Practica 1 de laboratorio

Arquitectura del computador

Por:

José Rivero C.I: 28.492.353

Edwin Machado C.I: 30.532.641

Qué es MARS:

MARS, que significa "MIPS Assembler and Runtime Simulator", es una herramienta de software diseñada para facilitar el desarrollo y la depuración de programas escritos en lenguaje ensamblador MIPS (Microprocessor without Interlocked Pipeline Stages). La arquitectura MIPS es una arquitectura de conjunto de instrucciones (ISA) ampliamente utilizada en la enseñanza de arquitecturas de computadoras y sistemas embebidos.

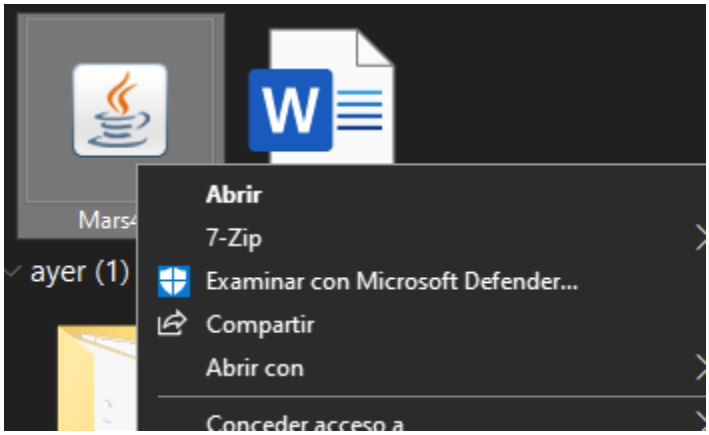
Instalación y Configuración de MARS en Windows

1. Descarga de MARS

Primero, descarga la última versión de MARS desde la página oficial del proyecto(http://courses.missouristate.edu/KenVollmar/mars/). Asegúrate de descargar la versión compatible con tu sistema operativo (Windows).

2. Ejecutar MARS

Dentro del directorio descomprimido, busca el archivo ejecutable llamado Mars4_5.jar. Puedes ejecutarlo haciendo clic derecho y abrir. Si eso no funciona, asegúrate de tener Java instalado y configurado en tu sistema.

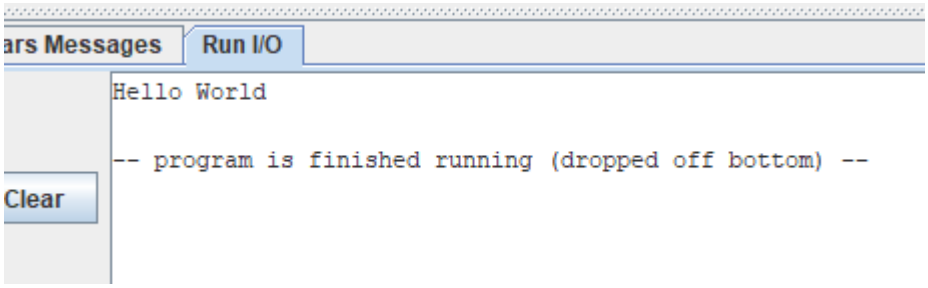


3. Configuración de Java

Es posible que necesites configurar la ruta del sistema para que MARS pueda encontrar tu instalación de Java. Para hacer esto, abre la pestaña "Settings" en la parte superior de la ventana de MARS y selecciona "Assemble File..." en el menú desplegable. Asegúrate de que la ruta al ejecutable de Java (por lo general, javaw.exe) esté configurada correctamente.

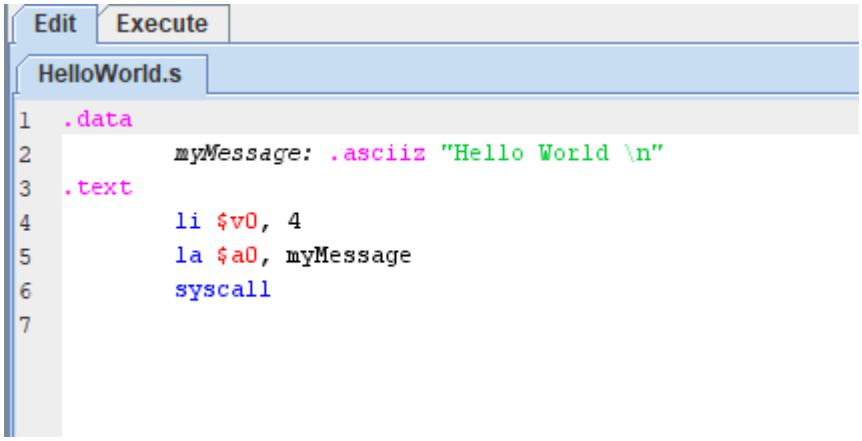
4. Comprobar la Configuración

Antes de comenzar a trabajar con MARS, verifica que la configuración sea correcta. Puedes hacerlo ensamblando un archivo MIPS de prueba o utilizando el archivo de ejemplo que viene con MARS.

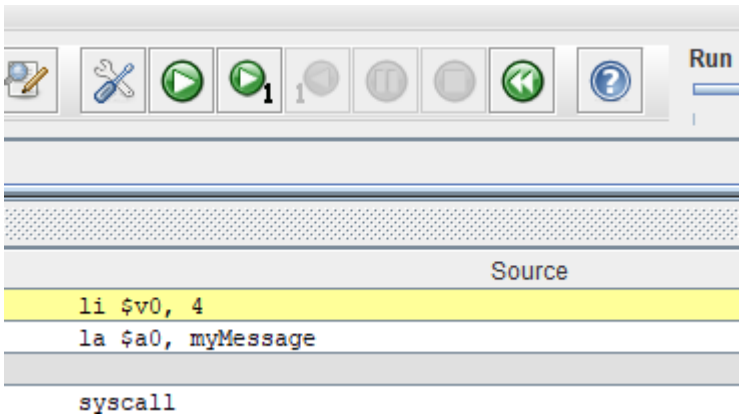


5. Uso Básico

Editor de Código: MARS proporciona un editor de código integrado. Escribe tu código MIPS en este editor.

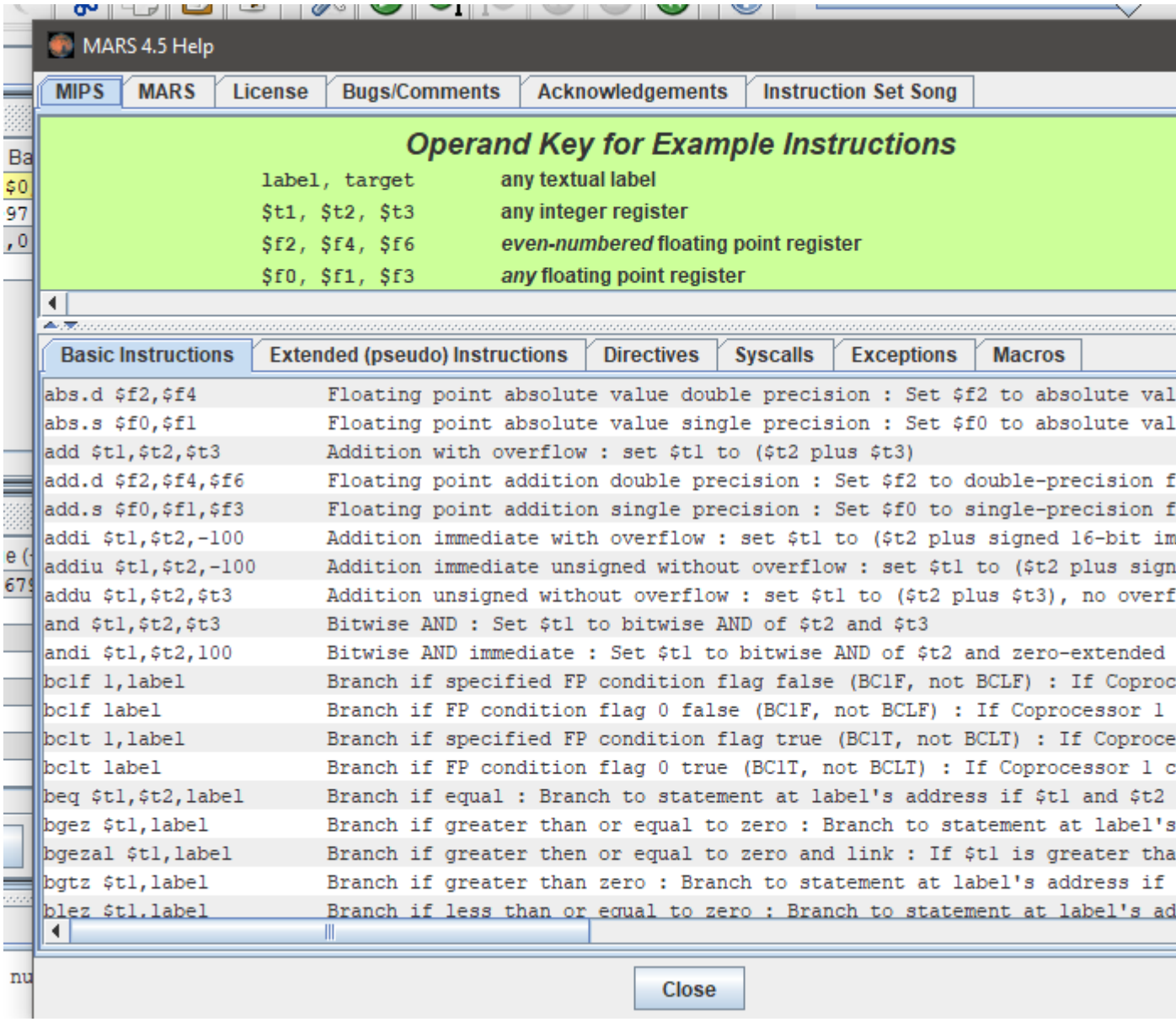


Ensamblado y Ejecución: Utiliza las opciones de ensamblado y ejecución para verificar la salida de tu código. Observa la ventana de registros y de memoria para depurar y entender el comportamiento de tu programa.



6. Recursos Adicionales

MARS viene con documentación que puedes consultar para obtener información detallada sobre sus características. Explora las opciones y ajustes disponibles para personalizar tu experiencia.



Códigos MARS:

Fibonacci:

Las principales diferencias entre los ejemplo académicos del libro y el de la página web de la universidad de Missouri, para el caso de Fibonacci tomaré el ejemplo de Fibonacci del libro (pág. 200) para hacer la comparación, este tiene un enfoque recursivo a diferencia de el de la página web que tiene un enfoque iterativo, por el enfoque académico del libro podemos notar que faltan tanto el .data y .text de la estructura del código, el código llama recursivamente a FIB y no hay una condición explicita que maneje el caso base para la recursión lo cual puede ocasionar una recursión infinita, en L1 después de la adicción (add $v0, v0, $s1) no hay un jr $ra para retornar a la función principal lo que puede causar errores.

En el ejemplo de la página de la web funciona correctamente al compilarlo como podemos ver a continuación:

```
The Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
-- program is finished running --
```

La cantidad de términos para la sucesión se define en el array de fibs y se le asigna el valor el tamaño por lo cual la cantidad de términos está definida en 19, si quisiéramos más términos podríamos modificar ese array o si queremos un término especifico entre menor a 19 podemos usar la implementación de input que está marcada como opcional dentro del código a modo de comentario ¿lo cual despliega el mensaje en pantalla “How many Fibonacci numbers to generate? (2 <= x <=19)” esto permite ingresar un numero entre el 2 y el 19, si para obtener términos mayores a 19 habría que modificar el tamaño del array en la sección .data:

```
.data
fibs:.word   0 : 19
size: .word  19
```

Row-major y Column-major:

Tanto Row-major como columna-major tienen un funcionamiento parecido la principal diferencia es la forma en que se almacena y se accede a los elementos de una matriz en memoria.

Row-major order (orden por filas):

En este método, los elementos de una fila se almacenan de manera contigua en la memoria. Esto significa que los elementos de la misma fila están almacenados uno tras otro en posiciones de memoria consecutivas. La dirección de memoria cambia principalmente a lo largo de las columnas. Al acceder a los elementos, se recorren todas las columnas para cambiar de elemento en la misma fila.

Column-major order (orden por columnas):

En este caso, los elementos de una columna se almacenan de manera contigua en la memoria. Así, los elementos de una misma columna se encuentran uno tras otro en posiciones de memoria consecutivas. La dirección de memoria cambia principalmente a lo largo de las filas. Al acceder a los elementos, se recorren todas las filas para cambiar de elemento en la misma columna.

Vamos a descocer el código para entenderlo mejor siendo el paso 7 la diferencia fundamental entre filas y columnas:

1.Se define un arreglo de palabras (data) que tiene espacio para una matriz de 16x16 elementos.

2.Se inicializan algunos registros: $t0 y $t1 se utilizan para almacenar el número de filas y columnas respectivamente. También se inicializan algunos otros registros para el control del bucle.

3.El bucle loop utiliza los registros $s0 (contador de filas) y $s1 (contador de columnas) para recorrer cada celda de la matriz.

4.Calcula el desplazamiento en la matriz (offset) para acceder a cada celda. Este desplazamiento se calcula como 4 * (fila * número_de_columnas + columna), ya que cada elemento en la matriz es una palabra de 4 bytes (word).

5.Almacena el valor en la posición correspondiente en la matriz usando la instrucción sw (store word).

6.Incrementa el valor que se asignará a la siguiente celda de la matriz.

7.F Hay instrucciones bne (branch not equal) que controlan el flujo del bucle. Si aún no se ha alcanzado el final de una fila, el bucle sigue recorriendo esa fila. Si se llega al final de una fila, se reinicia el contador de columnas y se incrementa el contador de filas para pasar a la siguiente fila. El bucle principal continúa hasta que se recorren todas las filas y columnas de la matriz.

7.C Hay instrucciones bne (branch not equal) que controlan el flujo del bucle. Si aún no se ha alcanzado la parte inferior de una columna, el bucle sigue recorriendo esa columna. Si se llega al final de una columna, se reinicia el contador de filas y se incrementa el contador de columnas para pasar a la siguiente columna. El bucle principal continúa hasta que se recorren todas las filas y columnas de la matriz.

8.Después de recorrer y asignar valores a toda la matriz, el programa finaliza con la llamada al sistema exit ($v0 = 10).

En el código podemos encontrar instrucciones como “mult” y “mflo”:

mult $s0, $t1 multiplicaría los valores contenidos en los registros $s0 y $t1. El resultado de la multiplicación se almacena en dos registros especiales llamados HI y LO.

mflo $s2 mueve el resultado de la multiplicación al registro $s2

El libro propone un ejercicio en C que también explica la manera de acceder a la memoria de un arreglo en una matriz:

Ejercicio 5.2

En este ejercicio se explora la localidad de memoria del cálculo con matrices. En el siguiente código C, los elementos de la misma fila se almacenan de forma contigua.

a.

```
for (I=0; I<8000; I++)
  for (J=0; J<8; J++)
    A[I][J]=B[J][0]+A[J][I];
```

b.

```
for (J=0; J<8; I++)
  for (I=0; I<8; I++)
    A[I][J]=B[J][0]+A[J][I];
```

En este ejercicio ambos bucles anidados realizan operaciones sobre una matriz “A” basados en los valores de otra matriz “B”. La diferencia es el como se accede y almacenan los elementos.

El primer código tiene un bucle externo que recorre 8000 elementos, y dentro de ese bucle, hay un bucle interno que recorre 8 elementos. La asignación de valores en la matriz A se realiza en función de los índices I y J.

En cambio, el segundo código tiene un bucle externo que recorre 8 elementos y un bucle interno que también recorre 8 elementos. La asignación de valores en la matriz A se realiza nuevamente en función de los índices I y J.

La diferencia crucial radica en cómo se accede a los elementos de las matrices A y B. En el primer código, los elementos de la matriz A se acceden siguiendo las filas de A y las columnas de B, mientras que, en el segundo código, la forma de acceso es a la inversa, siguiendo las filas de B y las columnas de A.

Suma Recursiva:

1. `.data` define la sección de datos. En este caso, se declara una variable `n` inicializada en 10 y una cadena de caracteres `result` que dice "El resultado es:".

2. `.text` define la sección de código.

3. `main:` es la etiqueta donde comienza la función `main`. Aquí se carga la dirección de memoria de `n` en el registro `$a0` y luego se carga el valor de `n` en el registro `$v0`.

4. Se mueve el contenido de `$a0` a `$v0`. Esto parece ser un error ya que `$a0` contendría la dirección de memoria de `n`, y mover esto a `$v0` sobrescribe el valor que acabamos de cargar de `n`.

5. Se llama a la función `sum` mediante `jal sum`.

6. Luego, se utiliza la llamada al sistema `syscall` para imprimir el mensaje almacenado en `result`.

7. Se carga el valor de `$t0` (el acumulador `acc` en el código `sum`) en `$a0` y se imprime mediante otra llamada al sistema `syscall`.

8. Finalmente, se hace una llamada al sistema para salir del programa.

9. `sum:` es una etiqueta donde comienza la función `sum`. Se verifica si `n` es menor o igual a 0. Si es así, se salta a `sum_exit`. Si no, se suma el valor de `n` a `$a1` (acumulador) y se decrementa `n`.

10. `sum_exit:` es una etiqueta donde se almacena el valor acumulado en `$t0` y se realiza un retorno de rutina.

Algoritmo de ordenamiento BubbleSort: