DidadoY

Previo 10

1. Buscad para qué sirven y que operandos admiten las instrucciones psubb, pcmpgtb, movdqa, movdqu y emms.
psubb: sirve para restar enteros de 128 bits (psubb a, b ---> b-a)

pcmpgtb: sirve para comparar con signo mayor o igual a nivel de bytes (pcmpgtb a, b ----> a >= b)

movdqa: sirve para mover elementos alineados double quadword de un registro MMX a otro, o de un registro MMX a una posición de memoria de 128 bits.

movdqu: sirve para mover elementos no alineados double quadword de un registro MMX a otro, o de un registro MMX a una posición de memoria de 128 bits.

emms: vacía los registros MMX (no necesita operandos)

2. Buscad para qué sirve y cómo se usa en C la propiedad __attribute__ y el atributo aligned.

__attribute__ sirve para especificar propiedades especiales a variables, parámetros de funciones, estructuras…

aligned sirve para especificar la forma en que los datos se organizan y acceden a la memoria de la computadora.

3. Programad en ensamblador sin usar instrucciones SSE una versión de la rutina que
hay en Procesar.c procurando hacerla lo m´as r´apida posible (1 solo bucle, acceso
secuencial...):

```
        movl $0, %edi  # edi <- i = 0
        movl 8(%ebp), %eax # eax <- mata[0]
        movl 12(%ebp), %ebx # ebx <- matb[0]
        movl 16(%ebp), %esi # esi <- matc[0]
        movl 20(%ebp), %edx # edx <- n
        imull %edx, %edx # edx <- n x n (para hacer solo 1 bucle)
bucle:  cmpl %edx, %edi
        jge fibucle
        movb (%eax), %cl #cl <- mata
        subb (%ebx), %cl #cl <- mata - matb
        movb %cl, (%esi) # matc = mata - matb
if:     cmpb $0, %esi
        jle else
        movb $255, (%esi) # matc = 255
        jmp fiif
else:   movb $0, (%esi) # matc = 0
fiif:   incl %edi #++i
        incl %eax #stride
        incl %ebx #stride
        incl %esi #stride
        jmp bucle
fibucle:
```

4. Explicad cómo se puede cargar un valor inmediato en un registro xmm usando la instrucción movdqu.

Primero se carga el inmediato a memoria para después cargarlo en xmm. Por ejemplo si hacemos movdqu xmm1, xmm2 moveremos de xmm2 a xmm1 siendo xmm2 la dirección base de memoria.

5. Programad en ensamblador una versión SIMD de la rutina que hay en Procesar.c usando las instrucciones psubb, pcmpgwtb y movdqu.

```
        movl $0, %edi  # edi <- i = 0
        movl 8(%ebp), %eax # eax <- mata[0]
        movl 12(%ebp), %ebx # ebx <- matb[0]
        movl 16(%ebp), %esi # esi <- matc[0]
        movl 20(%ebp), %edx # edx <- n
        imull %edx, %edx # edx <- n x n (para hacer solo 1 bucle)
bucle:  cmpl %edx, %edi
        jge fibucle
        movdqu (%eax), %xmm0 #xmm0 <- mata
        movdqu (%ebx), %xmm1 #xmm1 <- matb
        psubb %xmm1, %xmm0 #xmm0 <- mata - matb
        movdqu %xmm0, (%esi) <- matc = mata - matb
        movdqu $zero, %xmm2
        pcmpgtb %xmm0, %xmm2
        movdqu %xmm0, (%esi)
        addl $16, %edi #++i pero estamos en registros de 128bits
        addl $16, %eax #stride
        addl $16, %ebx #stride
        addl $16, %esi #stride
        jmp bucle
fibucle:
```

6. Escribid un código en ensamblador que, a partir de un valor almacenado en un registro, averigüe si es múltiplo de 16

```
#suponemos que el numero esta en el registro eax

        movl $0, %edx
        movl $16, %ebx
        divl %ebx # eax / 16 , edx = % (ebx/16)
        cmpl $0, %edx #comparamos, si el modulo de la division del valor que hay en el registro eax / 16 es 0, entonces es que es multiplo, si no, no es multiplo
        jne noesmultiplo
esmultiplo:
        ...
noesmultiplo:
        ...
```