TEC | Tecnológico de Costa Rica

Área Académica Ingeniería en Computadores

Arquitectura de Computadores II

Taller 3

Extensión SIMD: SSEx

Profesor

Ing. Luis Barboza Artavia

Estudiantes

Jose Ignacio Granados Marín

Mónica Selena Waterhouse Montoya

II – 2022

Investigación

1. ¿En qué consiste un extensión SIMD llamada SSE?

   SSE es un proceso o tecnología que permite operar múltiples datos con una sola instrucción. Los procesadores más antiguos solo procesan un único elemento de datos por instrucción. SSE permite que la instrucción maneje múltiples elementos de datos. Se utiliza en aplicaciones intensivas, como gráficos 3D, para un procesamiento más rápido [1].

2. ¿Cuáles tipos de datos son soportados por este tipo de instrucciones?

   Los tipos de datos soportados por este tipo de instrucciones son [2]:

   - 64-bit double-precision floating point
   - 64-bit integers
   - 32-bit integers
   - 16-bit short integers
   - 8-bit bytes
   - 8-bit characters

3. ¿Cómo se realiza la compilación de un código fuente en C que utilice el set SSEx de Intel?

   Para compilar un determinado archivo que haga uso de estas extensiones SIMD, el comando a realizar es el siguiente [3]:

$$gcc - msse3 - O3 - Wall - lrt < fileName >.c - o\ exe$$

4. ¿Qué importancia tienen la definición de variables y el alineamiento de memoria al trabajar con un set SIMD vectorial, como SSE?

   Cuando se realiza la declaración de una cierta cantidad de variables, en realidad se están reservando y asignando espacios de memoria de n-bits, dicho espacios deben estar alineados con la memoria, principalmente, porque las instrucciones realizarán cargas, reservas y operaciones con datos de tamaño de n-bits, los cuales representarán cada uno de los vectores que se hayan definido en un determinado código [4].

Análisis

1. Explicar las variables oddVector, evenVector y los diversos data en términos de cómo se definieron (instrucción utilizada), el tipo de dato que representan y por qué la cantidad de argumentos.

   ● **oddVector:** es un vector de 128-bit de números enteros, al que se le asignan cuatro números enteros impares de 32-bit.

   ● **evenVector:** es un vector de 128-bit de números enteros, al que se le asignan cuatro números enteros pares de 32-bit.

   ● **data:** es una variable de tipo entero que guarda el valor que retorna la función *_mm_extract_epi32*. Esta función recibe el vector y la posición que se desea extraer de él, de esta forma se obtiene un número entero de 32-bit.

2. ¿Qué sucede si las variables data se imprimieran con un ciclo y no uno por uno? ¿Por qué ocurre eso?

Al momento de realizar la compilación del código, se generaría un error debido a que la extensión *_mm_extract_epi32* recibe un entero como segundo parámetro un entero y no una variable. Es por dicha razón que se debe realizar la impresión de los datos uno a uno.

3. Compile el código fuente y adjunte una captura de pantalla con el resultado de la ejecución.

```
monica@monica-VirtualBox:~/Downloads/Taller3_SIMD$ gcc -o helloWorld helloWorld.c
monica@monica-VirtualBox:~/Downloads/Taller3_SIMD$ ./helloWorld
Probando SSE
Result **********************
5       7       9       11
monica@monica-VirtualBox:~/Downloads/Taller3_SIMD$
```

Ejercicios prácticos

1. Realice un programa en C que busque el elemento mayor en cada columna de una matriz y lo guarde en un vector. La matriz corresponde a 4x3 y está compuesta por enteros (32 bits).

```
monica@monica-VirtualBox:~/Documents/ArquiII/Workshop3$ gcc -o maxColumnNumber maxColumnNumber.c -msse4.1
monica@monica-VirtualBox:~/Documents/ArquiII/Workshop3$ ./maxColumnNumber
------ Get Matrix Values ------
Value [0, 0]: 1
Value [0, 1]: 2
Value [0, 2]: 3
Value [0, 3]: 4
Value [1, 0]: 5
Value [1, 1]: 6
Value [1, 2]: 7
Value [1, 3]: 8
Value [2, 0]: 9
Value [2, 1]: 10
Value [2, 2]: 11
Value [2, 3]: 12
------------------------------

------------ Matrix -----------
1       2       3       4
5       6       7       8
9       10      11      12
------------------------------

----------- Result ------------
9       10      11      12
------------------------------
```

```c
#include <emmintrin.h>
#include <smmintrin.h>
#include <stdio.h>

#define ROWS 3
#define COLS 4

void getMatrixValues(__m128i matrix[ROWS]) {

    printf("------ Get Matrix Values ------\n");

    for (int i = 0; i < ROWS; i++) {

        int values[] = {0, 0, 0, 0};
        int value;

        for (int j = 0; j < COLS; j++) {
            printf("Value [%d, %d]: ", i, j);
            scanf("%d", &value);
            values[j] = value;
        }

        matrix[i] = _mm_set_epi32(values[3], values[2], values[1], values[0]);
    }

    printf("--------------------------------\n\n");
}

void showMatrix(__m128i matrix[ROWS]) {

    printf("------------ Matrix -----------\n");

    for (int i = 0; i < ROWS; i++) {
        printf("%d \t", _mm_extract_epi32(matrix[i], 0));
        printf("%d \t", _mm_extract_epi32(matrix[i], 1));
        printf("%d \t", _mm_extract_epi32(matrix[i], 2));
        printf("%d \t", _mm_extract_epi32(matrix[i], 3));
        printf("\n");
    }

    printf("--------------------------------\n\n");
}

__m128i getMaxValues(__m128i matrix[ROWS]) {

    __m128i max_values = _mm_set_epi32(0, 0, 0, 0);

    for (int i = 0; i < ROWS; i++) {
        max_values = _mm_max_epi32(max_values, matrix[i]);
    }

    return max_values;
}

int main(int argc, char const *argv[]) {

    __m128i matrix[ROWS];
    getMatrixValues(matrix);
    showMatrix(matrix);
    __m128i max_values = getMaxValues(matrix);

    printf("----------- Result ------------\n");

    printf("%d \t", _mm_extract_epi32(max_values, 0));
    printf("%d \t", _mm_extract_epi32(max_values, 1));
    printf("%d \t", _mm_extract_epi32(max_values, 2));
    printf("%d \t\n", _mm_extract_epi32(max_values, 3));

    printf("--------------------------------\n\n");

    return 0;
}
```

2. Realice un programa en C que realice la multiplicación de un vector de 4 números enteros por una matriz 4x4.

```
nachogranados@NachoGranados-VirtualBox:~/Documents/GitHub/ComputerArchitecture2.Classwork3$ gcc -o multMatrixVector multMatrixVector.c -mss
e4.1
nachogranados@NachoGranados-VirtualBox:~/Documents/GitHub/ComputerArchitecture2.Classwork3$ ./multMatrixVector
------ Get Matrix Values ------
Value [0, 0]: 1
Value [0, 1]: 2
Value [0, 2]: 3
Value [0, 3]: 4
Value [1, 0]: 5
Value [1, 1]: 6
Value [1, 2]: 7
Value [1, 3]: 8
Value [2, 0]: 9
Value [2, 1]: 10
Value [2, 2]: 11
Value [2, 3]: 12
Value [3, 0]: 13
Value [3, 1]: 14
Value [3, 2]: 15
Value [3, 3]: 16
------------------------------

------------ Matrix -----------
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16
------------------------------
```

```
------ Get Vector Values ------
Value [0]: 1
Value [1]: 2
Value [2]: 3
Value [3]: 4
------------------------------

------------ Result ------------
------------ Matrix -----------
1       4       9       16
5       12      21      32
9       20      33      48
13      28      45      64
------------------------------
```

```c
#include <emmintrin.h>
#include <smmintrin.h>
#include <stdio.h>

#define N 4

void getMatrixValues(__m128i matrix[N]) {

    printf("------ Get Matrix Values ------\n");

    for (int i = 0; i < N; i++) {

        int values[] = {0, 0, 0, 0};
        int value;

        for (int j = 0; j < N; j++) {
            printf("Value [%d, %d]: ", i, j);
            scanf("%d", &value);
            values[j] = value;
        }

        matrix[i] = _mm_set_epi32(values[3], values[2], values[1], values[0]);
    }

    printf("-------------------------------\n\n");
}

__m128i createVector() {

    int values[] = {0, 0, 0, 0};
    int value;

    printf("------ Get Vector Values ------\n");

    for (int i = 0; i < N; i++) {
        printf("Value [%d]: ", i);
        scanf("%d", &value);
        values[i] = value;
    }

    __m128i vector = _mm_set_epi32(values[3], values[2], values[1], values[0]);
    printf("-------------------------------\n\n");

    return vector;
}

void multMatrixVector(__m128i result[N], __m128i matrix[N], __m128i vector) {

    for (int i = 0; i < N; i++) {
        result[i] = _mm_mullo_epi32(matrix[i], vector);
    }
}

void showMatrix(__m128i matrix[N]) {

    printf("------------ Matrix -----------\n");

    for (int i = 0; i < N; i++) {
        printf("%d \t", _mm_extract_epi32(matrix[i], 0));
        printf("%d \t", _mm_extract_epi32(matrix[i], 1));
        printf("%d \t", _mm_extract_epi32(matrix[i], 2));
        printf("%d \t", _mm_extract_epi32(matrix[i], 3));
        printf("\n");
    }

    printf("-------------------------------\n\n");
}

int main(int argc, char const *argv[]) {

    __m128i matrix[N], result[N];
    getMatrixValues(matrix);
    showMatrix(matrix);
    __m128i vector = createVector();
    multMatrixVector(result, matrix, vector);

    printf("----------- Result ------------\n");
    showMatrix(result);

    return 0;
}
```

Referencias

[1] Intel. (2022, 13 octubre). *Intel® Instruction Set Extensions Technology.* Intel Support. https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html

[2] *MMX and SSE MMX data types.* Ele.uva.es. https://www.ele.uva.es/~jesman/BigSeti/ftp/Microprocesadores/Intel/00_mmx_sse.pdf.

[3] P. Cordes and J. M. (2018, 10 junio). *Does compiler use SSE instructions for a regular C code?* Stack Overflow. https://stackoverflow.com/questions/50786263/does-compiler-use-sse-instructions-for-a-regular-c-code

[4] *Memory alignment requirements for SSE and AVX instructions.* (2020, 29 octubre). blog.ngzhian. https://blog.ngzhian.com/sse-avx-memory-alignment.html