Organización y Arquitectura de Computadoras
2020-1
Práctica 6: Lenguaje Ensamblador

Profesor: José de Jesús Galaviz Casas [*]

Límite de entrega: Octubre 13, 2019

1. Objetivos

Generales:

- Introducir al alumno a la programación en lenguaje ensamblador.

Particulares:

Al finalizar la práctica el alumno estará familiarizado con el desarrollo de programas escritos en lenguaje ensamblador para la arquitectura MIPS, abarcando:

- Las funciones que ofrece la arquitectura.
- El conjunto de instrucciones.
- El proceso de ensamblado en código máquina del lenguaje ensamblador.
- Simular la ejecución de código máquina.

2. Requisitos

- **Conocimientos previos:**
  - Programación en algún lenguaje de alto nivel.
  - Aritmética de punto flotante.
  - Conjunto de instrucciones.
  - Modos de direccionamiento.

[*] Diseñada por Roberto Monroy Argumedo

- **Tiempo de realización sugerido:**
  5 horas.
- **Número de colaboradores:**
  Parejas
- **Software a utilizar:**
  - *Java Runtime Environment* versión 5 o superior.
  - El paquete MARS [**Mars**].

3. Planteamiento

La función primordial de una computadora electrónica moderna es la de ejecutar instrucciones para llevar a cabo una tarea, las instrucciones se le suministran en el lenguaje binario que la computadora inherentemente sabe ejecutar, dicho lenguaje se denomina **lenguaje máquina**. Para un ser humano resulta altamente complicado analizar y programar en el lenguaje máquina por lo que típicamente se usan lenguajes de alto nivel o, en su defecto, un lenguaje cercano al de máquina, con el mismo nivel semántico que él, pero usando palabras (mnemónicos) que le permiten al programador recordar las instrucciones. Este lenguaje, llamado *ensamblador*, puede ser fácilmente **ensamblado** (traducido) a lenguaje máquina. Para habituarnos a programar con él, escribiremos pequeños programas usando el conjunto de instrucciones de la arquitectura MIPS, además de simular su ejecución.

4. Desarrollo

MIPS es una arquitectura de tipo *load-store* con un conjunto de instrucciones reducido, fue desarrollado en la década de los ochenta y destaca por la elegancia del lenguaje y el control del flujo del programa, lo cual lo hace útil para comprender los conceptos de la programación en lenguaje ensamblador. En [**Patterson**] se pueden consultar las particularidades del conjunto de instrucciones así como detalles de su diseño e implementación. **MARS** [**Vollmar**] (MIPS *Assembler and Runtime Simulator*), como su nombre lo indica, es un ensamblador y simulador de la arquitectura **MIPS-32**, fue diseñado a partir de las especificaciones descritas en [**Patterson**], por lo que resulta convenviente para nuestro propósito. Además cuenta con un editor de texto con el que se puede crear y editar archivos de código fuente, el cual ofrece resaltado y sugerencias de completado de texto.

4.1. Instrucciones, directivas, etiquetas y comentarios

Las **directivas** son comandos para el ensamblador que simplifican algunas tareas, como guardar datos en la memoria con la codificación adecuada o reservar espacio de memoria. Según su efecto se colocan en distintos lugares del

código, pero todas comienzan con un punto (.). Otra característica útil del lenguaje son las **etiquetas**, las cuales nos ayudan a señalar direcciones de datos o instrucciones en la memoria, ya que en el momento de programar no conocemos las direcciones concretas de memoria en donde será almacenado nuestro programa. Las etiquetas son sustituidas posteriormente por el ensamblador y el sistema operativo por las direcciones realeas de la memoria. Para escribirlas se permite usar una combinación de caracteres alfanuméricos, puntos (.) y guiones bajos (_), no puede comenzar con caracteres numéricos. Para marcar una dirección de una instrucción o dato, se escribe la etiqueta, dos puntos (:) y la instrucción o el dato. Finalmente, dada la complejidad para entender un programa escrito en ensamblador, es recomendable documentar prácticamente cada línea de código, todo lo que se escriba después de una almohadilla (#), será considerado como un **comentario** y será ignorado por el ensamblador.

En el menú de ayuda del simulador puedes encontrar las instrucciones y directivas implementadas por el simulador MARS, también puedes consultar el manual en línea [**MarsGuide**].

4.2. Estructura básica de un programa en MIPS

Los archivos de código fuente para el lenguaje ensamblador MIPS se escriben en un archivo de texto plano con extensión `asm`. Por convención, la memoria del procesador MIPS se divide en múltiples segmentos, por lo que regularmente un archivo de código fuente cuenta con dos secciones:

- Datos. Con la directiva `.data` se le indica al ensamblador que los siguientes datos serán almacenados en el segmento de datos de la memoria, éstos serán utilizados por el programa durante su ejecución.
- Instrucciones. Se indica al ensamblador, con la directiva `.text`, que las siguientes instrucciones serán almacenadas en el segmento de texto de la memoria.

4.3. Ensamblado y simulación

Para simular la ejecución de un programa con MARS, primero es necesario guardar los cambios y ensamblarlo. El simulador nos mostrará el estado inicial de los segmentos de memoria y registros, al igual que las instrucciones traducidas en lenguaje máquina y en lenguaje ensamblador. Es posible controlar la velocidad de ejecución del programa o ejecutarlo paso a paso para analizar su comportamiento.

5. Entrada

En cada ejercicio se debe implementar un programa distinto por lo que cada uno tiene una entrada distinta. Las entradas deben almacenarse en memoria y declararse al inicio del código fuente.

6. Salida

Al igual que en las entradas, cada programa tiene una salida distinta. Los resultados deben guardarse en el registro `$v0`.

7. Variables libres

En el ejercicio 4, el número de iteraciones queda a discreción del alumno.

8. Procedimento

Escribe cada uno de los progamas de los ejercicios en lenguaje ensamblador de MIPS, cada uno en un archivo de código fuente distinto. Ensambla el código y ejecuta la simulación de cada uno.

9. Ejercicios

1. Escribe un programa que calcule la exponenciación logarítmica, basado en el siguiente algoritmo:

```
def exp_log(x,n):
    r = 1
    y = x
    while n > 1:
        if n%2 == 1:
            r = r * y
        n = n/2
        y = y * y
    r = r*y
    return r
```

2. Escribe un programa que calcule el máximo común divisor de dos números.

3. Empleando el coprocesador y las instrucciones de punto flotante de precisión sencilla, escribe un programa que calcule la serie:

$$4 \cdot \sum_{n=0}^{m} \frac{1}{4n+1} - \frac{1}{4n+3}$$

donde m es el número de iteraciones que se ejecutará el programa.

10. Preguntas

1. En el ejercicio 3:

   a) ¿A qué valor tiende la serie?

b) ¿Cuántos dígitos de la constante se pueden calcular con precisión sencilla? Justifica tu respuesta.

c) ¿Cuántas iteraciones son necesarias para calcular el mayor número de dígitos?

2. ¿Existe alguna diferencia en escribir programas en lenguaje ensamblador comparado con escribir programas en leguajes de alto nivel?

3. ¿En qué casos es preferible escribir programas en lenguaje ensamblador y en qué casos es preferible hacerlo con un lenguaje de alto nivel?