República Bolivariana de Venezuela
Universidad de Carabobo
Facultad de Ciencias y Tecnología
Departamento de Computación
Asignatura: Arquitectura del Computador

Bachiller: Brayan Ceballos C.I. 29.569.937

Informe – Práctica #1

Tutorial: Configuración del ambiente de trabajo de MARS en Windows 11

MARS es un emulador de CPU MIPS32 de código abierto que se usa para la enseñanza e investigación de la arquitectura de computadores. En este tutorial, aprenderás a configurar el ambiente de trabajo de MARS en el sistema operativo Windows 11. Como requisito el equipo dónde se va a descargar necesita:

- Un sistema operativo Windows 11.
- Un procesador de 64 bits.
- Al menos 4GB de RAM.
- Al menos 10GB de espacio libre en disco.

Para la configuración se hace lo siguiente:

- Abre MARS.
- En la barra de menú, haz clic en Herramientas – Opciones.
- En la ventana de opciones, selecciona la pestaña Emulador.
- En la sección Tipo de procesador, selecciona MIPS32.
- En la sección Memoria, establece la Memoria física a 16MB.
- En la sección Puertos, establece el Puerto de depuración a 1234.
- Haz clic en Aceptar para guardar los cambios.

Una vez seguidos estos pasos minuciosamente se puede trabajar con el ecosistema MARS. A continuación, usaremos el algoritmo de ordenamiento burbuja en MIPS32 para hacer referencia al ejemplo de interacción con el sistema.

1. En la ventana principal de MARS, haz clic en Nuevo para crear un nuevo archivo de código.
2. Se inserta el siguiente código en el archivo:

```
.data

array: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

.text

main:

  li $s0, 0

loop:

  beq $s0, 10, end

  lw $t0, array($s0)
  lw $t1, array($s0 + 4)

  slt $t2, $t0, $t1

  beq $t2, 1, swap

  addi $s0, $s0, 4
  j loop

swap:

  sw $t1, array($s0)
  sw $t0, array($s0 + 4)

  addi $s0, $s0, 4
  j loop

end:

  li $v0, 10
  syscall
```

3. Una vez con código copiado pues se guarda con el nombre de su preferencia, para este caso por comodidad lo guardamos como *ordenamiento_burbuja.s.*
4. Por último, haz clic en Ejecutar para correr el código.

MARS compilará el código y lo ejecutará en la CPU MIPS32 virtual. El resultado del algoritmo de ordenamiento burbuja se mostrará en la ventana de depuración.

Comparación entre Ejemplos Prácticos y Académicos

A continuación, haremos una comparación pertinente entre los ejercicios propuestos en la página web *https://courses.missouristate.edu/kenvollmar/mars/download.htm* y los del libro *Estructura y Diseño de Computadores, by D. Patterson & J. Hennessy.*

Ejemplo 1: Fibonacci

Cómo primer ejemplo tenemos el código en MIPS32 de Fibonacci. El código es muy similar al código que me proporcionaste primero, pero tiene algunas diferencias importantes. La principal diferencia es que el código académico (Patterson y Hennessy) utiliza números de punto flotante para representar los números Fibonacci. Esto permite utilizar la instrucción add.d para sumar los números, lo que es más eficiente que utilizar la instrucción add para sumar los números enteros.

Otra diferencia es que el código académico emplea una variable i para el manejo del bucle, en lugar de utilizar la variable t1 como en el código a comparar.

Por último, el código académico utiliza una rutina print diferente para imprimir los números Fibonacci. Esta rutina imprime cada número Fibonacci en una línea separada, con un espacio entre cada número.

Ejemplo 2: Recorrido en orden fila

Al hacer una comparación detallada podemos destacar lo siguiente, pero antes se quiere definir que el código al que haremos referencia como académico, corresponde al código del libro ya mencionado y como segundo tendremos el código a comparar. A continuación, se hace la comparación detallada:

- **Tamaño de la matriz:** El código académico está fijado en una matriz 16x16. Esto se refiere a que el código no se puede adaptar fácilmente para usar un tamaño de matriz diferente. En cambio, el código a comparar es más flexible ya que se puede usar con cualquier tamaño n de la matriz.
- **Orden de la iteración:** Los dos códigos utilizan diferentes órdenes para para sus bucles anidados. El código académico utiliza bucles anidados con el contador de filas primero, mientras que el otro utiliza bucles anidados con el contador de columnas primero. Esta diferencia en el orden de la iteración puede tener un impacto significativo en el rendimiento, según la aplicación específica.
- **Cálculo del desplazamiento:** Los dos códigos utilizan diferentes métodos para calcular el desplazamiento de cada elemento de la matriz. El código académico utiliza una operación de multiplicación y adición, mientras que el a comparar, tiene un enfoque basado en desplazamiento eficiente. Este enfoque está basado en desplazamiento general veloz, ya que no requiere una instrucción de multiplicación.
- **Control de la iteración:** Los dos códigos también utilizan distintos métodos para controlar el flujo de sus bucles. El segundo código comprueba el final de la matriz y luego comprueba el final de una fila, mientras que el académico comprueba el final de una fila y luego comprueba el

final de la matriz. Esta diferencia es que el control de la iteración también puede tener un ligero impacto en el rendimiento.

En general el segundo código es mucho más flexible, eficiente y conciso que el académico. También es más legible y fácil de entender. Esto debido al uso de nombres de variables más descriptivos y códigos más concisos.

Ejemplo 3: Recorrido en orden por columna

El código recorre un arreglo en orden de columnas. Esto significa que visita todos los elementos de una columna antes de pasar a la siguiente columna. El código utiliza un bucle anidado para iterar sobre las filas y columnas del arreglo. El contador de bucle para las filas es $s0, y el contador de bucle para las columnas es $s1. El valor que se almacenará en el arreglo es $t2.

La principal diferencia entre los dos códigos es la forma en que se calcula el desplazamiento de cada elemento del arreglo. El código de recorrido en orden de columnas utiliza un bucle anidado para iterar sobre las filas y columnas del arreglo, y luego calcula el desplazamiento de cada elemento utilizando la fórmula de desplazamiento. Por el contrario, el académico utiliza un solo bucle para iterar sobre los elementos del arreglo, y luego calcula el desplazamiento de cada elemento utilizando la fórmula desplazamiento = 4*($t2 / 16) + $t2 % 16.

Otra diferencia entre ambos es la forma en que se implementa el control del bucle. El código de recorrido en orden de columnas utiliza dos variables de control de bucle, $s0 y $s1. La variable $s0 es el contador de bucle para las filas, y la variable $s1 es el contador de bucle para las columnas. Caso contrario al ejemplo académico que usa como variable de control de bucle, $t2.

Se destaca en ambos, que son implementaciones correctas y eficientes de un recorrido en orden de columnas de un arreglo de 16x16 palabras. La elección de qué código utilizar probablemente se basará en la preferencia personal o en los requisitos específicos de la aplicación.

Evaluación de ejercicio : Suma recursiva

En este ítem se evaluará un ejercicio del libro de *Diseño y Arquitectura de Computador de Patterson y Hennessy* para terminar los elementos faltantes para que este funcione. Mostramos el ejercicio a corregir:

```
sum: slti$a0, 1              # comprueba si n <= 0
     beq$a0, $zero, sum_exit # salto a sum_exit si n <= 0
     add$a1, $a1, $a0        # suma n a acc
   addi$a0, $a0, -1          # decrementa n
   j sum                     # salto a sum
sum_exit:
   add$v0, $a1, $zero        # devuelve el valor acc
   jr $ra                    # retorno de rutina
```

¿Qué se le agregaría para que este código funcione?

La primera falta, la falta de inicialización de $a0, puede provocar resultados inesperados. Por ejemplo, si $a0 no está inicializado a un valor mayor que 0, el algoritmo puede entrar en un bucle infinito. Esto se debe a que la primera instrucción del algoritmo, slti $a0, 1, siempre devolverá un valor de 1, lo que hará que el beq $a0, $zero, sum_exit nunca se ejecute.

La segunda falta, la falta de almacenamiento del resultado, también puede provocar resultados inesperados. Por ejemplo, si $a1 no se almacena en la pila, el valor de $a1 se perderá cuando el algoritmo regrese a la rutina que lo llamó. Esto significa que la rutina que llamó al algoritmo no podrá obtener el resultado de la suma. Así quedaría el código corregido:

```
sum:
    # Inicialización de $a0
    li $a0, 10

    # Bucle principal
sum_loop:
    # Comprueba si n <= 0
    slti $a0, $a0, 1

    # Salta a sum_exit si n <= 0
    bnez $a0, sum_exit

    # Suma n a acc
    add $a1, $a1, $a0

    # Decrementa n
    addi $a0, $a0, -1

    # Salta a sum_loop
    j sum_loop

sum_exit:
    # Almacena el resultado en la pila
    sw $a1, 0($sp)

    # Devuelve el resultado
    lw $v0, 0($sp)
    jr $ra
```

Con estas modificaciones se corrigen las dos faltas del ejemplo académico y hacen que este funcione correctamente. En particular, la modificación li $a0, 10 inicializa $a0 a un valor mayor que 0. Esto garantiza que el algoritmo no entre en una iteración infinita.

Algoritmo de Ordenamiento: Quicksort

Adjunto con el informe irá un código en MIPS32 correspondiente al ordenamiento Quicksort para que se vea su escritura en MIPS32, en el presente informe solo se explicará el funcionamiento.

Inicialmente el algoritmo comprueba si el arreglo está vacío. En caso de que sí, este termina. De lo contrario, el algoritmo elige un pivote al azar. Luego, el algoritmo llama a la rutina partición para dividir el arreglo en dos subconjuntos, uno menor que el pivote y otro mayor o igual que el pivote.

Esta rutina partición comienza iniciando los índices del principio y del final del subconjunto izquierdo. Luego, el algoritmo entra en un bucle principal, acá comprueba si el índice del principio es mayor o igual que el índice del final. Si lo es, el algoritmo sale del bucle.

De lo contrario, el algoritmo comprueba si el elemento en el índice del principio es menor que el pivote. Si lo es, el algoritmo intercambia los elementos en los índices del principio y del final. Luego, el algoritmo incrementa el índice del principio. Finalmente, se decrementa el índice y vuelve a la iteración principal.