Laboratorio 4

Karina Martínez Guerrero - 2017102001

May 21, 2025

1 Investigación

1.1 Tipos de instrucciones de ARMv4

1.1.1 Instrucciones de Salto

Estas instrucciones pueden cambiar el orden de flujo del programa simplemente escribiendo en el registro PC [1].

- BL: esta instrucción antes de realizar el salto guarda la posición de la instrucción precedente a la instrucción de salto, para poder retornar desde donde fue llamada.
- BEQ label: salto condicional a label si la bandera de cero es uno, indicando que los numero comparados son iguales.

Se pueden usar para crear saltos a funciones, y regresar al flujo principal después de ejecutar la función

1.1.2 Instrucciones de Procesamiento de Datos

Las instrucciones de procesamiento de datos, realizan cálculos en los registros de propósitos generales.

Se dividen en tres categorías: Aritméticas / Lógicas, de Comparación y de Multiplicación [2].

- ADD: suma el primero operador con el segundo, el resultado es guardado en Rd.

- CMP: compara dos registros, esta comparación es una resta cuyo resultado modifica las banderas, de esta forma es posible detectar si un numero es mayo a otro (N=1) si dos números son iguales (Z=1), etc.

1.1.3 Instrucciones de Transferencias del registro Status

El registro actual (Current Program Status Register CPSR), contiene las banderas de condición, los bits de de manejo de las interrupciones, el modo del procesador, etc. Cada excepción posee además un Saved Program Status Register (SPSR), el cual es utilizado para preservar el valor del CPSR cuando ocurre una excepción [2].

- MRS: copia un el CPSR o SPSR a un registro. Cuando una excepción ocurre, y es posible que una próxima excepción del mismo tipo ocurra, el SPSR corre peligro de ser corrompido, para evitar esto, el SPSR debe ser guardado inmediatamente cuando entramos en la excepción y recuperado antes de salir de la misma.

- MSR: copia el valor de un registro o un valor literal a alguno de los PSR (CPSR o SPSR), la modificación de los bits dentro de ese registro, dependerá del modo en que este funcionando el procesador en el momento de la copia, y de la máscara que se indique en la instrucción.

1.1.4 Instrucciones para Carga y Escritura en Memoria

Las Instrucciones de carga y almacenamiento, realizan transferencias de datos entre los registros y la memoria [2].

- LDR: carga un valor de la memoria de tamaño word o byte y lo escribe en un registro.

- LDM: cargar múltiples registros(un subset o todos ellos) desde la memoria.

1.1.5 Instrucciones para la Generación de Excepciones

Existen dos tipos de instrucciones para generar excepciones [2].

- SWI: Estas instrucciones causan que ocurra una excepción por interrupción de software, normalmente para realizar llamadas al sistema operativo.

- BKPT: genera una excepción de aborto. En el caso de tener un debugger instalado en el vector de aborto, una excepción generada de esta forma es tratada como un breakpoint.

1.2 Set de registros de ARMv4

La arquitectura ARM utiliza 16 registros. Los registros R0–R12 se utilizan para guardar variables mientras que los registros R13-R15 tienen usos especiales [3]. A continuación la tabla 1 resume las funciones de los registros.

Table 1: Conjunto de registros de ARM

Nombre	**Uso**
R0	Argumento / Valor de retorno / Variable temporal
R1 - R3	Argumento / Variable temporal
R4 - R11	Variables preservadas
R12	Variable temporal
R13 (SP)	Puntero a la pila de memoria
R14 (LR)	Puntero de link
R15 (PC)	Contador de programa

1.3 Funcionamiento de branch en ARMv4

ARM usa instrucciones de branch o rama para saltar secciones de código o repetir la ejecución de código, por lo que posibilita la implementación de bucles o estructuras como if/else.

Los programas generalmente se ejecutan en secuencia, y el contador de programa (PC) incrementa en 4 cada vez para apuntar a la siguiente instrucción. Las instrucciones de branch cambian el contador de programa y por tanto el orden de ejecución del código [3].

Existen dos tipos de instrucciones de branch en ARM: la rama simple (B) y la rama con link (BL). La última guarda en LR la dirección a la que debe retornar luego de finalizar la ejecución del código de la rama.

1.4 Implementación de la condición if/else en ARMv4

Como se mencionó en la sección anterior, la condición if/else se puede implementar con el uso de ramas, para saltar cierto bloque de código basado

en condiciones. El código a continuación tomado de [3] demuestra la implementación de un if/else con branch.

```
CMP R0, R1 ; R0 == R1?
BNE L1 ; if not equal, skip if block
ADD R2, R3, #1 ; if block: f = i + 1
B L2 ; skip else block
L1
    SUB R2, R2, R3 ; else block: f = f - i
L2
```

1.5 Procedimiento para transformar el código en ensamblador a binario

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

Para la transformación de código ensamblador a lenguaje máquina, se definen tres tipos de formatos de instrucción: de procesamiento de datos, de memoria y de rama o branch.

Cada tipo de instrucción define cómo organizar los bits dentro de una palabra, e incluye un identificador del tipo de instrucción en los bits 26 y 27. El identificador puede ser 00 para instrucciones de procesamiento de datos, 01 para instrucciones de memoria y finalmente 10 para instrucciones de branch. La Fig 1 muestra la organización de bits para una instrucción de procesamiento de datos.

Cada operación tiene definida su código en binario para los campos, además del campo condicional, que tiene un valor de 1110 para ejecución incondicional y

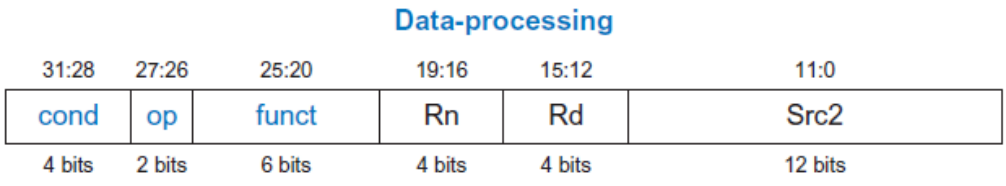

Figure 1: Bits en una instrucción de procesamiento de datos [3]

1.5.1 Instrucciones de procesamiento de datos

Una instrucción de 32-bits tiene seis campos: cond, op, funct, Rn, Rd, y Src2. La operación que se realiza por instrucción está representada por los campos op, funct y cond, que son el código de operación, el código de función y si la operación se ejecuta de manera condicional, respectivamente.

Los campos Rn, Rd y Src2 son los operadores de la instrucción.

1.5.2 Instrucciones de memoria

Las instrucciones de memoria poseen los mismos campos que las instrucciones de procesamiento de datos.

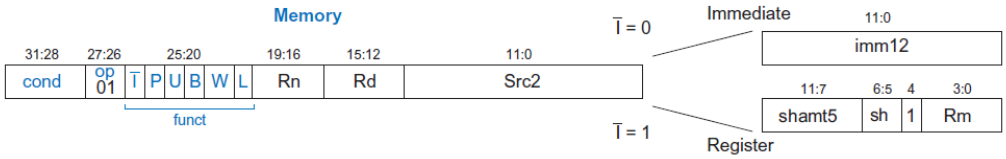

Figure 2: Bits en una instrucción de memoria [3]

1.5.3 Instrucciones de branch

Las instrucciones de branch solo tienen un operador, una constante o inmediato de 24 bits.

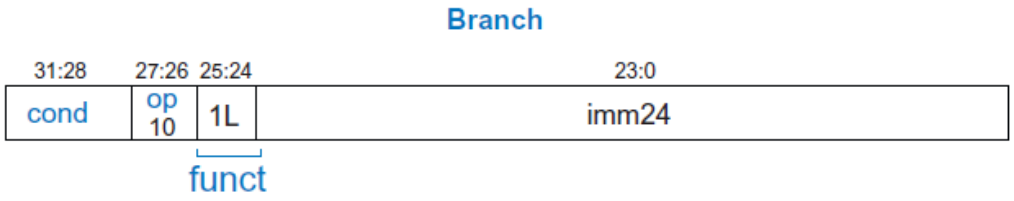

Figure 3: Bits en una instrucción de branch [3]

1.5.4 Herramientas usadas para la conversión

Para convertir el código ensamblador en lenguaje máquina existen assemblers o ensambladores, que realizan esta tarea y colocan el código en un archivo objeto. En general los compiladores realizan esta tarea tras compilar el código en alto nivel y tras el ensamblado, realizan el enlace, que combina el código máquina con otras librerías y archivos para producir un programa ejecutable completo [3].

El ensamblador realiza dos pasadas sobre el código ensamblador. En la primera pasada, el ensamblador asigna las direcciones de las instrucciones y encuentra todos los simbolos como etiquetas y nombres de variables globales. Estos nombres y direcciones se almacenan en una tabla de símbolos. En la segunda pasada, el ensamblador produce el código máquina y toma las direcciones de la tabla.

2 Ejercicios Prácticos

El código completo para la resolución de cada problema se encuentra en el repositorio de Github

2.1 Problema 1

Conversión a ensamblador del 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
```

Resultado de la simulación del código

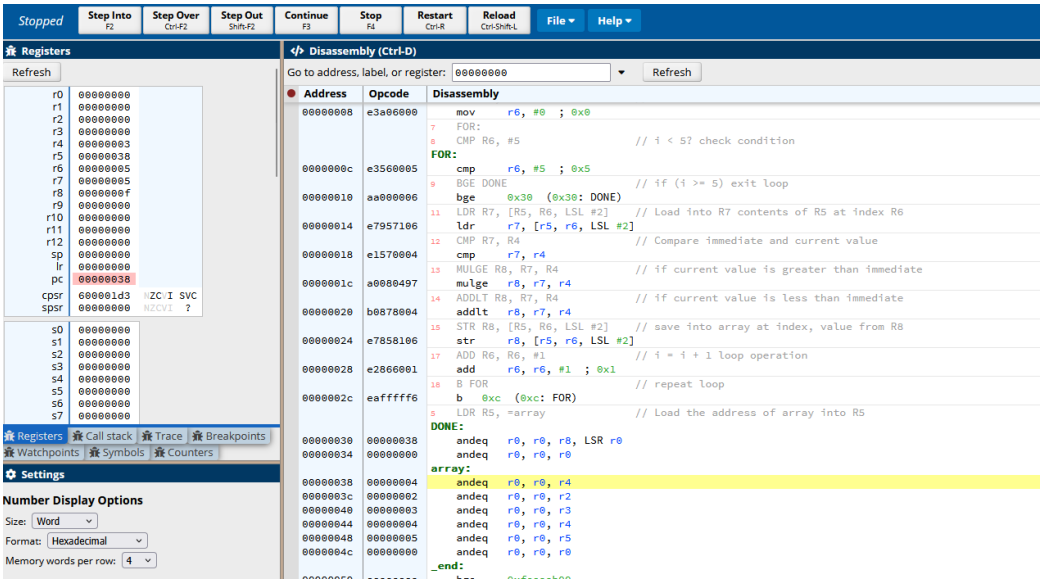

Figure 4: Fin de la ejecución del código

2.2 Problema 2

Código en ensamblador para calcular el factorial de un número

El registro R0 muestra el resultado en hexadecimal 6.

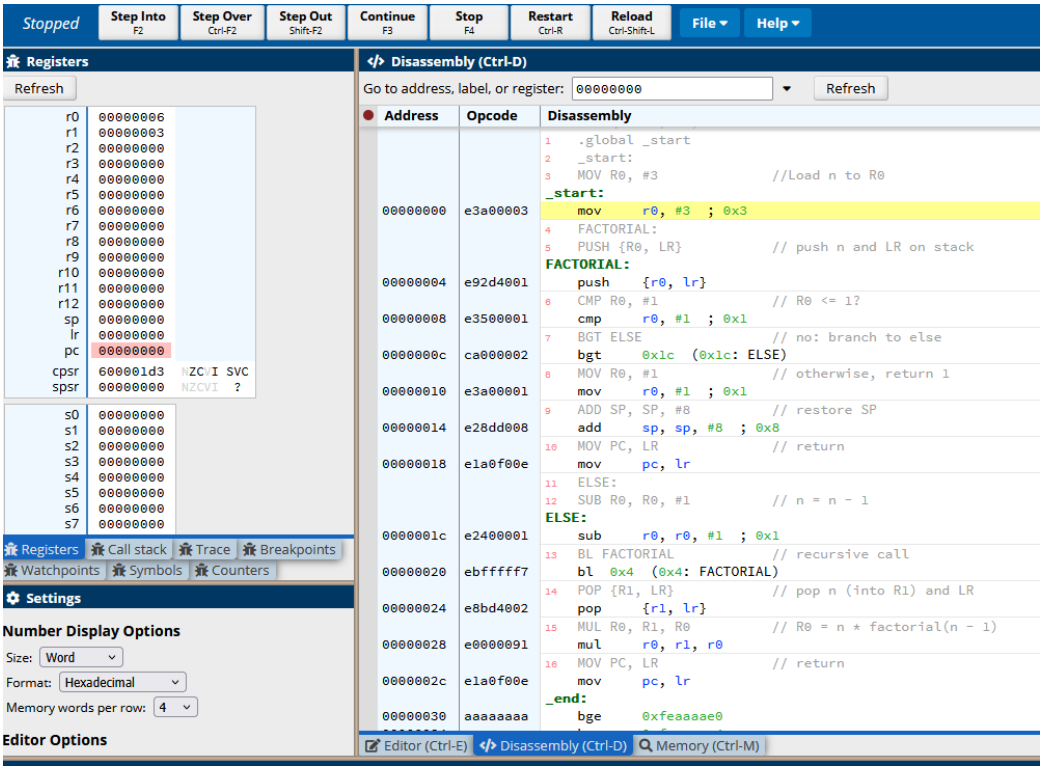

Figure 5: Fin de la ejecución del código del 3!

El registro R0 muestra el resultado en hexadecimal 78, que en decimal es 120.

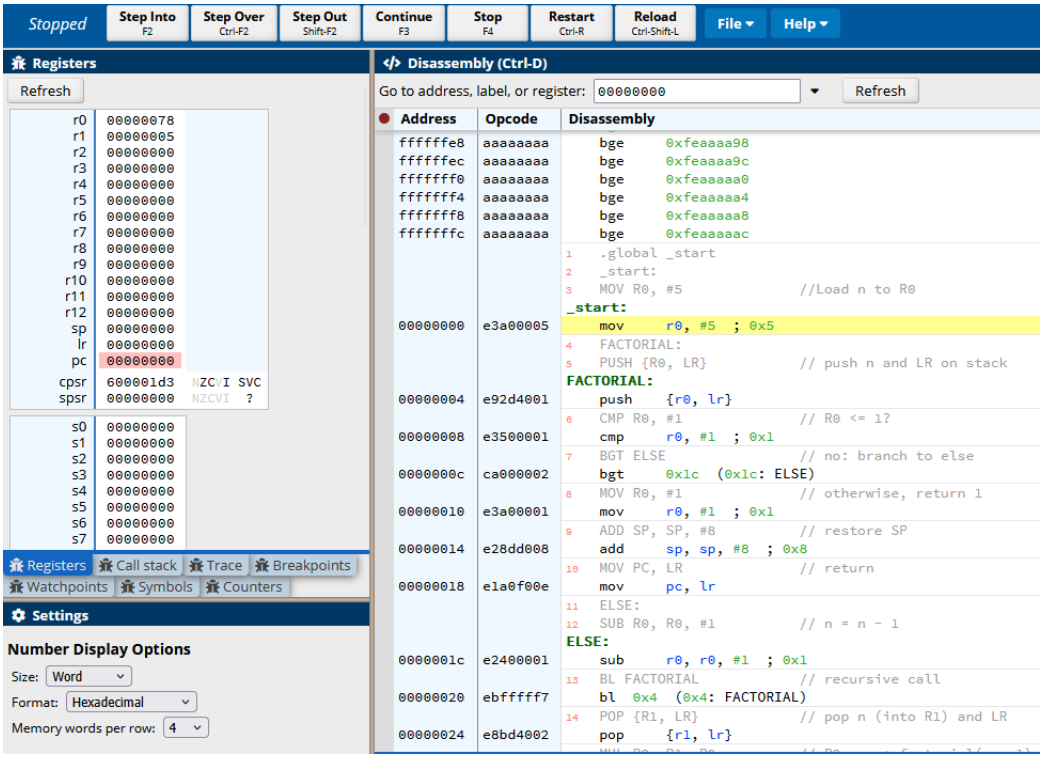

Figure 6: Fin de la ejecución del código del 5!

El registro R0 muestra el resultado en hexadecimal 13B0, que en decimal es 5040.

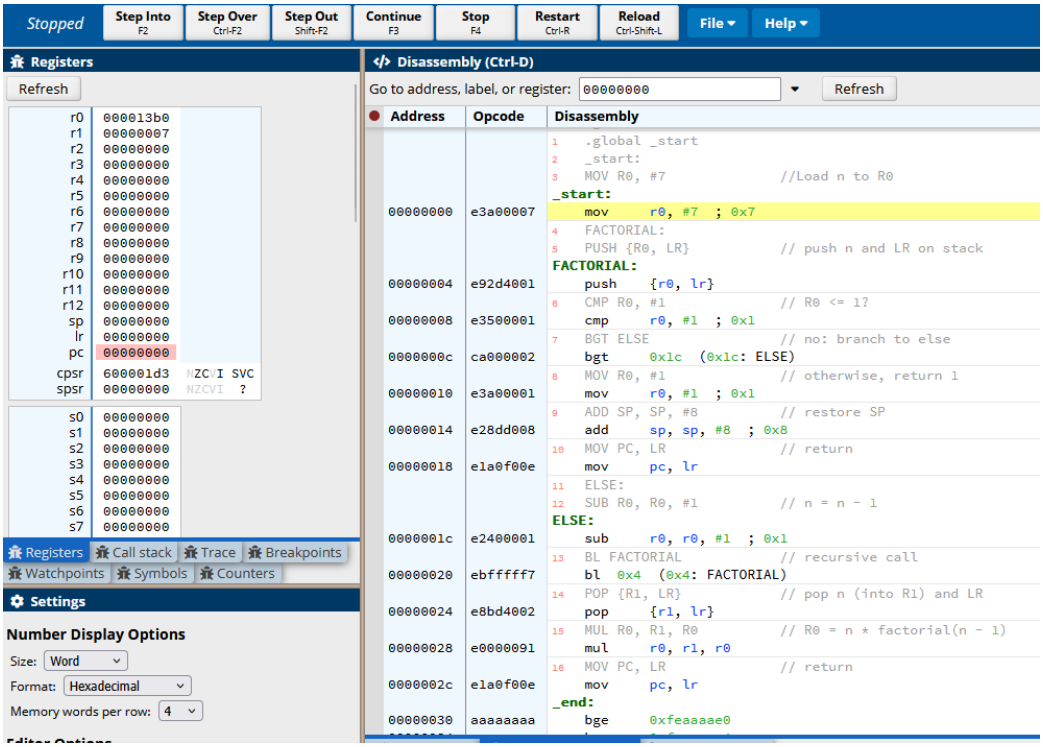

Figure 7: Fin de la ejecución del código del 7!

2.3 Problema 3

Se muestra el final de la ejecución del código para la secuencia de teclas: 0xe048, 0xe048, 0xe050, 0xaaaa, 0xe048. En R0 se guarda el último valor del contador escrito a memoria.

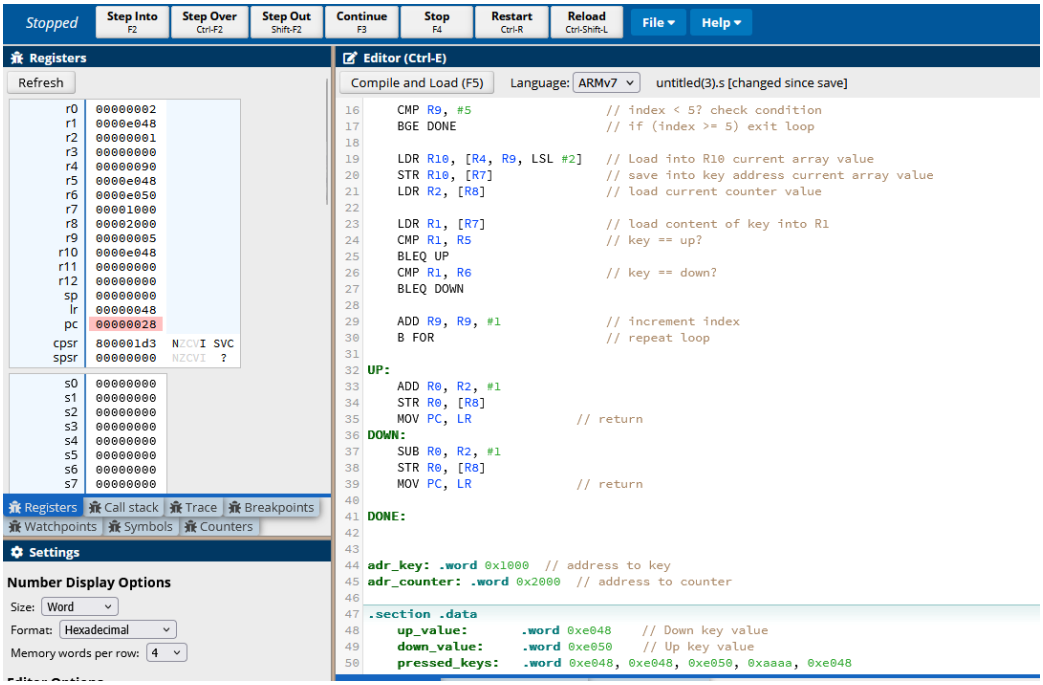

Figure 8: Fin de la ejecución del código del contador

References

[1] G. Steiner, "WebHome/ASMIntroduccionARM/ASMSaltoARM - Técnicas Digitales II Wiki," Utn.edu.ar, Jun. 29, 2011. https://ciii.frc.utn.edu.ar/TecnicasDigitalesII/WebHome/ASMIntroduccionARM/ASMSaltoARM (accessed May 18, 2025).

[2]]G. Steiner, "WebHome/ASMIntroduccionARM - Técnicas Digitales II Wiki," Utn.edu.ar, Jun. 29, 2011. https://ciii.frc.utn.edu.ar/TecnicasDigitalesII/WebHome/ASMIntroduccionARM (accessed May 17, 2025).

[3] S. L. Harris and David Money Harris, Digital design and computer architecture, 1st ed. Amsterdam ; Paris: Elsevier, Cop, 2015, pp. 173–177. Accessed: Mar. 01, 2025. [Online]. Available: https://dl.acm.org/doi/10.5555/2815529