

Tutorial básico para el uso de la aplicación MARS

MARS (*Mips Assembly and Runtime Simulator*, <http://www.cs.missouristate.edu/MARS/>), es una aplicación *open source* para PC desarrollada por la Universidad de Missouri que facilita el ensamblado y la simulación de la ejecución de programas escritos en lenguaje ensamblador para el procesador MIPS. MARS dispone de una interfaz de desarrollo integrada (IDE) de fácil usabilidad y comprensión. También la puedes descargar desde el Moodle de la asignatura (sección tema 3).

En este tutorial se usará un programa ejemplo escrito para MIPS, problema 3.8 de curso, que calcula el factorial de un número escrito en una posición de memoria identificada por el símbolo (N) y escribe el resultado en otra posición identificada por (F). El objetivo de este tutorial no es entender el código dado, sino aprender a usar con un ejemplo concreto la aplicación MARS.

```
.text 0x0000
    lw $s1, N($0)          # s1 := N
    beq $s1, $0, Fone       # N == 0 ? -> F := 1
    addi $s2, $s1, -1       # s2 == next_factor := N - 1
    beq $s2, $0, Fone       # N-1 == 0 ? -> F := 1

    add $s3, $s1, $0        # s3 == cumulative_product := N

L1:   add $a0, $s3, $0       # a0 := cumulative_product
    add $a1, $s2, $0        # a1 := next_factor
    jal Fact                # v0 := cumulative_product * next_factor
    add $s3, $v0, $0        # cumulative_product := cumulative_product * next_factor
    addi $s2, $s2, -1       # next_factor := next_factor - 1
    beq $s2, $0, end        # next_factor == 0 ? -> F := cumulative_product
    j L1

Fone: addi $s3, $0, 1        # cumulative_product := 1
end:   sw $s3, F($0)        # F := cumulative_product

stop: j stop

Fact:  and $t0, $t0, $0      # t0 == cumulative_addition := 0
    add $t1, $a1, $zero     # t1 == counter := b

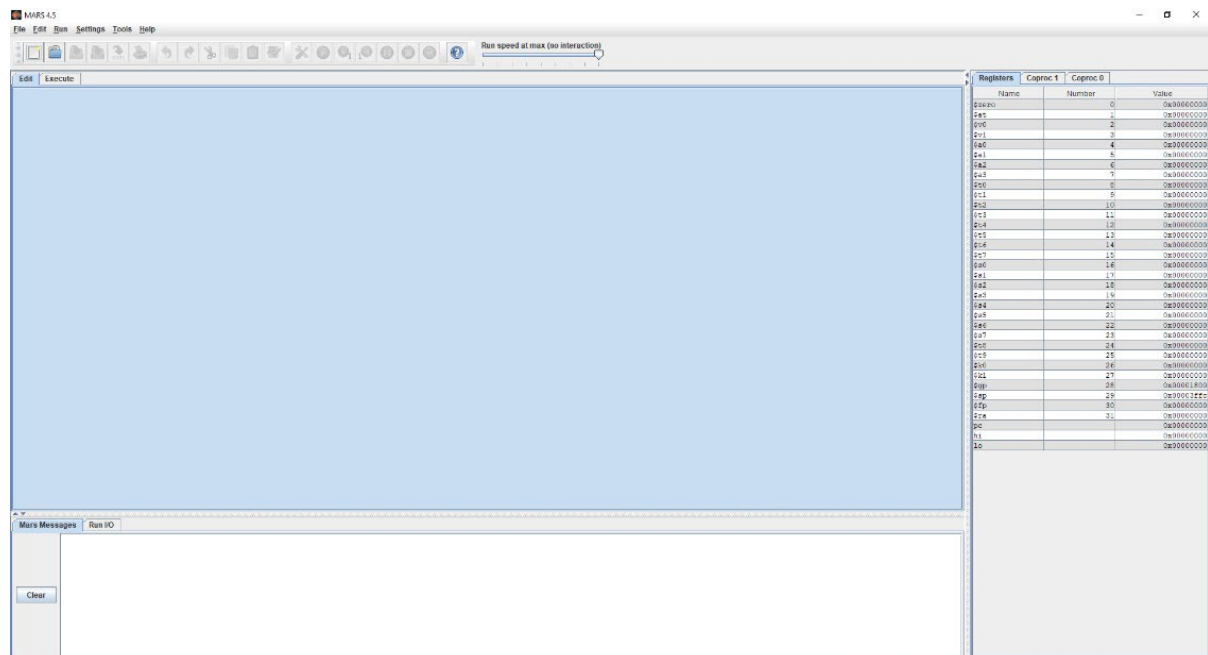
L2:   add $t0, $t0, $a0      # cumulative_addition := cumulative_addition + a
    addi $t1, $t1, -1       # counter := counter - 1
    beq $t1, $0, L3        # counter == 0 ? -> return cumulative_addition
    j L2                   # loop

L3:   add $v0, $t0, $0       # v0 := cumulative_addition
    jr $ra                 # return v0

.data 0x2000
    N: .word 0x00000003
    F: .word 0x00000000
```

1. Abrir MARS


Se accede a la aplicación MARS como a cualquier otra aplicación para Windows. Una vez se ejecuta, se abre una ventana como la de la figura.




2. Configurar la aplicación

Antes de empezar a escribir el código, conviene configurar la aplicación a nuestros gustos o necesidades. Se puede mantener la configuración por defecto que ofrece MARS, excepto la configuración del mapa de memoria. Esta **configuración necesaria** para el desarrollo de la asignatura se ejecuta desde el menú **Settings\Memory Configuration...** en donde hay que elegir la configuración **“Compact, Text at Address 0”**, opción necesaria para poder escribir el código a partir de la posición 0x0000 y los datos a partir de la posición 0x2000. Una vez seleccionada y guarda esta opción, MARS arrancará siempre con esta configuración, salvo que la aplicación se inicialice de nuevo, tal y como ocurre en las aulas de prácticas. Una vez que se ha descrito la configuración adecuada, ya se puede escribir un primer programa.

3. Escribir un programa

La ventana central (azul) es la pantalla de edición “Edit” y se utiliza para escribir el código. Para comenzar a escribir un programa, acceda al menú File\New o haga click con el ratón en el icono correspondiente a esta función .  La ventana de edición es operativa y permite la escritura.

Escriba o más fácil, copie y pegue el programa facilitado para calcular el factorial. Guarde en un archivo con el nombre Factorial.asm el código dado, accediendo al menú File\Save as... o haga click en el icono para esta  función .



La nueva interfaz tiene una apariencia como la de la figura superior y el significado de las tres nuevas ventanas es el siguiente:

“Text Segment”. Esta ventana asocia a cada dirección de la memoria de código (desde 0x0000 hasta 0x0044) el código máquina, el código ensamblador con las etiquetas o símbolos ya traducidos a un número y el código ensamblador tal y como ha sido escrito.



“Labels”. Esta ventana recoge la tabla de símbolos para la compilación efectuada. Si esta ventana no aparece, hay que acceder al menú de configuración (**Settings**) para seleccionar esta opción.

“Data Segment”. Esta ventana recoge el contenido de la memoria de datos. Por defecto presenta las direcciones a partir de la 0x2000, pero se puede seleccionar y visualizar cualquier dirección que se desee de forma sencilla, seleccionando dicha dirección en la lista inferior de la ventana.

5. Ejecutar un programa

Si tras la compilación todo está correcto, MARS ofrece varias opciones de ejecución:

- Ejecutar el código completo.

Seleccione esta opción haciendo click en el icono  , se ejecutará el programa completo y se detiene en la última instrucción si ésta ha sido codificada. Como no existen en MARS instrucciones específicas para detener la ejecución del código del tipo “HALT” o “STOP”, se utiliza una línea de código del tipo **“fin: j fin”**. Esta instrucción no detiene el código, sino que ejecuta un bucle infinito, ya que salta sobre sí misma. Para detener la ejecución es necesario hacer click en el icono de  parada , ubicado como el anterior en la barra de herramientas.

Una vez completado el código, se debe comprobar el resultado alcanzado. Para ello se dispone de la ventana con los registros de propósito general en la pestaña **“Registers”** en la ventana a la derecha de la pantalla y con el contenido de la memoria de datos, para el caso que se necesite comprobar el resultado de una escritura en la memoria en la ventana **“Data Segment”** ya comentada.

La utilización de los diferentes registros, aquellos que se pueden ver listados en la pestaña **“Registers”**, se puede hacer o bien a través de su nombre, como se ha realizado en el ejemplo que os proporcionamos, o a través de su número. Por ejemplo, para utilizar el registro **\$v0**, también podríamos hacerlo escribiendo directamente **\$2**. Se recomienda la

primera opción por ser más intuitiva, pero ambas opciones son correctas.

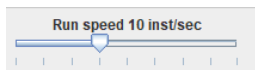
El programa ejemplo, calcula el factorial del número escrito en memoria en la dirección dada por la etiqueta **N** (0x2000) que en este caso contiene el valor 3. El resultado de la ejecución del programa es: $3! = 3 \times 2 \times 1 = 6$. Este resultado debe quedar guardado en el registro \$s3 en la ejecución por última vez de la instrucción "**L3: add \$s3, \$v0, \$0**" y también en la posición de memoria **F** (0x2004) al ejecutar "**end: sw \$s3, F(\$0)**".

Compruebe también que el contenido final del registro contador de programa "**pc**" apunta a la dirección **0x0024** que contiene la instrucción "**parar: j parar**", que no es la última instrucción escrita, sino la última instrucción que ha sido ejecutada, utilizada para detener por medio de un bucle infinito, la ejecución del programa.

Cambie el valor 3 por otro cualquiera y compruebe que el resultado obtenido en ambos casos es el factorial del nuevo número propuesto. Si el programa está bien diseñado y escrito, no debe depender del número del que se quiere obtener el factorial.

En la pestaña "**Registers**", además de los 32 registros de propósito general de MIPS, se puede ver el contenido del registro contador de programa "**pc**" así como de los registros "**hi**" y "**lo**" utilizados como destino en multiplicaciones. Las otras dos pestañas de esta ventana "**Coproc 1**" y "**Coproc 0**" se utilizan para las operaciones de coma flotante, que no se abordan en este curso.

La velocidad de ejecución de las instrucciones se puede controlar por medio de la barra de selección




Nota: Un ordenador convencional ejecuta millones de instrucciones cada segundo. Ejecutando el código varias veces, modifique la velocidad de ejecución en el rango permitido y compruebe el significado de esta función.

Existen otras opciones para ejecutar el código por bloques diseñando paradas intermedias o **breakpoints**, pero esta funcionalidad queda fuera de los objetivos del tutorial.

- Ejecutar las instrucciones una por una.

Esta opción se empleará cuando se quiere comprobar el comportamiento correcto de un código escrito para MIPS. Haciendo click con el ratón, respectivamente en los iconos



, se puede avanzar o retroceder respectivamente una instrucción. Cada vez que se ejecuta una instrucción, se observa cómo cambia el valor del registro contador de programa “**pc**” y también puede observarse el resultado de la misma, ya sea viendo el contenido en el registro destino que corresponda o en la posición de memoria en la que se haya programado escribir con “**sw**” correspondiente. Para volver en un solo click al comienzo del programa hay que pulsar en el icono . Al pulsar este icono se observa como el registro “pc” vuelve a cero.

Si al ejecutar el programa, no se han alcanzado los resultados esperados, se debe identificar el error o los errores cometidos, modificar de la forma adecuada la edición del programa (punto 3) y repetir los puntos 4 y 5 hasta conseguir los resultados propuestos.

MARS es una aplicación que permite simular con una mayor profundidad la arquitectura del procesador MIPS, con este tutorial se pretende mostrar las funcionalidades básicas de la aplicación para poder desarrollar la práctica propuesta y escribir los primeros programas en el lenguaje ensamblador diseñados para ser ejecutados en la arquitectura del procesador MIPS objeto del curso.