

# Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores



## Taller de diseño digital

---

### Laboratorio 4

ARMv4

---

**Estudiante**

Soto Varela Óscar

**Carné**

2020092336

**Profesor:**

Luis Alonso Barboza Artavia

21 de mayo de 2025

---

## Laboratorio 4: Código ensamblador: ARMv4

---

Fecha de asignación: 09 de mayo 2025

Fecha de entrega: 21 de mayo 2025

Profesores: Luis Barboza Artavia  
Jeferson González Gómez

---

### 1. Introducción

El código ensamblador es un lenguaje de programación de bajo nivel que representa instrucciones específicas para un procesador. A diferencia de los lenguajes de programación de alto nivel, el código ensamblador está estrechamente vinculado a la arquitectura del procesador y utiliza mnemónicos y códigos de operación para representar las operaciones que la CPU debe realizar. Cada instrucción en código ensamblador se traduce directamente a una instrucción de máquina ejecutable por el procesador.

ARMv4 es una arquitectura de procesador desarrollada por ARM Holdings a principios de la década de 1990, conocida por su eficiencia energética y versatilidad. Introdujo características importantes como Thumb, una tecnología de instrucción de 16 bits para mejorar la densidad de código y el rendimiento en dispositivos con recursos limitados. ARMv4 fue ampliamente adoptada en una variedad de dispositivos móviles y embebidos, sentando las bases para el éxito continuo de ARM en la industria de semiconductores.

En este laboratorio el estudiante resolverá problemas mediante el uso de código ensamblador de ARMv4.

### 2. Investigación

Esta evaluación es **individual**. Para el desarrollo de este laboratorio se deben responder las siguientes preguntas.

1. Investigue los tipos de instrucción que tiene ARMv4. Nombre dos ejemplos de cada tipo y un posible uso.
2. Explique el set de registros que tiene ARM. ¿Cuál es el contenido de cada uno?
3. Explique el funcionamiento del *branch*.
4. ¿Cómo se implementa un condicional (*if/else*) en ARMv4?

5. Explique el procedimiento para transformar el código en ensamblador a binario. Investigue herramientas que realizan el proceso.
6. Explique la diferencia de *little endian* y *big endian*. ¿Cuál es su implicación cuando pasan las instrucciones a lenguaje máquina (binario)?

### 3. Ejercicios Prácticos

A continuación se presentan 3 ejercicios prácticos, los cuales debe resolver de manera completa **individualmente**. Para cada ejercicio debe presentar el código ensamblador en ARMv4, prueba del código ensamblador en herramienta correspondiente (e.g., [VisUAL](#), [CPUlator](#), etc.) y el archivo en lenguaje máquina (binario) equivalente.

#### 3.1. Problema 1

Asuma que se tiene un arreglo con 10 valores y una constante definida por cada estudiante. Realice el código ensamblador (ARMv4) para el siguiente pseudocódigo:

```
int [] array = {...}
int y =

for (i = 0, 1, ..., 10)
    if array[i] >= y
        array[i] = array[i] * y
    else
        array[i] = array[i] + y
```

#### 3.2. Problema 2

Realice el código ensamblador (ARMv4) para calcular factorial de un número  $X$ . Es decisión propia cómo se manejará el número  $X$  en el código. Realice la demostración para 3 valores de  $X$  distintos.

#### 3.3. Problema 3

Suponga que un procesador tiene anexado un contador, que representa la posición en pantalla (VGA) de un *sprite*, así como un teclado que debe controlar dicha posición cuando se recibe la tecla de flecha superior (para aumentar el contador) y la flecha inferior (para disminuirlo). Asuma que el valor de la tecla se encuentra siempre disponible en la dirección de memoria 0x1000, y el

contador se encuentra mapeado en la dirección 0x2000. Los valores en hexadecimal de las teclas son:

Flecha de arriba -> 0xE048

Flecha de abajo -> 0xE050

Con base en lo anterior, realice un programa usando ARMv4 que constantemente lea el valor de la tecla y actualice el contador correspondiente, aumentándolo o disminuyéndolo dependiendo de cuál flecha se presionó. El programa NO debe cambiar el contador en caso de que la tecla presionada no sea una de las dos opciones válidas.

**Nota:** Para efectos de la prueba del código, simule el valor de la tecla, escribiendo en la dirección de memoria correspondiente de manera manual para al menos 5 iteraciones del bucle. Se debe probar las dos teclas válidas, así como al menos una tecla inválida, para asegurarse de que el sistema funciona correctamente.

## 4. Evaluación

La evaluación de este laboratorio será individual, por medio de un informe escrito y repositorio donde se encuentren los códigos fuente (ensamblador y lenguaje máquina) con la solución a los problemas planteados. En el informe escrito deberán presentarse las respuestas de la sección de investigación, así como capturas de pantalla de las pruebas (simulación en herramienta elegida) para cada uno de los ejercicios prácticos.

La entrega se debe realizar por medio del TEC-Digital en la pestaña de evaluación. No se aceptan entregas extemporáneas después de la fecha de entrega a las 11:59 pm como máximo. **Los documentos serán sometidos a control de plagios.**

**Abstract**— This document presents a practical analysis and research on the ARMv4 architecture within the context of the Digital Design Workshop course. It outlines the main types of instructions available in this architecture, including examples and typical use cases. The processor's register set is also described, along with its role in code execution. Further, the report explores the behavior of branch instructions, the implementation of conditional structures, and the procedure for converting assembly code into binary, taking into account the implications of Big Endian and Little Endian storage formats. Finally, practical exercises and digital tools are included to allow experimentation with these instructions in a simulated environment

## I. INTRODUCCIÓN

La arquitectura ARMv4 representa una de las primeras versiones ampliamente adoptadas de la familia de procesadores ARM, caracterizada por su simplicidad, eficiencia energética y conjunto reducido de instrucciones (RISC). Esta arquitectura ha sido empleada en diversos sistemas embebidos, dispositivos móviles y aplicaciones donde el bajo consumo y el rendimiento son factores críticos, lo cual agrega importancia a que los ingenieros en computadores se familiaricen con esta arquitectura.

El presente informe tiene como objetivo analizar las principales características de la arquitectura ARMv4, enfocándose en el conjunto de instrucciones, la estructura de registros, el flujo de control mediante instrucciones de salto y condicionales, y los aspectos relacionados con el almacenamiento en formatos Big Endian y Little Endian. Además, se abordan ejemplos prácticos de codificación en ensamblador y su correspondiente traducción a lenguaje máquina, utilizando herramientas digitales que permiten simular y verificar el comportamiento del procesador.

Este estudio se enmarca en el contexto del Taller de Diseño Digital, proporcionando a los estudiantes una comprensión más profunda del funcionamiento interno de un procesador, de los principios fundamentales de la programación a bajo nivel y sobre la arquitectura ARMv4, conocimientos que serán de suma importancia para realizar el laboratorio 4 del curso.

## II. INVESTIGACIÓN

*II-A. Investigue los tipos de instrucción que tiene ARMv4. Nombre dos ejemplos de cada tipo y un posible uso*

La arquitectura ARM, diseñada bajo el paradigma RISC (Reduced Instruction Set Computing), tradicionalmente organiza su conjunto de instrucciones en 4 distintas categorías básicas según su función, pero en específico ARMv4 reduce las tradicionalmente 4 categorías a solo 3 ya que combina las instrucciones de salto condicional y de salto incondicional en una sola categoría, donde ambos tipos utilizan el mismo formato binario de instrucción, esto permite reducir complejidad y mejorar regularidad [1].

*II-A1. Instrucciones de procesamiento de datos:* Estas instrucciones realizan operaciones aritméticas y lógicas entre registros, sin acceso directo a memoria [2].

- **ADD** (*Add*): Suma valores entre dos registros.
- **AND** (*Logical AND*): Realiza una operación lógica AND bit a bit entre registros.
- Ejemplo de uso:

```
ADD R0, R1, R2
AND R3, R4, R5
```

La instrucción ADD suma los valores de R1 y R2 y almacena el resultado en R0. La instrucción AND realiza una operación AND bit a bit entre R4 y R5, y guarda el resultado en R3.

*II-A2. Instrucciones de carga y almacenamiento:* Permiten transferir datos entre memoria y registros.

- **LDR** (*Load Register*): Carga datos desde memoria a un registro.
- **STR** (*Store Register*): Almacena el contenido de un registro en memoria.
- Ejemplo de uso:

```
LDR R0, [R1]
STR R2, [R3, #4]
```

LDR carga en R0 el contenido de memoria apuntada por R1. STR guarda el valor de R2 en la dirección calculada como R3 + 4.

*II-A3. Instrucciones de salto y control de flujo:* Permiten alterar el flujo de ejecución mediante saltos condicionales o incondicionales.

- **B (Branch):** Salta incondicionalmente a una etiqueta.
- **BEQ (Branch if Equal):** Salta si la condición de igualdad se cumple (flag Z activa).
- Ejemplo de uso:

```
B end_loop
BEQ equal_label
```

B transfiere el control a *end\_loop* incondicionalmente. BEQ transfiere el control a *equal\_label* solo si el resultado anterior fue cero.

*II-B. Explique el set de registros que tiene ARM. ¿Cuál es el contenido de cada uno?*

La arquitectura ARMv4 cuenta con 17 registros visibles al programador, todos de 32 bits, y se pueden dividir en registros generales y registros especiales [1].

Los registros de R0 hasta R12 son de propósito general, pueden ser utilizados por los programadores para almacenar información.

La ISA ARMv4 recomienda el siguiente uso para los registros. Esta recomendación suele ser seguida por los compiladores que convierten el código fuente de lenguajes de alto nivel a lenguaje ensamblador.

Para los registros de propósito general (R0-12) se tiene la siguiente distribución.

- **R0-R3:** Paso de parámetros a funciones y retorno de valores.
- **R4-R11:** Almacenamiento de variables locales (preservadas entre llamadas a funciones).
- **R12:** Registro temporal (variable temporal)

Los siguientes registros R13, R14, R15 y el registro de estado actual del programa (CPSR) forman parte del circuito de control utilizado para la secuenciación del programa. Hay que tener especial cuidado con estos registros de propósito especial para evitar cambios incorrectos en sus valores almacenados, ya que podrían causar un comportamiento inesperado del programa. A continuación se detalla el contenido de cada uno de estos registros especiales:

- **R14 (SP - Stack Pointer):** Contiene la dirección de la cima de la pila
- **R15 (LR - Link Register):** Guarda la dirección de retorno cuando se llama una subrutina, permite volver al punto de llamada después de una función.
- **R15 (PC - Program Counter):** Contiene la dirección de la siguiente instrucción a ejecutar. Se incrementa automáticamente a medida que se ejecuta el programa.
- **CPSR (Current Program Status Register):** Contiene banderas de estado (como Z, N, C, V) que reflejan los resultados de las operaciones aritméticas. También incluye bits de control del modo del procesador y habilitación de interrupciones.

En la figura 1 se muestra una imagen que resume el set de registros de ARMv4:

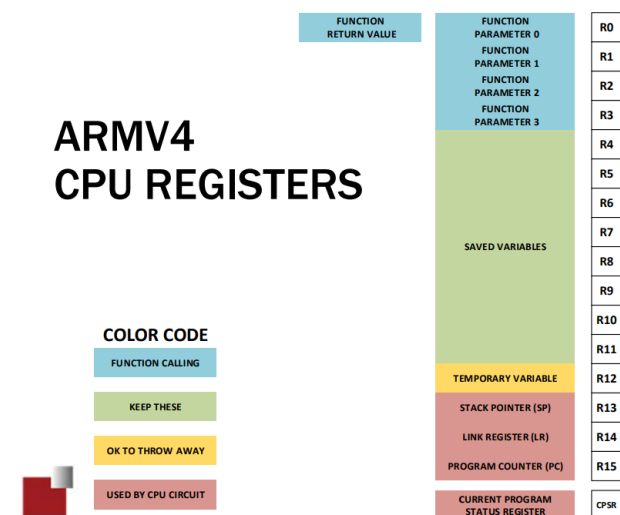


Figura 1. Set de Registros ARMv4 [1]

*II-C. Explique el funcionamiento del branch*

El branch utiliza para cambiar el flujo de ejecución del programa, alterando directamente el valor del contador de programa (PC, que es el registro R15). Esto permite implementar estructuras como bucles, condicionales y llamadas a funciones. También se tiene BL (branch with link) que lla a una subrutina guardando la dirección de retorno en el Link Register (LR, que es R14) ó de forma condicional con BEQ label que salta si el resultado anterior fue cero, por mencionar algunas.

## II-D. ¿Cómo se implementa un condicional (if/else) en ARMv4?

En ARMv4, las estructuras condicionales como if/else se implementan mediante instrucciones de comparación (CMP) seguidas de instrucciones de salto condicional como BEQ (Branch if Equal). La instrucción CMP compara dos registros y actualiza las banderas del registro CPSR, que luego son evaluadas por la instrucción de salto.

Una forma de estructurar un if/else es la siguiente:

1. Se comparan los valores con CMP.
2. Mediante BEQ, si la condición del if se cumple, se salta directamente al bloque if.
3. Si la condición no se cumple, se ejecuta el bloque else que sigue inmediatamente después.
4. Al final del bloque else, se incluye un salto incondicional (B) para evitar que se ejecute el bloque if.

A continuación se muestra un código de ejemplo:

```
.global _start
_start:

    MOV R0, #1    // a = 1
    MOV R1, #2    // b = 2

    CMP R0, R1    // compara a y b
    BEQ if

else:
    MOV R2, #0    // R2 = 0
    B end_if_else

if:
    MOV R2, #1    // R2 = 1

end_if_else:
// sigue el programa
```

Se puede copiar y pegar directamente el código anterior con ARM7 basic en CPU emulador si lo desea. Enlace al sitio: <https://cpulador.01xz.net/?sys=arm>.

## II-E. Explique el procedimiento para transformar el código en ensamblador a binario. Investigue herramientas que realizan el proceso

En caso de que el código no tenga saltos se puede utilizar la herramienta online que se tiene en el siguiente enlace <https://shell-storm.org/online/Online-Assembler-and-Disassembler/>.

En esta página primero hay que seleccionar las opciones de ARM y hex. Una vez hecho esto basta copiar el código y obtener el código binario. Es importante destacar que el código no puede tener saltos ni comentarios.

A en la figura 2 se muestra un pequeño ejemplo:

### Online Assembler and Disassembler

by @Jonathan Salwan using Keystone and Capstone projects.

global \_start  
\_start:  
MOV R0, #1  
MOV R1, #2

ARM ARM (thumb) AArch64 Mips (32) Mips (64) PowerPC (32) PowerPC (64)  
Sparc x86 (16) x86 (32) x86 (64)

Inline Python Hex C-Array

Assemble

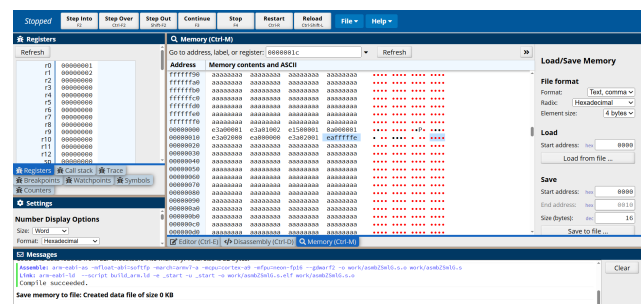
Assembly - Little Endian  
01 00 a0 e3 02 10 a0 e3

Assembly - Big Endian  
e3 a0 00 01 e3 a0 10 02

Figura 2. Código ARM a binario.

El resultado sería el Big Endian.

Otra manera sería usando el mismo CPUlator, hay que fijarse en la memoria al inicio y al final del programa de la siguiente manera para descargar el programa de forma binaria como se muestra en la figura 3:





**II-F. Explique la diferencia de little endian y big endian. ¿Cuál es su implicación cuando pasan las instrucciones a lenguaje máquina (binario)?**

La diferencia entre little endian y big endian se refiere a cómo se almacenan los bytes de un número multi-byte (como un entero de 32 bits) en la memoria de una computadora. Para Big Endian el byte más significativo (MSB) se guarda primero (en la dirección de memoria más baja) mientras que en el Little Endian el byte menos significativo (LSB) se guarda primero (en la dirección de memoria más baja).

### III. EJERCICIOS PRÁCTICOS

Puede acceder al código fuente de los ejercicios prácticos y sus binarios mediante el siguiente enlace de repositorio de Github:

[https://github.com/CAMANEM/osoto\\_digital\\_design\\_lab\\_2025/tree/main](https://github.com/CAMANEM/osoto_digital_design_lab_2025/tree/main)

#### III-A. Problema 1

En la siguiente figura 4 se puede observar que se guarda  $R1 = 2$  como valor para realizar las comparaciones. Además, el primer valor de la lista (1) se le suma 2 por ser menor que  $R1$ .

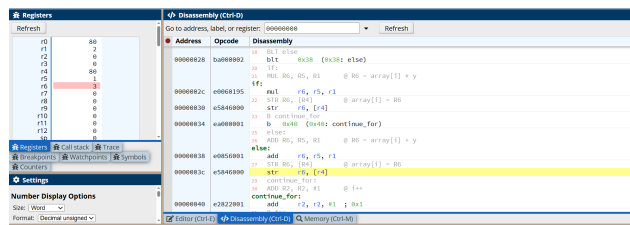


Figura 4. Se muestra como  $i[0]$  pasa a ser 3, ya que  $i \geq 2$ ;  $i[0] += 2$

Mientras que en la figura 5 dado que  $(i[2] == 3)$   $\wedge$   $(r1 == 2)$  se tiene que entonces se debe realizar la multiplicación cuyo resultado da 6.

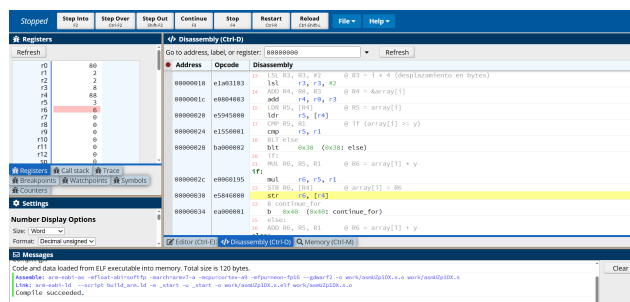


Figura 5. Para  $i[2]$   $\geq 2$  tiene un resultado de multiplicación de  $3 * 2 = 6$  que se guarda en  $r6$

Para el resto de números de la lista se siguen realizando la suma o multiplicación según corresponde de la misma manera que como se mostró en las anteriores imágenes. Has que llega a  $i == 9$  que es el último elemento de la lista y el código se detiene después de realizar la última operación de suma o resta como se muestra en la figura 6.

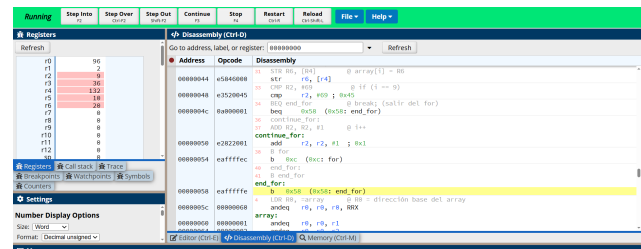


Figura 6. Para  $i[9]$   $\geq 2$  tiene un resultado de multiplicación de  $10 * 2 = 20$  que se guarda en  $r6$ , y se termina la ejecución por terminar la lista de 10 elementos

#### III-B. Problema 2

En las figuras 7 y 8 se evidencia el cálculo para el factorial de 5.

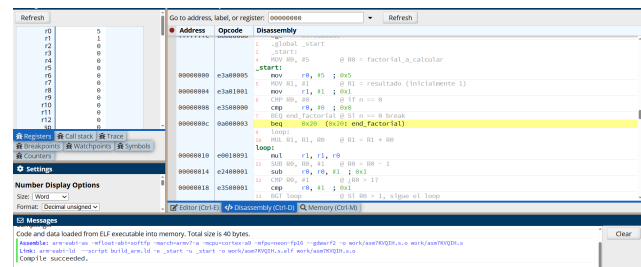


Figura 7. Inicio de factorial de 5

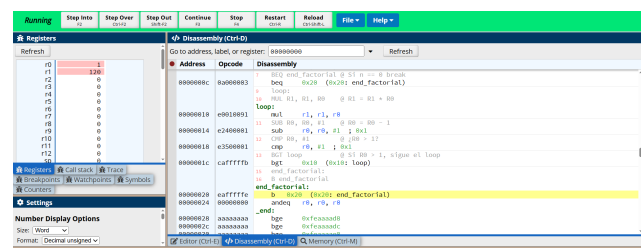


Figura 8. Resultado de factorial de 5

En las figuras 9 y 10 se evidencia el cálculo para el factorial de 12.



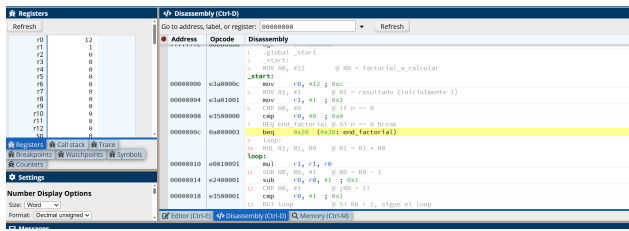


Figura 9. Inicio de factorial de 12

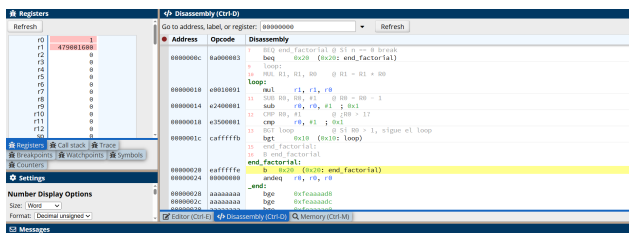


Figura 10. Resultado de factorial de 12

En las figuras 11 y 12 se evidencia el cálculo para el factorial de 7.

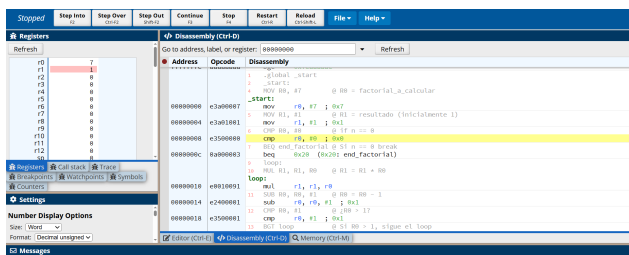


Figura 11. Inicio de factorial de 7

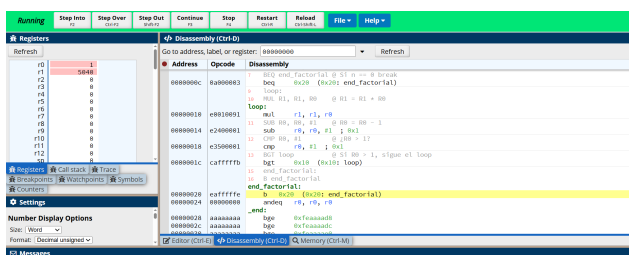


Figura 12. Resultado de factorial de 7

### III-C. Problema 3

En la figura 13 se observa cuando se presiona tecla arriba y se suma 1 a la posición del sprite, resultando en 1. En figura 14 se presiona la tecla abajo y se pasa de 1 a 0. Mientras que en la figura 15 se presiona una tecla inválida y no se suma ni se resta el valor de la posición y se continúa con el ciclo.

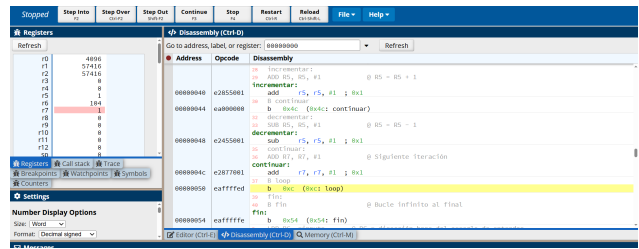


Figura 13. Se presiona tecla arriba y la posición del sprite pasa a ser 1

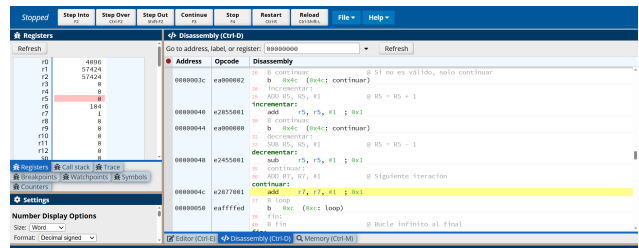


Figura 14. Se presiona tecla abajo y la posición del sprite pasa a ser 0

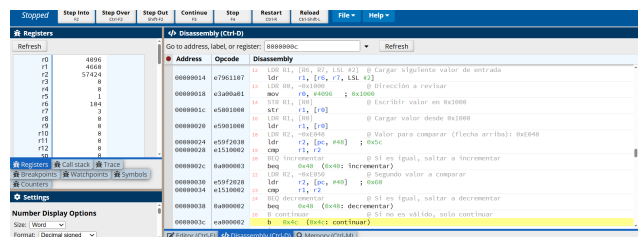


Figura 15. Se presiona una tecla inválida por lo que no se suma ni se resta y el ciclo continúa

## REFERENCIAS

- [1] Russ Meier. Armv4 isa instructions. <https://faculty-web.msoe.edu/meier/ce1921/slidesets/isaarm-instructions.pdf>, 2020. CE1921 Course Material.
- [2] David Seal. *ARM architecture reference manual*. Pearson Education, 2001.
- [3] David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, 2nd edition, 2012.