Universidad de San Carlos de Guatemala

Guatemala Vacaciones diciembre 2024

Arquitectura de Computadores y Ensambladores 1 - Sección A

Ingeniero Luis Espino

Auxiliar Jhonathan Daniel Tocay Cotzojay

Proyecto Fase 3

Grupo 3

Nombre	Carné
Bismarck Estuardo Romero Lemus	201708880
Pedro Alejandro Zetino Páez	202004750
Christian Alexandro Aragón García	202000308
Naomi Rashel Yos Cujcuj	202001814
Kevin Estuardo Sotoj García	201710130
Josué Nabí Hurtarte Pinto	202202481
Ángel Isaías Mendoza Martínez	202180003
Anderson Danilo García Alvizures	201602834

Introducción

El proyecto Fase 3 se centró en el análisis de datos generados por un sistema de monitoreo y riego automatizado, utilizando el lenguaje ensamblador ARM64. Este sistema fue diseñado para optimizar la gestión del agua y la temperatura en un invernadero, analizando parámetros como la humedad relativa, las temperaturas interna y externa, y el nivel de agua en el tanque. El objetivo principal de esta etapa fue desarrollar un programa que permitiera realizar cálculos estadísticos básicos y exportar los resultados en un archivo de texto plano (.txt), además de implementar un menú interactivo para facilitar el uso del sistema.

Explicación de las Funciones

El programa desarrollado cuenta con dos bloques principales de funcionalidad que fueron cuidadosamente diseñados para manejar datos de manera eficiente:

1. Menú del programa

El menú principal del programa es el punto de interacción entre el usuario y el sistema. Se implementó con instrucciones en ensamblador ARM64 para gestionar la navegación y selección de opciones. Las funcionalidades disponibles incluyen:

- **Mostrar los nombres de los integrantes del grupo:** Esta opción imprime en pantalla una lista de los desarrolladores del proyecto. Se utiliza un conjunto de instrucciones de carga (LDR) y almacenamiento (STR) para extraer y mostrar los nombres desde un segmento de memoria predefinido.
- **Análisis estadístico:** La opción más compleja del programa, encargada de calcular:
    - **Media aritmética:** Se suman los valores almacenados en un arreglo y se dividen por el total de elementos. Esto se realiza mediante un bucle iterativo con instrucciones de suma acumulativa (ADD) y división (FDIV).
    - **Moda:** Se utiliza un algoritmo de conteo que incrementa registros específicos para identificar el valor más frecuente.
    - **Valores máximos y mínimos:** La función findMaxMin se encarga de comparar cada elemento del arreglo con el máximo y el mínimo actual, actualizando los registros correspondientes si se encuentra un nuevo extremo.

    - **Rango:** Se calcula como la diferencia entre el valor máximo y el mínimo utilizando una simple instrucción de resta (SUB).

- **Exportar resultados:** Los datos analizados se escriben en un archivo .txt en un formato estructurado. Esto se implementa mediante llamadas al sistema operativo utilizando instrucciones de sistema (SYS) para manejar archivos.
- **Salir del programa:** Termina la ejecución limpiamente, liberando cualquier recurso usado.

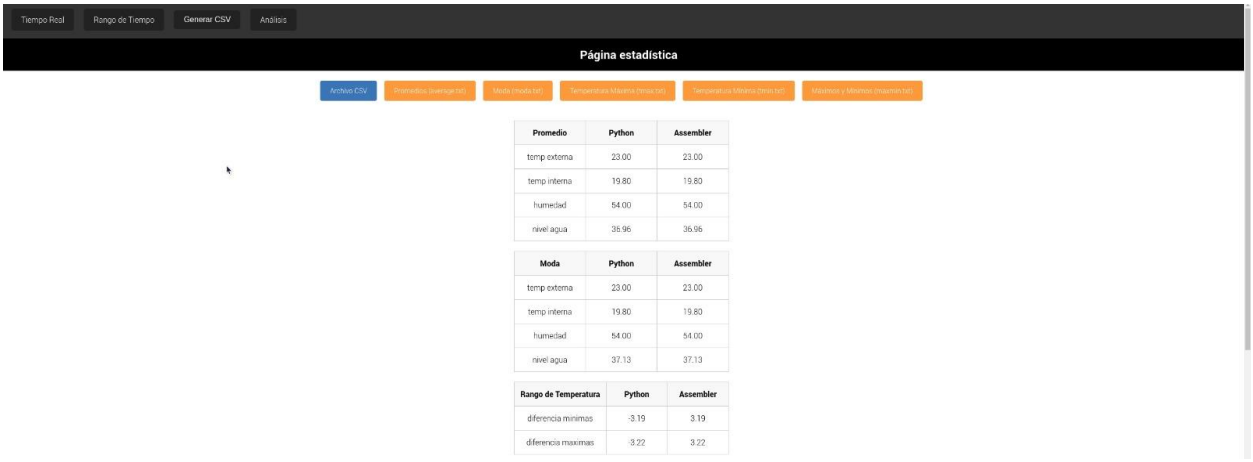
2. Cálculo de valores máximos y mínimos

La función findMaxMin es una de las más importantes, ya que garantiza la identificación precisa de los valores extremos en los datos analizados. El flujo de esta función es el siguiente:

1. **Inicialización:**
    - Se cargan los valores iniciales desde el arreglo en memoria a registros (LDR), estableciendo los primeros valores como el máximo y el mínimo provisional.
2. **Comparación iterativa:**
    - Un bucle recorre cada elemento del arreglo. Utiliza la instrucción FCMP para comparar el valor actual con el máximo provisional. Si el valor actual es mayor, el registro correspondiente se actualiza con una instrucción de almacenamiento (STR).
    - Del mismo modo, se compara cada elemento con el mínimo provisional y se actualiza si el valor es menor.

3. **Almacenamiento del resultado:**
    - Una vez terminado el bucle, los valores máximo y mínimo definitivos se guardan en ubicaciones específicas de memoria para su uso posterior.

Esta implementación garantiza la eficiencia gracias al uso directo de registros y la optimización de ciclos en el ensamblador, aprovechando las capacidades del conjunto de instrucciones ARM64.



Archivo CSV | Promedios (average.txt) | Moda (moda.txt) | Temperatura Máxima (tmax.txt) | Temperatura Mínima (tmin.txt) | Máximos y Mínimos (maxmin.txt)

Promedio	Python	Assembler
temp externa	23.00	23.00
temp interna	19.80	19.80
humedad	54.00	54.00
nivel agua	36.96	36.96

Moda	Python	Assembler
temp externa	23.00	23.00
temp interna	19.80	19.80
humedad	54.00	54.00
nivel agua	37.13	37.13

```
main:
.LFB22:
        .cfi_startproc
        stp     x29, x30, [sp, -112]!
        .cfi_def_cfa_offset 112
        .cfi_offset 29, -112
        .cfi_offset 30, -104
        mov     x29, sp
        stp     x19, x20, [sp, 16]
        .cfi_offset 19, -96
        .cfi_offset 20, -88
        adrp    x19, .LC0
        adrp    x20, .LC6
        add     x19, x19, :lo12:.LC0
        stp     x21, x22, [sp, 32]
        .cfi_offset 21, -80
        .cfi_offset 22, -72
        adrp    x22, .LC4
        adrp    x21, .LC5
        stp     x23, x24, [sp, 48]
        .cfi_offset 23, -64
        .cfi_offset 24, -56
        adrp    x24, .LC2
        adrp    x23, .LC3
        stp     x25, x26, [sp, 64]
        .cfi_offset 25, -48
        .cfi_offset 26, -40
        add     x26, sp, 108
        adrp    x25, .LC1
        stp     x27, x28, [sp, 80]
        .cfi_offset 27, -32
        .cfi_offset 28, -24
        adrp    x28, .LC20
        adrp    x27, .LC22
        add     x28, x28, :lo12:.LC20
        add     x27, x27, :lo12:.LC22
        b       .L9
        .p2align 2,,3
```

```
findMaxMin:
.LFB23:
        .cfi_startproc
        mov     w6, w1
        mov     x1, x4
        cmp     w6, 0
        ble     .L16
        ldr     s1, [x0]
        str     s1, [x2]
        str     s1, [x3]
        cmp     w6, 1
        beq     .L3
        mov     x5, 1
        .p2align 3,,7
.L8:
        ldr     s0, [x0, x5, lsl 2]
        ldr     s1, [x2]
        fcmpe   s0, s1
        bgt     .L9
.L4:
        ldr     s1, [x3]
        fcmpe   s0, s1
        bmi     .L10
.L6:
        add     x5, x5, 1
        cmp     w6, w5
        bgt     .L8
        ldr     s1, [x3]
.L3:
        ldr     s0, [x2]
        fcvt    d1, s1
        adrp    x0, .LC1
        add     x0, x0, :lo12:.LC1
        fcvt    d0, s0
        b       printf
        .p2align 2,,3
.L9:
        str     s0, [x2]
        b       .L4
        .p2align 2,,3
.L10:
        str     s0, [x3]
        b       .L6
        .p2align 2,,3
.L16:
        adrp    x0, .LC0
        add     x0, x0, :lo12:.LC0
        b       printf
        .cfi_endproc
```

```
main:
.LFB22:
        .cfi_startproc
        stp     x29, x30, [sp, -336]!
        .cfi_def_cfa_offset 336
        .cfi_offset 29, -336
        .cfi_offset 30, -328
        adrp    x1, .LC0
        adrp    x0, .LC1
        mov     x29, sp
        add     x1, x1, :lo12:.LC0
        add     x0, x0, :lo12:.LC1
        bl      fopen
        cbz     x0, .L24
        stp     x19, x20, [sp, 16]
        .cfi_offset 20, -312
        .cfi_offset 19, -320
        add     x20, sp, 80
        mov     x2, x0
        mov     w1, 256
        str     x21, [sp, 32]
        .cfi_offset 21, -304
        mov     x21, x0
        mov     x0, x20
        str     d12, [sp, 40]
        .cfi_offset 76, -296
        adrp    x19, .LC3
        stp     d8, d9, [sp, 48]
        .cfi_offset 73, -280
        .cfi_offset 72, -288
        add     x19, x19, :lo12:.LC3
        stp     d10, d11, [sp, 64]
        .cfi_offset 75, -264
        .cfi_offset 74, -272
        bl      fgets
        mov     x0, 61572651155456
        mov     x1, 61572651155456
        movk    x0, 0x408f, lsl 48
        movk    x1, 0xc08f, lsl 48
        fmov    d9, x0
        fmov    d10, x1
        fmov    d11, d9
        fmov    d12, d10
        .p2align 3,,7
```

```
main:
.LFB22:
        .cfi_startproc
        stp     x29, x30, [sp, -384]!
        .cfi_def_cfa_offset 384
        .cfi_offset 29, -384
        .cfi_offset 30, -376
        adrp    x1, .LC0
        adrp    x0, .LC1
        mov     x29, sp
        add     x1, x1, :lo12:.LC0
        add     x0, x0, :lo12:.LC1
        bl      fopen
        cbz     x0, .L16
        adrp    x1, .LC3
        add     x1, x1, :lo12:.LC3
        stp     x23, x24, [sp, 48]
        .cfi_offset 24, -328
        .cfi_offset 23, -336
        mov     x24, x0
        adrp    x0, .LC4
        add     x0, x0, :lo12:.LC4
        stp     x25, x26, [sp, 64]
        .cfi_offset 26, -312
        .cfi_offset 25, -320
        bl      fopen
        mov     x26, x0
        cbz     x0, .L17
        adrp    x0, .LC13
        add     x0, x0, :lo12:.LC13
        add     x25, sp, 96
        stp     x19, x20, [sp, 16]
        .cfi_offset 20, -360
        .cfi_offset 19, -368
        mov     x2, x24
        ld1     {v0.16b - v1.16b}, [x0]
        stp     x21, x22, [sp, 32]
        .cfi_offset 22, -344
        .cfi_offset 21, -352
        add     x22, sp, 128
        stp     d8, d9, [sp, 80]
        .cfi_offset 73, -296
        .cfi_offset 72, -304
        mov     w1, 256
        mov     x0, x22
        st1     {v0.16b - v1.16b}, [x25]
        bl      fgets
        adrp    x21, .LC6
        mov     x2, x24
        add     x21, x21, :lo12:.LC6
        mov     x0, x22
        mov     w23, 0
        mov     w1, 256
        bl      fgets
        cbz     x0, .L18
        .p2align 3,,7
```

Conclusión de los resultados obtenidos

El programa desarrollado demostró ser funcional y eficiente para el análisis estadístico básico de los datos del sistema de riego automatizado. La implementación en ensamblador ARM64 permitió optimizar el uso de recursos y garantizar un rendimiento rápido y confiable. Los resultados obtenidos facilitaron la comprensión del comportamiento del sistema, permitiendo identificar áreas de mejora en la gestión del agua y la temperatura. Además, la exportación de los datos en un archivo .txt brinda flexibilidad para realizar análisis adicionales fuera del sistema.

Enlace del repositorio de GitHub

https://github.com/Josue013/-ACYE1-Proyecto_G3