Práctica de Laboratorio 1

Wendell Polo C.I 31648447 and Santiago Blanca C.I 33148530

Universidad de Carabobo

Facultad de Ciencias y Tecnología
Departamento de Computación

24 de febrero de 2026

Objetivo

Familiarizar al estudiante con la sintaxis del lenguaje ensamblador MIPS32 y su ejecución en el entorno MARS, resolviendo problemas clásicos de programación. Se presta especial atención a la gestión de registros, llamadas a funciones, recursividad, estructuras de control, manejo de la pila y configuración del entorno de trabajo.

1 ¿Cómo se implementa la recursividad en MIPS32? ¿Qué papel cumple la pila ($sp)?

La recursividad en MIPS32 se implementa mediante el uso explícito de la pila, administrada a través del registro $sp (stack pointer). Cada vez que una función recursiva es llamada, es necesario reservar espacio en la pila para almacenar los registros que podrían ser modificados durante la ejecución, tales como $ra (return address) y cualquier registro temporal o guardado que esté en uso.

Antes de realizar una nueva llamada recursiva, se disminuye el valor de $sp para reservar espacio en memoria y se guardan los registros necesarios. Al finalizar la ejecución de la función, los registros se restauran desde la pila y se incrementa nuevamente $sp para liberar el espacio reservado. Finalmente, la instrucción `jr $ra` permite regresar correctamente al punto desde donde fue llamada la función.

2 ¿Qué riesgos de desbordamiento existen? ¿Cómo mitigarlos?

Desbordamiento de pila

El desbordamiento de pila ocurre cuando se realizan llamadas recursivas excesivas que agotan el espacio de memoria reservado para la pila. Esto puede provocar el bloqueo del programa o compor-

tamientos impredecibles. Generalmente sucede cuando no existe un caso base adecuado o cuando la profundidad de la recursión es demasiado grande.

¿Cómo mitigarlo?

Para evitar el desbordamiento de pila se recomienda:

- Definir un caso base sólido que garantice la finalización de la recursión.
- Verificar cuidadosamente los casos de borde.
- Reservar y liberar correctamente el espacio en la pila $sp.
- Evaluar si el problema puede resolverse mediante una solución iterativa cuando la profundidad de recursión sea considerable.

3 ¿Qué diferencias encontraste entre una implementación iterativa y una recursiva en cuanto al uso de memoria y registros?

En el caso del algoritmo de paridad, la implementación iterativa resulta más directa y ocupa menos espacio en memoria durante su ejecución. Además, utiliza menos registros, ya que no requiere almacenar información adicional en la pila por cada llamada.

Por otro lado, la implementación recursiva necesita reservar espacio en la pila en cada invocación de la función, lo que incrementa el consumo de memoria y exige una administración más cuidadosa de los registros. Aunque puede resultar más elegante desde el punto de vista conceptual, su costo en memoria es mayor en comparación con la versión iterativa.

4 ¿Qué diferencias encontraste entre los ejemplos académicos del libro y un ejercicio completo y operativo en MIPS32?

Los ejemplos del libro muestran casos aislados de el funcionamiento de instrucciones y de la pila, omitiendo a veces la inicializacion de ciertos registros además no manejan datos de entrada ni salida con syscall si no directamente se encofan en operarlos para mostrar la ejecucion paso a paso.

Por otra parte, en un ejercicio completo debemos inicializar los registros involucrados, se hace uso de la entrada y salida y se implementa los casos mostrados en el libro pero en conjunto.

5 Ejecución paso a paso en MARS

Supongamos que la entrada por pantalla es 5.

Paso	Instrucción	Descripción (Entrada = 5)
1	`li $v0, 4`	$v0 recibe 4 para indicar la syscall de impresión de cadena.
2	`la $a0, msg`	$a0 recibe la dirección del mensaje almacenado en la sección .data.
3	`syscall`	Se imprime el mensaje solicitando el número.
4	`li $v0, 5`	$v0 recibe 5 para indicar la syscall de lectura de entero.
5	`syscall`	El usuario introduce 5. El valor 5 se guarda en $v0.
6	`move $s0, $v0`	El valor ingresado (5) se copia a $s0. Ahora $s0 = 5.
7	`li $t0, 1`	Se carga el valor 1 en $t0. Este valor servirá como referencia para detener el ciclo.
8	`ble $s0, $t0, finciclo`	Se evalúa si $s0 $\leq$ 1. Como $s0 = 5, no se cumple la condición y continúa el ciclo.
9	`addi $s0, $s0, -2`	Se resta 2 a $s0. Ahora $s0 = 3.
10	`j ciclo`	Salta nuevamente a la etiqueta ciclo.
11	`ble $s0, $t0, finciclo`	Se evalúa si 3 $\leq$ 1. No se cumple la condición.
12	`addi $s0, $s0, -2`	Se resta 2 nuevamente. Ahora $s0 = 1.
13	`j ciclo`	Se vuelve a la etiqueta ciclo.
14	`ble $s0, $t0, finciclo`	Ahora $s0 = 1. Como 1 $\leq$ 1, se cumple la condición y salta a finciclo.
15	`li $v0, 1`	$v0 recibe 1 para indicar la syscall de impresión de entero.
16	`move $a0, $s0`	El valor final de $s0 (1) se mueve a $a0 para imprimirlo.
17	`syscall`	Se imprime el número 1 en pantalla (indicando que es impar).
18	`li $v0, 10`	$v0 recibe 10 para finalizar el programa.
19	`syscall`	El programa termina su ejecución.

6 Justificar la elección del enfoque (iterativo o recursivo) según eficiencia y claridad en MIPS.

Tomando en cuenta los puntos que detallamos en la tercera parte, escogimos el enfoque iterativo para el algoritmo de Paridad, este nos ofrece mayor eficiencia en cuanto a espacio en memoria, ya que no tiene que reservar espacio en cada iteración, sino que funciona con el espacio reservado en un principio, y en cuanto a tiempo de ejecución, debido a que no depende de un caso base que al llegar a el tenga que retornarse a cada llamada recursiva, simplemente depende de el fin del ciclo; además de mayor legibilidad en el codigo y menor cantidad de líneas.

7 Análisis y Discusión de los Resultados

Al poner a prueba el algoritmo iterativo, se puede observar que el orden es de O(n), al someterlo a entradas muy grandes, hace que el tiempo de ejecución sea intolerable, dandonos cuenta que la correcta implementacion para este algoritmo es con el uso de la instrucción andi(And immediate) que solamente necesita verificar el ultimo bit de el número siendo un algoritmo O(1).

De igual manera este algoritmo no funciona para numeros negativos, siendo la condición de parada cuando el numero sea $n \leq 1$.

Por otra parte, ejecutamos en MARS el enfoque iterativo y recursivo para ver el comportamiento tomando en cuenta para ambos la entrada n=5, en el enfoque iterativo observamos que tiene un total de 23 Instrucciones:

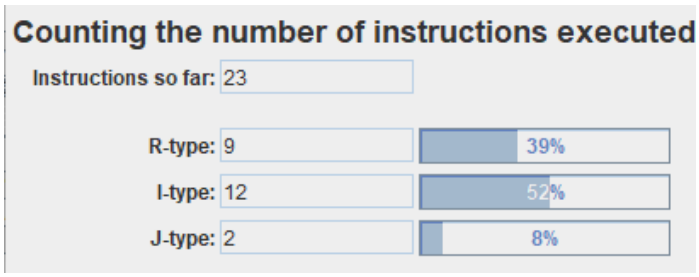

Figure 1: Contador de Instrucciones de MARS – Enfoque Iterativo

En cambio, el enfoque recursivo podemos observar que cuenta con 89 instrucciones:

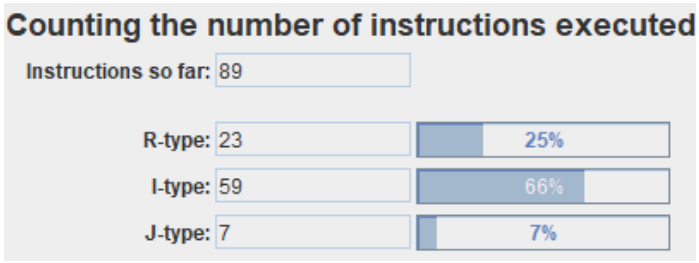

Figure 2: Contador de Instrucciones de MARS – Enfoque Recursivo