



Gabriel Valle Carrasco
Katya Vanessa Torres Robles
Arquitectura de microcontroladores
Practica 1
11/02/2025

Objetivo

Desarrollar las habilidades en programación ASM.

Práctica

Escribir en ASM para OKDO/LPC55S69 las siguientes funciones:

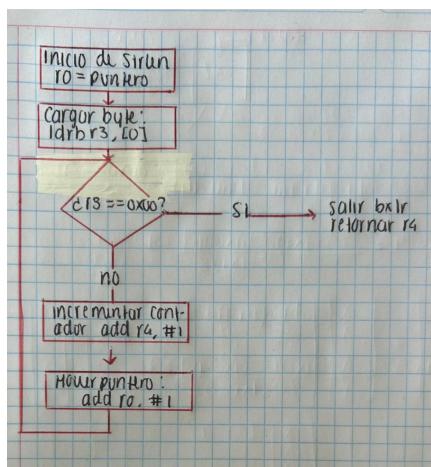
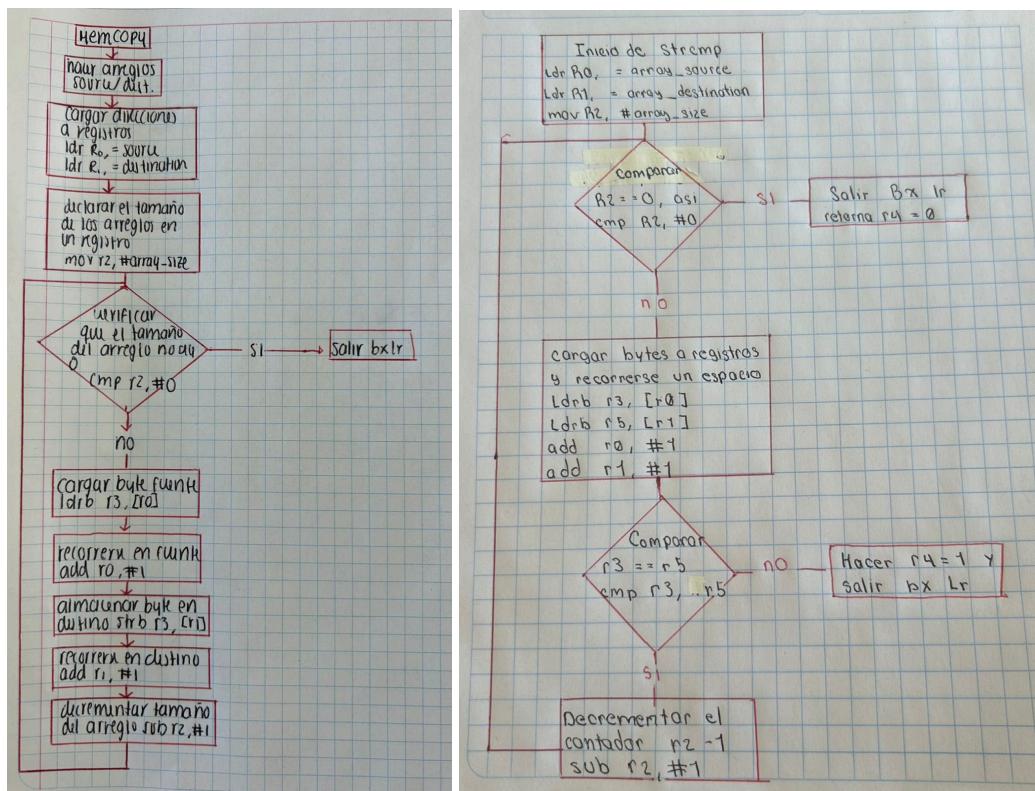
- **Memcpy**: copia datos del apuntador fuente al apuntador destino. Esta función recibe 3 parámetros; 2 apuntadores fuente y destino, y la cantidad de datos a copiar.
- **strcmp**: Compara dos cadenas de caracteres, regresa un 0 si son iguales, un 1 si son diferentes. La función recibe dos parámetros, que son los apuntadores de las cadenas a comparar. Regresa el resultado de la operación.
- **strlen**: Calcula el tamaño de una cadena de caracteres (sin incluir el carácter nulo). Recibe de parámetro un apuntador. Regresa la cantidad de caracteres.

Desarrollo

Al inicio no sabíamos como empezar la práctica hasta que hicimos los diagramas de flujo para comprender la lógica de las funciones. Después de esto le pedimos ayuda a una IA para darnos una idea de como comenzar el código, ya entendiendo lo que está pasando ajustamos el código inicial para hacer el resto de las rutinas.

```
7 main:|
8     push {r3, lr}
9     add r3, sp, #4
0
1     /* Cargar parámetros para MemCopy */
2     ldr r0, =array_source      /* r0 = puntero fuente */
3     ldr r1, =array_destination /* r1 = puntero destino */
4     mov r2, #array_size        /* r2 = tamaño a copiar (4 bytes) */
5     mov r4, #array_flag         /* r4 = 0x0, la usamos como bandera */
6
7     //bl MemCopy              /* llamar a MemCopy */
8     //bl Strcmp                /* llamar a Strcmp */
9     bl Strlen                 /* llamar a Strlen */
0
1     mov r0, #0                  /* valor de retorno 0 */
2     pop {r3, pc}               /* restaurar registros y retornar */
3
4     /* your code goes here */
5
```

Primero declaramos las direcciones de los datos con los que vamos a trabajar.



46 MemCopy:

```

47 /*Copia datos del apuntador fuente al apuntador destino. Esta función recibe 3
48 parámetros; 2 apuntadores, fuente y destino, y la cantidad de datos a copiar.*/
49     cmp r2, #0          /* verificar si el tamaño es cero */
50     beq out            /* si es cero, salir BRANCH IF EQUAL OUT */
51     ldrb r3, [r0]        /* Cargar byte desde fuente */
52     add r0, #1          /* Incrementar puntero fuente */
53     strb r3, [r1]        /* Almacenar byte en destino */
54     add r1, #1          /* Incrementar puntero destino */
55     sub r2, #1          /* Decrementar contador */
56     b MemCopy          /* repetir si no es cero BRANCH IF NOT EQUAL loop */
57
58 out:
59     bx lr

```

Captura de la función MemCopy

```

63 Strcmp:
64 /* Compara dos cadenas de caracteres, regresa un 0 si son iguales, un 1 si son diferentes.
65 La función recibe dos parámetros, que son los apuntadores de las cadenas a comparar. Regresa
66 el resultado de la operación.*/
67     cmp r2, #0           /* compara el registro dos con 0*/
68     beq loopsalida      /* BRANCH IF EQUAL a loopsalida */
69     ldrb r3, [r0]        /* carga byte del arreglo fuente */
70     add r0, #1          /* se mueve al siguiente byte del arreglo fuente*/
71     ldrb r5, [r1]        /* carga byte del arreglo destino*/
72     add r1, #1          /* se mueve al siguiente byte del arreglo destino*/
73     cmp r3, r5          /* compara los bytes de los arreglos fuente y destino*/
74     bne not_equal       /* si no son iguales| salir BRANCH IF NOT EQUAL not_equal*/
75     sub r2, #1          /* Decrementa tamaño de tamaño de arreglo*/
76     b Strcmp            /* repite loop */
77
78 not_equal:
79     add r4, #1          /* Hace al registro 4 = 1*/
80     b loopsalida        /* SALIR */
81
82 loopsalida:
83     bx lr              /* SALIR */
84

```

Captura de la función Strcmp.

```

85 Strlen:
86 /* Calcula el tamaño de una cadena de caracteres (sin incluir el carácter nulo). Recibe de
87 parámetro un apuntador. Regresa la cantidad de caracteres. */
88     ldrb r3, [r0]        /* Carga en el registro 3 el primer objeto del arreglo */
89     cmp r3, #0x00        /* Compara el registro 3 con 0 */
90     beq loopout         /* BRANCH IF EQUAL => loopout */
91     add r4, #1          /* Suma 1 al contador */
92     add r0, #1          /* Se mueve de posición en el arreglo */
93     b Strlen            /* Repite loop */
94
95 loopout:
96     bx lr              /* SALIR */
97
98

```

Captura de la función Strlen.

Conclusiones

Katya:

Esta práctica me ayudó a comprender mejor el ensamblador, y la lógica para cumplir cada rutina que se pidió, aun así me sigue costando un poco de trabajo y requerir ayuda de Gabriel y de la IA para entender el por qué de la estructura, pero me siento más lista para la próxima práctica.

Gabriel:

Antes de la práctica tenía una idea muy vaga de cómo funcionaba todo el proceso de hacer una rutina en ensamblador. Durante la primera función, visualizando los registros y la memoria, fue que entendí cómo estaba funcionando todo. Después de la primera función las siguientes dos fueron más fáciles de desarrollar, gracias a la práctica comprendo mejor la arquitectura y funcionamiento del microcontrolador.