

Práctica 2 - Modelo de procesamiento SIMD

Organización del Computador 2

2do Cuatrimestre 2022

La práctica se divide en secciones, para cada una se sugiere un conjunto de instrucciones útiles para resolver los ejercicios. Los ejercicios marcados con ★ constituyen un subconjunto recomendado de ejercitación. Algunos ejercicios poseen una serie de preguntas bajo el indicador **Para pensar** cuyo objetivo es fomentar la discusión, análisis y profundización de los temas. Se aconseja leerlas a modo de repaso y consolidación.

Instrucciones de movimiento de datos y aritméticas básicas

- Mov. de datos: `movd`, `movq`, `movdqu`
- Aritméticas: `paddb`, `paddw`, `paddq`, `paddq`, `psubb`, `psubw`, `psubd`, `psubq`

Ejercicio 1

- a) `void SumarVectores(char *A, char *B, char *Resultado, int dimension)`

Para pensar: ¿Qué sucede si la suma de dos componentes de los vectores supera el valor 255? ¿Qué diferencia hay entre las instrucciones `paddusw` y `paddsw`? ¿Y entre `paddusb` y `paddsb`? ¿Qué cambios deberían realizarse sobre las funciones anteriores si el tipo de datos es `short`, `int` o `long long int`?

Nota: Puede asumir que la dimensión de los vectores es de un tamaño múltiplo de la cantidad de elementos que procesa simultáneamente.

Instrucciones de comparación, lógicas y de nivel de bit

- Comparación: `pcmpgtb`, `pcmpgtw`, `pcmpgtd`, `pcmpeqb`, `pcmpeqw`, `pcmpeqd`
- Lógicas: `pand`, `por`, `pxor`, `pandn`
- Nivel de Bit: `psrlw`, `psrld`, `psrlq`, `psrldq`, `psllw`, `psllq`, `pslldq`

Ejercicio 2

- a) `void InicializarVector(short *A, short valorInicial, int dimension)`
b) `void DividirVectorPorPotenciaDeDos(int *A, int potencia, int dimension)`
c) `void FiltrarMayores(short *A, short umbral, int dimension)`
Pone en unos (`0xF...F`) aquellos elementos del vector cuyo valor sea mayor al umbral y en ceros (`0x0...0`) aquellos que sean menores o iguales.

Para pensar: ¿Qué cambios debería hacerles a las funciones anteriores en caso de que la dimensión de los vectores no sea múltiplo de la cantidad de elementos que procesa simultáneamente?

Instrucciones de movimiento de datos y aritméticas complejas

- Desempaquetado: `punpcklbw`, `punpcklwd`, `punpckldd`, `punpckhbw`, `punpckhwd`, `punpckhdd`
- Aritméticas: `pmullw`, `pmulld`, `pmulhw`, `pmulhd`, `pmaddwd`, `pmaxub`, `pmaxuw`, `pmaxud`, `pminub`, `pminuw`, `pminud`
- Empaquetado: `packsswb`, `packssdw`, `packuswb`, `packusdw`

Ejercicio 3

- a) void MultiplicarVectores(short *A, short *B, int *Res, int dimension)
- b) int ProductoInterno(short *A, short *B, int dimension)
- c) void SepararMaximosMinimos(char *A, char *B, int dimension)
Deja en A los máximos y en B los mínimos. Es decir, para cada i ,
 $A[i] = \max(A[i], B[i])$ y $B[i] = \min(A[i], B[i])$
- d) void SumarRestarAlternado(int *A, int *B, int* Res, int dimension)
Es decir, el Res tiene que seguir el siguiente patrón: $\text{Res} = (A_1 + B_1, A_2 - B_2, A_3 + B_3, A_4 - B_4, \dots)$

Para pensar: ¿Qué cambios habría que hacerle a la función **MultiplicarVectores** si el tipo de datos de los elementos de Res fuese short? ¿Qué diferencia hay entre las instrucciones packuswb y packsswb?

Instrucciones de reordenamiento de datos

- Reordenamiento: pshufb, pshufw, pshufd

Ejercicio 4

- a) void Intercalar(char *A, char *B, char *vectorResultado, int dimension)

Para pensar: ¿Cómo haría la función Intercalar si no dispusiese de las instrucciones de reordenamiento? Considere dos alternativas.

Instrucciones de punto flotante

- Mov. de datos: movups, movaps, movupd, movapd
- Aritméticas: addps, addpd, subps, subpd, mulps, mulpd, divps, divpd, sqrtps, sqrtpd

Ejercicio 5

- a) void SumarErroresResiduales(float *A, float *B, float *Res, int dimension)

$$\text{SSE} = \sum_{i=1}^N (A_i - B_i)^2$$

- b) void NormalizarVector(float *A, float *Res, int dimension)

Para pensar: ¿Qué cambios debería hacerles a las funciones anteriores si el tipo de dato ahora fuese punto flotante de **doble precisión**? ¿Qué diferencia hay entre la instrucción addps y addss? ¿Y entre addpd y addsd?

Instrucciones de conversión de tipos de datos

- Conversión: cvtdq2ps, cvtps2dq, cvtdq2pd, cvtpd2dq

Ejercicio 6

- a) void ProductoEscalar(short *A, float escalar, int dimension)
- b) void ParteEntera(float *A, int *Res, int dimension)
- c) void Normalizar(int *A, float *Res, int dimension)

Para pensar: ¿Qué cambios debería hacerles a las funciones anteriores si el tipo de dato ahora fuese punto flotante de **doble precisión**?

Ejercicios de parcial

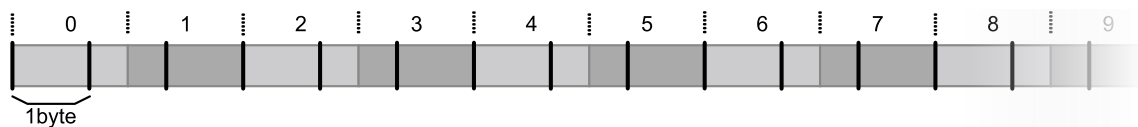
Ejercicio 7

Considerar un vector de 16 números enteros con signo de 24 bits almacenados en big-endian.

- Construir una función en ASM utilizando SIMD que dado un puntero al vector de números mencionado, retorne la suma de los números del vector como un entero de 4 bytes.
- Modificar la función anterior para que a los números pares los multiplique por π . En este caso el resultado debe ser retornado como *double*.

Ejercicio 8 ★

Sea un vector de enteros sin signo de 12 bits ordenados de forma contigua como muestra la figura:



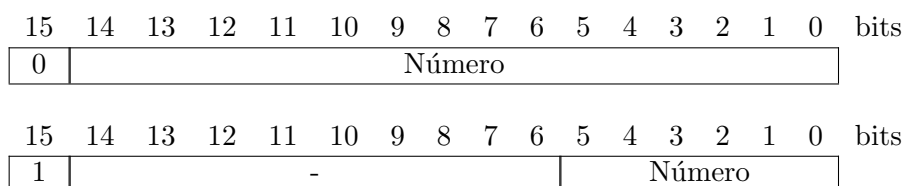
Notar que cada valor del vector ocupa un byte y la mitad del siguiente, o medio byte y el siguiente completo.

- Construir una función en ASM usando SIMD que dado un arreglo determine si este está ordenado de menor a mayor.
- Construir una función en ASM usando SIMD que dado un arreglo y un número de 12 bits determine si el número pertenece al arreglo.

Nota: Considerar para la solución que el arreglo tiene exactamente 24 elementos.

Ejercicio 9 ★

Considerar un tipo de datos de 16 bits denominado **quinceyseis**, que dependiendo del bit más significativo, almacena dos datos diferentes. Si el bit 15 es cero, entonces se almacena un número **con signo** de 15 bits. Si el bit 15 es uno, entonces se almacena un número **sin signo** de 6 bits en la parte menos significativa. Los bits restantes en este caso pueden tener cualquier valor.

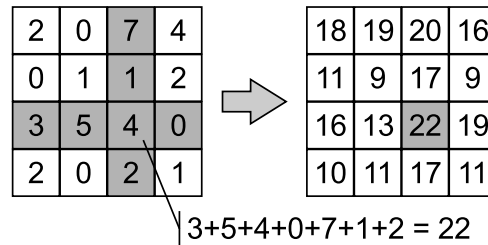


Sea un vector de **quinceyseis** con tamaño (**size**) múltiplo de 8. Implementar en ASM utilizando SIMD y procesando la mayor cantidad de datos simultáneamente, las siguientes funciones:

- `void convertirQuince2SeisConSaturacion(quinceyseis* data, size_t size):` Transforma todos los números de 15 bits en números de 6 bits saturando.
- `float promedioQuinceSeis(quinceyseis* data, size_t size):` Calcula el promedio entre todos los números en precisión simple.

Ejercicio 10

Se tiene una matriz de 4×4 enteros sin signo de 32bits. Se desea implementar una función denominada **sumatorias** que se encarga de generar una nueva matriz que contiene por cada elemento la sumatoria de todos los elementos en la fila y columna de la matriz original, contando solo una vez cada uno de los números.



- a) Implementar en ASM utilizando instrucciones de SIMD la función `sumatorias` (`unsigned int* sumatorias(unsigned int* matriz)`). Procesar la mayor cantidad posible de elementos, justificar.
- b) Considerando que la matriz de entrada no contiene enteros, sino *floats* y que en la nueva matriz resultado se deben almacenar enteros de 32bits, modifique el código anterior para reflejar este cambio (puede reescribir todo el código o una parte del mismo).