Universidad Simón Bolívar
Departamento de Computación y TI
CI-3815 Organización del Computador
Septiembre-Diciembre 2018

Tarea

El objetivo de esta tarea es que el estudiante se familiarice con el simulador MARS.

Parte 1: Revisión de material e instalación de MARS

- Revise el material disponible en Moodle relacionado con el Simulador MARS
- Descargue la última versión de MARS (*MIPS Assembler and Runtime Simulator)*
- Una vez instalado MARS, existen tres opciones para su ejecución
    - **Opción A: Desktop.** Guardar el archivo jar en alguna carpeta. Ejecutar MARS haciendo doble click sobre el ícono del simulador.
    - **Opción B: Interprete de comandos DOS usando archivo jar.** Guarde el archivo jar en algún directorio. Abra un interprete de comandos DOS en dicha carpeta usando cmd. Ejecute MARS usando el comando **java -jar Mars4_5.jar**
    - **Option C: Interprete de comandos DOS usando Java classes.** Guarde el archivo jar en algún directorio. Abra un interprete de comandos DOS en dicha carpeta usando cmd. Extraiga los archivos MARS usando el comando DOS **jar -xf Mars4_5.jar.** Ejecute MARS con el comando DOS **java Mars**

Parte 2: Ambiente MARS

- Una vez que MARS esté ejecutando, abra el archivo EsPrimo.asm seleccionando **File → Open** en la parte superior de la ventana de MARS. Esto hará que se vea en la ventana Edit de MARS el programa cargado.
- Analice el archivo y note que el editor de texto de MARS mostrará automáticamente el número de la línea de código del archivo fuente.
- Seleccione **Run → Assemble** y note el cambio en la apariencia de MARS. Analice las ventanas:
    - Text Segment
    - Data Segment
    - Labels
    - Registers

Registers

Antes de ejecutar el código ensamblado, seleccione la pestaña **Register**. Note que la mayoría de los registros están en cero excepto Stack Pointer ($sp), Global Pointer ($gp) y Program Counter ($pc).

1. Cuál es el contenido de estos registros en Hexadecimal y en Decimal ?

	Hexadecimal	Decimal
$sp		
$gp		
$pc		

Labels

La ventana **Labels** muestra todas las etiquetas o símbolos utilizados en nuestro programa escrito en lenguaje ensamblador y la dirección de memoria asociada con dicha etiqueta. Si se hace click sobre el nombre o la dirección en la tabla de símbolos se mostrará dónde se encuentra ya sea en el área de texto o de datos.

Rellene la tabla a continuación mostrando las etiquetas presentes en el programa cargado en MARS. Indique el nombre de la etiqueta y su dirección en hexadecimal. Además indique si la etiqueta se encuentra en el segmento de texto o de datos y cuál es su contenido en memoria.

Etiqueta	Dirección (Hex)	Text / Data	Contenido (Hex)

Text Segment

2. Cuántas instrucciones tiene el programa en lenguaje ensamblador? Nota: Sólo se consideran aquellas líneas de programa que posteriormente serán traducidas a lenguaje de máquina y colocadas en el segmento de texto.

3. En cuántas instrucciones de lenguaje de máquina fueron traducidas las instrucciones del programa en lenguaje ensamblador?

4. Cuántos bytes ocupa el programa en el segmento de Texto ?

5. Ubique la instrucción **bne $t1, $zero, is_prime_loop** de su programa. En cuál dirección de memoria se encuentra almacenada?

6. Cómo fue ensamblada dicha instrucción? De la respuesta en Hexadecimal

Data Segment

7. Cuál es el contenido de la dirección de memoria 0x10010004? Indique su contenido en ASCII y en Hexadecimal ?

ASCII								
Hex								

8. Cuántos bytes ocupa el segmento de datos ?

Parte 3. Ejecución

Ejecute el programa usando **Run → Go**

9. Cuáles registros cambiaron su contenido?

10. En cuál dirección de memoria se encuentra la última instrucción ejecutada?

Parte 4. Servicios de Entrada y Salida

MARS ofrece una forma de interactuar con el usuario recibiendo datos de entrada y/o devolviendo datos de salida. Estas entradas y salidas, así como otros servicios son ofrecidos a través de llamadas al sistema de operación usando la instruccion syscall.

Cómo es posible decirle al sistema de operación cuál servicio queremos utilizar?. La respuesta es simple. Antes de usar la instruccion syscall, es necesario cargar en diferentes registros los parámetros requeridos por el sistema para indicarle qué tipo de servicio debe ejecutar. Revise el menú de ayuda de MARS para conocer todos los servicios (SYSCALLS) disponibles y los parámetros requeridos.

Modique el programa *EsPrimo.asm* para que solicite el número a verificar y escriba por consola el resultado indicando “El número es primo” o “El número no es primo” en lugar de 1 o 0.

Fecha de Entrega: lunes 15 de octubre hasta las 11:55pm. Esta tarea es **INDIVIDUAL**

La entrega consiste en dos archivos que subirá al aula virtual:

- un archivo asm con el programa modificado y que identificará con su carnet (01-23456.asm)
- un pdf con las respuestas de las secciones 2 y 3 (01-23456.pdf)

```
.data
message:    .asciiz "Bienvenido "
num:        .word 45                        # Numero a verificar si es primo o no

.text
main:
      lw    $a0, num
      addi  $t0, $zero, 2                   # int x = 2

is_prime_test:
      slt   $t1, $t0, $a0
      bne   $t1, $zero, is_prime_loop       # if (x > num)
      addi  $v0, $zero, 1                   # El numero es primo!!
      b return                              # y en $v0 almacena 1

is_prime_loop:                              # else
      div   $a0, $t0
      mfhi  $t3                             # c = (num % x)
      slti  $t4, $t3, 1
      beq   $t4, $zero, is_prime_loop_con # if (c == 0)
      add   $v0, $zero, $zero               #
      b     return                          # en $v0 almacena  0 si
                                            # el numero no es primo
is_prime_loop_con:
      addi  $t0, $t0, 1                     # x++
      j     is_prime_test                   # continua verificando si
                                            # es primo o no

return:
      add   $a0, $zero, $v0                 # Escribe el resultado en la
      li    $v0, 1                          # salida estandar
      syscall                               # 1 para numero primo
                                            # 0 para no primo


      li    $v0, 10                         # Finaliza la ejecucion
      syscall
```