Laboratorio de Electrónica Digital II
Práctica No. 5: Ordenamiento Ascendente de Números Enteros con Signo Empleando el Algoritmo *Merge Sort*

Profesores
Luis Fernando Castaño L. (luis.castanol@udea.edu.co)
Luis Germán García M. (german.garcia@udea.edu.co)

Octubre 3, 2023

Fecha de entrega:	Del 10 al 13 de Octubre de 2023
Medio de entrega:	https://virtualingenieriaudea.co/
Sustentación:	Horario de Laboratorio
Valor Práctica:	6% del curso

1 Introducción

En esta práctica de laboratorio, el grupo de estudiantes hará uso del lenguaje ensamblador para procesadores ARM de 32-bits, con el propósito de implementar el algoritmo *Merge Sort*, para ordenar números enteros con signo en orden ascendente (no decreciente). El algoritmo se compone de dos funciones, **MergeSort** y **Fuse**, las cuales deben ser implementadas en lenguaje ensamblador respetando los algoritmos dados en esta guía. La función **MergeSort** es una función recursiva, por lo que se hace necesario el uso del *Stack* del procesador. Los datos de entrada al programa estarán

ubicados en la memoria en posiciones específicas. Los números ordenados deberán ser escritos a la memoria sin sobreescribir los datos de entrada. El funcionamiento del programa se comprobará mediante el uso del simulador CPUlator.

2 Objetivo de la Práctica

Desarrollar un programa para el procesador ARM, empleando lenguaje ensamblador, que implemente el algoritmo *Merge Sort* para ordenar un conjunto de números enteros con signo ubicados en la memoria de manera ascendente (no decreciente) y realizar la correcta simulación para comprobar su funcionamiento.

3 Procedimiento

Para el correcto diseño e implementación de la práctica, es necesario leer completamente esta guía. Se sugiere seguir el procedimiento indicado a continuación:

a. Comprender la arquitectura del conjunto de instrucciones y manejo del *Stack* del procesador ARM, además del algoritmo *Merge Sort*.

b. Elaborar los Pseudocódigos o diagramas de flujo para el programa en ensamblador a desarrollar, el cual debe llevar a cabo la funcionalidad que se indicará más adelante.

c. Desarrollar el programa solicitado empleando lenguaje ensamblador para el procesador ARM basado en los Pseudocódigos o diagramas de flujo realizados previamente.

d. Simular el programa desarrollado para el procesador ARM y verificar el correcto funcionamiento del mismo. Llevar a cabo las correcciones pertinentes, si fuese el caso.

e. Enviar el código fuente de su programa en lenguaje ensamblador para el procesador ARM antes de la fecha límite.

f. Sustentar el diseño en el horario de laboratorio correspondiente. Consideraciones sobre la sustentación se dan al finalizar esta guía.

4 Especificaciones

El programa a desarrollar debe estar en capacidad de realizar el ordenamiento ascendente (no decreciente) de un conjunto N de valores numéricos con signo. El método de ordenamiento a implementar se denomina *Merge Sort* y será implementado mediante dos funciones llamadas **mergeSort** y **fuse**. La función **mergeSort** ordena una sucesión de valores en orden no decreciente empleando el algoritmo **fuse**. Los algoritmos para las funciones **mergeSort** y **fuse** se describen en los Algoritmos 1 y 2, respectivamente. El grupo de estudiantes se encargará de llamar la función **mergeSort** desde el cuerpo principal de su programa en ensamblador, pasando, de manera apropiada, los argumentos

que dicha función requiere. Para mayor información sobre el algoritmo, pueden consultar el libro MATEMÁTICAS DISCRETAS, Sexta edición, de Richard Johnsonbaugh.

Algorithm 1: Función MergeSort: ordenamiento no decreciente

Input : Sucesión $s_i, ..., s_j$, índice del primer elemento de la sucesión i, índice del último elemento de la sucesión j

Output: Sucesión $s_i, ..., s_j$ en orden no decreciente

```
1  mergeSort (s, i, j)
       // Caso base i igual a j
2      if (i == j) then
3          return
       // Divide la sucesión
4      m ← ⌊(i + j)/2⌋
       // Ordenar las dos sucesiones
5      mergeSort(s, i, m)
6      mergeSort(s, m + 1, j)
       // Fusionar ambas sucesiones
7      fuse(s, i, m, j, c)
       // Copiar la sucesión c, la salida de fuse, en la sucesión s
8      for k ← i to j do
9          s_k ← c_k
       // Retorno
10     return
```

El valor de N, así como los N valores numéricos a ordenar, estarán ubicados en la memoria. Los rangos de valores para los datos que serán entregados al programa son los siguientes:

(i) N: valor entero en el rango $2 \leq N \leq 200$.

(ii) Datos: valores enteros en el rango $-2^{31}(-2147483648) \leq VALOR \leq 2^{31} - 1(+2147483647)$.

Todos los valores antes mencionados se cargarán, junto con las instrucciones del programa, en el momento en que inicie la simulación. Durante la ejecución, el programa almacenará en memoria los valores ordenados de manera no decreciente.

El código a desarrollar se basará en la plantilla que se indica en la Fig. 1. Para los datos que se leen desde la memoria tenga en cuenta que:

(i) N se encuentra ubicado en la dirección dada por la etiqueta **N**.

(ii) El primer valor de los datos se encuentra ubicado en la dirección dada por la etiqueta **Data**, el segundo en **Data + 4**, el tercero en **Data + 8** y así sucesivamente.

Algorithm 2: Función Fuse: fusión de dos sucesiones

Input : Dos sucesiones no decrecientes $s_i, ..., s_m$ y $s_{m+1}, ..., s_j$, índices i, m y j
Output: Sucesión $c_i, ..., c_j$ que consiste en los elementos $s_i, ..., s_m$ y $s_{m+1}, ..., s_j$ mezclados en una sucesión no decreciente

```
1  fuse (s, i, m, j, c)
       // p es la posición en la sucesión s_i, ..., s_m
2      p ← i
       // q es la posición en la sucesión s_{m+1}, ..., s_j
3      q ← m + 1
       // r es la posición en la sucesión c_i, ..., c_j
4      r ← i
       // Se copia el menor entre s_p y s_q
5      while p ≤ m ∧ q ≤ j do
6          if s_p < s_q then
7              c_r ← s_p
8              p ← p + 1
9          else
10             c_r ← s_q
11             q ← q + 1
12         r ← r + 1
       // Se copia el resto de la primera sucesión
13     while p ≤ m do
14         c_r ← s_p
15         p ← p + 1
16         r ← r + 1
       // Se copia el resto de la sucesión
17     while q ≤ j do
18         c_r ← s_q
19         q ← q + 1
20         r ← r + 1
       // Retorno
21     return
```

Por otro lado los datos, resultado del ordenamiento, se escribirán en la memoria así: el primer dato de la lista de valores ordenados se almacenará en la dirección de memoria dada por la etiqueta **SortedData**, el segundo en **SortedData + 4**, el tercero en **SortedData + 8** y así sucesivamente.

Recuerde que la función **mergeSort** es una función recursiva, por lo que es obligatorio emplear el *Stack* del procesador ARM.

```
.global _start
    .equ    MAXN,   200
    .text
_start:
    /* Inicio de Programa:
     * Inicialización de registros y lectura de valores requeridos desde la memoria.
     */
    WRITE THE INITIALIZATION HERE

    /* Cuerpo del programa:
     * Código principal que hace uso de la subrutina mergeSort para ordenar los N valo
     * de la sucesión de manera no decreciente en memoria.
     */
    WRITE THE BODY HERE

    /* Fin de Programa:
     * Bucle infinito para evitar la búsqueda de nuevas instrucciones.
     */
finish:
    b finish

    /* Funciones mergeSort y Fuse:
     * Cuerpos de las dos funciones indicadas, haciendo uso del Stack del procesador
     * y realizando lo indicado en los algoritmos dados en la guía.
     */
    WRITE THE FUNCTIONS HERE

    .data
    /* Constantes y variables propias:
     * Utilice esta zona para declarar sus constantes y variables requeridas
     */

    /* Constantes y variables dadas por el profesor:
     * Esta zona contiene la definición de N, Data y SortedData, las cuales pueden ir
     * en cualquier orden
     */
N:          .dc.l   12
Data:       .dc.l   1,15,-79,35,16,-564,8542,-89542,12021,54215,12,-35
SortedData: .ds.l   MAXN
```

Fig. 1: Plantilla para el código a desarrollar

El grupo de trabajo deberá guardar su programa en un archivo de nombre **mergesort_xy.asm**, donde **x** es la letra inicial del primer apellido del integrante 1 y **y** es la letra inicial del primer apellido del integrante 2. Para la simulación, utilice el simulador dado en las referencias (al final de este documento) y compruebe el funcionamiento del programa con sus propios valores de prueba.

5 Entrega

El grupo de trabajo deberá adjuntar únicamente el archivo en ensamblador (extensión asm). Para esta práctica no se solicitará ningún tipo de reporte.

El nombre del archivo deberá tener el siguiente formato:
p5_primerapellidointegrante1_primerapellidointegrante2_horariolaboratorio.asm.
Ejemplo: si el primer apellido de ambos integrantes es **Castano** y **Garcia**, respectivamente, y el laboratorio es el Martes 9-12, entonces el archivo debe ser nombrado: *p5_castano_garcia_m9-12.asm.*

6 Evaluación

La evaluación de la práctica se divide en dos partes: funcionamiento (50%) y sustentación (50%). La nota del funcionamiento se asigna por igual a todos los integrantes del grupo de trabajo (máximo dos personas por equipo), mientras que la nota de sustentación es individual. La sustentación podrá realizarse de tres maneras posibles:

a. Un examen corto al finalizar la sesión de laboratorio.

b. Solicitud de cambios al código por parte del profesor de laboratorio.

c. Un par de preguntas orales que pongan a prueba los conocimientos del estudiante sobre el desarrollo de la práctica.

En caso un estudiante obtenga una nota inferior a 3.0 en la sustentación, la nota final de la práctica para el estudiante en mención será la que obtuvo en la sustentación, es decir, no se tendrá en cuenta el funcionamiento en el cálculo.

Cada grupo de trabajo deberá sustentar la práctica en un tiempo de 15 minutos, 8 minutos para revisar la funcionalidad y 7 minutos para la sustentación en cualquiera de las tres modalidades presentadas. Es importante tener abierto el CPUlator con el programa listo para cuando el profesor llegue a su puesto de trabajo. No habrá tiempo para hacer correcciones de último momento.

7 Referencias

a. Harris, Sarah and Harris, David. Digital Design and Computer Architecture: ARM Edition. Morgan Kaufmann Publishers Inc. 2015. ISBN: 0128000562

b. CPUlator Computer System Simulator
`https://cpulator.01xz.net/?sys=arm`