Organización de Computadores
Segundo Semestre 2021
Prof. Carlos Gonzalez, Viktor Tapia

Laboratorio 1: Instrucciones MIPS y Simulación en MARS

Objetivos de aprendizaje

- Predecir funcionamiento de un programa MIPS
- Traducir programas escritos en un lenguaje de alto nivel a MIPS
- Escribir programas MIPS que usan instrucciones aritméticas, de salto y memoria
- Usar MARS (un *IDE* para MIPS) para escribir, ensamblar y depurar programas MIPS

Entrega

Sube los archivos a través del Google Classroom correspondiente a tu sección de laboratorio:

1) Al terminar la sesión de laboratorio, prepara un breve reporte de la actividad de Laboratorio en un archivo PDF que contenga al menos las siguientes secciones: Introducción, desarrollo, resultados y conclusiones.

2) Al terminar la sesión de laboratorio sube todos los archivos ".s" que se piden en las distintas actividades de Laboratorio.

Parte 1: Probar las Predicciones del Pre-Laboratorio en MARS

Responder las siguientes preguntas y reporta los resultados:

1. Abre un archivo nuevo en MARS. Escribe el programa 1 del Pre-Laboratorio en el editor y luego guárdalo como "test1.s". Ensambla el programa, luego ejecútalo. ¿Los valores de los registros son como esperabas? Si no, reinicia y ejecuta el programa paso a paso para ver cómo cambian los valores de los registros.
2. Repite el procedimiento anterior para el programa 2 del Pre-Laboratorio. Nombra este programa "test2.s".
3. En el programa 2, ¿qué pasaría si se cambia la línea con `beq` a: "`beq $t0, $t0, A`"?

4. Repite el procedimiento en 1) para el programa 3 (nombre de archivo "test3.s").

Parte 2: Escribir programas MIPS

1. Considera el siguiente código tipo C. Traduce este código en instrucciones MIPS, y guárdalas en un archivo llamado "`program1.s`".

```
int x = 20; // usar $t0 para almacenar los valores de x
int y = x+5; //usar $t1 para almacenar los valores de y
int z = (x+4)-(y-9); //Usar $t2 para almacenar el valor de z
```

Ejecuta tu código para asegurarte que se comporta correctamente, y reporta los valores de `$t0,$t1 y $t2` al finalizar el programa.

2. Considera el siguiente código tipo C. Traduce este código en instrucciones MIPS, y guárdalas en un archivo llamado "`program2.s`". Cambia el valor inicial de "x" de modo que puedas comprobar que se ejecuta correctamente en todos los casos y reporta los valores de "y" de cada caso.

```
int x = 1;      // usar $t3 para almacenar valores de x
int y = -1;    // usar $t4 para almacenar valores de y
int z = x+y     // Seleccione un registro para guardar este valor
if (z == 0) {
    y++;
} else if (z == 1) {
    y--;
} else {
    y = 100;
```

```
}
```

3. Considera el siguiente código tipo C. Traduce este código en instrucciones MIPS, y guárdalas en un archivo llamado “program3.s” Ejecuta tu código para asegurarte que se comporta correctamente, asegurando que las ubicaciones de memoria guardan los valores esperados y reporta los resultados.

```
int[] a = new int[4];
// traduzca solo el código bajo esta línea.
// Asuma que el arreglo comienza en la dirección de memoria 0x10010000
a[0] = 3;
a[3] = a[0] - 1;
```

4. Considera el siguiente código tipo C. Traduce este código en instrucciones MIPS, y guárdalas en un archivo llamado “program4.s”. Ejecuta tu código para asegurarte que se comporta correctamente para distintos valores iniciales de a y b (enteros positivos) y reporta los resultados.

```
int a = 2;   // usar $t6 para almacenar valores de a
int b = 10;  // usar $t7 para almacenar valores de b
int m = 0;   // usar $t0 para almacenar valores de m
while (a > 0) {
       m += b;
       a -= 1; }
```

5. Para integrar todo, considera el siguiente código tipo C. Traduce este código en instrucciones MIPS, y guárdalas en un archivo llamado “program5.s”

```
int[] arr = {11, 22, 33, 44};
arrlen = arr.length; // traducción de lo de arriba está dada
// complete la traducción de lo siguiente...
int evensum = 0;          // usar $t0 para valores de evensum
for (int i=0; i<arrlen; i++) {
   if (arr[i] & 1 == 0) {  // ¿Qué significa esta condición?
     evensum += arr[i];

} }
```

Tu código MIPS debería comenzar con algo así:

```
.data
arr: .word 10 22 15 40
end:
.text
la $s0, arr    # esta instrucción pone la dirección base de arr en $s0
la $s1, end
subu $s1, $s1, $s0
srl $s1, $s1, 2  # ahora $s1 = num elementos en arreglo. ¿Cómo?
```

Todo bajo .data hasta .text es el segmento de datos, donde podemos declarar datos estáticos como arreglos. Todo debajo .text es el segmento de texto o código, donde escribimos las instrucciones del programa. Cuando ni .data ni .text están presentes en el archivo, se asume que el archivo completo contiene un segmento de texto. Asegúrate de probar tu programa con arreglos de distinto contenido y largos y reporta los resultados.