Nombre: Jia Long Ji Qiu Grupo: 23

Nombre: Jiabo Wang

Hoja de respuesta al Estudio Previo

1. Explicad para qué sirven y qué operandos admiten las instrucciones:

`psubb`

psubb hace la diferencia entre enteros de bytes empaquetados.

PSUBB src, dest // dest <- dest - src, src = {m128, xmm}, dest = {xmm}

`pcmpgtb`

pcmpgtb compara enteros de bytes empaquetados con signo.

PCMPGTB src, dest // Si dest > src, el byte correspondiente de dest pasará a ser 0xFF, en caso contrario será 0x00, src = {m128, xmm}, dest = {xmm}

`movdqa`

movdqa mueve un double quadword alineado de un registro xmm a otro, de xmm a memoria o viceversa. NOTA: No se puede utilizar dos operandos de memoria en la misma instrucción.

MOVDQA src, dest // dest <- src, dest = {m128, xmm}, src = {m128, xmm}

`movdqu`

movdqu mueve un double quadword sin alinear de un registro xmm a otro, de xmm a memoria o viceversa. NOTA: No se puede utilizar dos operandos de memoria en la misma instrucción.

MOVDQU src, dest // dest <- src, dest = {m128, xmm}, src = {m128, xmm}

`emms`

emms permite al procesador salir del estado MMX (empty MMX state).

EMMS

2. La propiedad `__attribute__` y el atributo `aligned` sirven para:

__attribute__ permite especificar propiedades especiales a variables a la hora de declararlas. El atributo aligned es uno de estos y especifica los bytes a los que debe estar alineada la variable.

3. Programad en ensamblador una versión de la rutina que hay en `Procesar.c` procurando hacerla lo más rápida posible.

```
        pushl %ebp
        movl %esp, %ebp
        pushl %ebx
        pushl %edi
        pushl %esi

        movl 8(%ebp), %eax          # eax <- @mata
        movl 12(%ebp), %ebx         # ebx <- @matb
        movl 16(%ebp), %ecx         # ecx <- @matc
        movl 20(%ebp), %esi         # esi <- n
        imul %esi, %esi          # esi <- n*n
        movl $0, %edi               # edi <- 0
for:    movb (%eax, %edi), %dl
        subb (%ebx, %edi), %dl
```

```
if:     cmpb $0, %dl
        je else
        movb $255, (%ecx, %edi)
        jmp endif
else:   movb $0, (%ecx, %edi)
endif:  incl %edi
        cmpl %esi, %edi
        jge endfor
        jmp for
endfor: popl %esi
        popl %edi
        popl %ebx
        movl %ebp, %esp
        popl %ebp
        ret
```

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

Debido a que movdqu no admite valores inmediatos, sino que hace movimientos de un registro xmm a otro, memoria a xmm o xmm a memoria, haría falta cargar dicho valor inmediato (128 bits) en memoria previamente para moverlo al registro xmm.

5. Programad en ensamblador una versión `SIMD` de la rutina que hay en `Procesar.c`.

```
        pushl %ebp
        movl %esp, %ebp
        pushl %ebx
        pushl %edi
        pushl %esi
        movl 8(%ebp), %eax          # eax <- @mata
        movl 12(%ebp), %ebx         # ebx <- @matb
        movl 16(%ebp), %ecx         # ecx <- @matc
        movl 20(%ebp), %esi         # esi <- n
        imul %esi, %esi          # esi <- n*n
        movl $0, %edi               # edi = k = 0
        psubb %xmm0, %xmm0
```

```
for:    movdqu (%eax, %edi), %xmm1
            # xmm1 <- mata[k]..[k + 15]
        movdqu (%ebx, %edi), %xmm2
        psubb %xmm2, %xmm1
            # xmm1 <- xmm1 - matb[k]..[k + 15]
        pcmpgtb %xmm0, %xmm1
        movdqu %xmm1, (%ecx, %edi)
        addl $16, %edi
        cmpl %esi, %edi
        jge endfor
        jmp for
endfor: popl %esi
        popl %edi
        popl %ebx
        movl %ebp, %esp
        popl %ebp
        ret
```

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

Suponiendo que el valor está almacenado en el registro %eax:

```
        andl $0xF, %eax
        jne false           // Salta a false cuando ZF = 0,
                            // es decir, cuando los últimos
                            // 4 bits de %eax != 0

true:   ...
        jmp end

false:  ...

end:    ...
```

Un valor será multiple de 16 si y solo si sus 4 bits de menor peso equivalen a 0x0, por lo tanto, basta con hacer una operación aritmética AND con la máscara 0xF y comprobar si el resultado es 0 o no.