %edx  cmpl %r8d,  %edi  je .L10 movl %edx, %eax  movl (%r12,%rax,4), %ecx			9	esi.
jb .L6 addl %r8d, %edx cmpl %r8d, %edi  je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx		%r9d		,
.L6 addl %r8d, %edx cmpl %r8d, %edi je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx		75. 64	4	jb
addl %r8d, %edx cmpl %r8d, %edi je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx				. L6
%r8d, %edx cmpl %r8d, %r8d, %edi je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx			6	addl
%edx cmpl %r8d, %edi je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx				
%r8d, %edi  je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx		%edx		
%r8d, %edi  je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx				
je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx				
.L10 movl %edx, %eax movl (%r12,%rax,4), %ecx		%edi		
movl %edx, %eax movl (%r12,%rax,4), %ecx			j	je
%edx, %eax movl (%r12,%rax,4), %ecx				
%eax movl (%r12,%rax,4), %ecx				
movl (%r12,%rax,4), %ecx			9	%edx,
(%r12,%rax,4), %ecx		%eax		
			n	novl
<u>imull</u>		(%r12,		
			j	rwntt

		%r13d,
	%ecx	
		addl
		%ecx,
	(%rbx,%rax,4)	
		leal
	0/	1(%rdx),
	%eax	
		cmpl
		%eax,
	%r14d	jbe
		.L10
		movl
		IIIOVI
	(%r12,%rax,4)	%ACY
		addl
		\$2, %edx
		imull
		%r13d,
	%ecx	150,
		addl
		%ecx,
	(%rbx,%rax,4)	///////////////////////////////////////
		cmpl
		%edx,
	%r14d	,
		jbe
		.L10
		MOAT
		movl %edx,
		%edx,
	%eax	%edx,
	%eax	
	%eax (%r12,%rax,4)	%edx, imull , %r13d
	%eax (%r12,%rax,4)	%edx, imull , %r13d addl
	%eax (%r12,%rax,4)	%edx, imull , %r13d
	%eax (%r12,%rax,4) (%rbx,%rax,4)	%edx, imull , %r13d addl
	%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:	%edx, imull , %r13d addl %r13d,
	%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:	%edx, imull , %r13d addl %r13d,
	%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:	%edx, imull , %r13d addl %r13d, addq \$16, %rsp
	%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:	%edx, imull , %r13d addl %r13d,
	%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:	%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme
	%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state	%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48</pre>	%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_
	%eax  (%r12,%rax,4)  (%rbx,%rax,4) .L10:  mber_state  cfa_offset 48	%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_ movq
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48</pre>	%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48 %rsi</pre>	<pre>%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp,</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48 %rsi</pre>	<pre>%edx, imull , %r13d add1 %r13d, addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48 %rsi</pre>	<pre>%edx, imull , %r13d add1 %r13d, addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp,</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48 %rsi %edi</pre>	<pre>%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi,</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48 %rsi %edi</pre>	<pre>%edx, imull , %r13d addl %r13d, addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10: mber_state cfa_offset 48 %rsi %edi</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx</pre>
	%eax  (%r12,%rax,4)  (%rbx,%rax,4) .L10:  mber_state cfa_offset 48  %rsi  %edi	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:  mber_state cfa_offset 48 %rsi %edi  cfa_offset 40</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:  mber_state cfa_offset 48 %rsi %edi  cfa_offset 40</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_ popq</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:  mber_state cfa_offset 48 %rsi %edi  cfa_offset 40</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_ popq %rbp</pre>
	%eax  (%r12,%rax,4)  (%rbx,%rax,4) .L10:  mber_state  cfa_offset 48  %rsi  %edi  cfa_offset 40	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_ popq %rbp .cfi_def_</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:  mber_state cfa_offset 48 %rsi %edi  cfa_offset 40 cfa_offset 32</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_ popq %rbp .cfi_def_</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:  mber_state cfa_offset 48 %rsi %edi  cfa_offset 40 cfa_offset 32</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp,  xorl %edi, popq %rbx .cfi_def_ popq %rbp .cfi_def_ popq %rbp .cfi_def_</pre>
	<pre>%eax (%r12,%rax,4) (%rbx,%rax,4) .L10:  mber_state cfa_offset 48 %rsi %edi  cfa_offset 40 cfa_offset 32</pre>	<pre>%edx, imull , %r13d addl %r13d,  addq \$16, %rsp .cfi_reme .cfi_def_ movq %rbp, xorl %edi, popq %rbx .cfi_def_ popq %rbp .cfi_def_</pre>

	cfa_offset 24
	popq
	%r13
	.cfi_def_
	cfa_offset 16
	popq
	%r14
	.cfi_def_
	cfa_offset 8
	jmp
	clock_gettime