Taller de Programación II – Segundo Cuatrimestre de 2014

Simulador de Máquina Genérica

Arquitectura de la Máquina

La máquina genérica a simular tiene 16 registros designados de 0 a F (en base hexadecimal). Cada registro tiene un byte de longitud.

La memoria principal consta de 256 celdas de un byte, con direcciones representadas con patrones hexadecimales entre 00 y FF. Los puertos de entrada/salida se consideran en correspondencia con la memoria: FC, FD para entrada de datos, y FE, FF para salida. En ambos puertos, la primera celda es para control y la segunda para datos.

Para poder leer del puerto de entrada su celda de control debe contener el patrón 01 indicando que hay un dato para consumir, y luego de la carga del dato se debe almacenar en ella el valor 00 indicando que se consumió; para poder escribir en el puerto de salida el esquema es inverso: el consumidor es el dispositivo de salida y el programa es el que produce, así que se debe almacenar el dato en la celda de datos del puerto de salida y luego almacenar el valor 01 en la celda de control para indicar que el dispositivo tiene un dato para consumir.

Los valores enteros se almacenan en complemento a dos en 8 bits (-128..127). Los valores de punto flotante se almacenan en 8 bits, de los cuales el primero (más significativo o del extremo alto) se emplea para el signo, los tres siguientes para el exponente en exceso de 4 (-4..3), y los cuatro restantes (menos significativos o del extremo bajo) para la mantisa (.bbbb).

Cada instrucción de la máquina tiene 2 bytes de longitud (ocupa dos celdas). Los primeros cuatro bits constituyen el código de operación, los 12 bits restantes constituyen los operandos de la instrucción.

A continuación se describen las instrucciones de máquina en notación hexadecimal. Se utilizan las letras R, S y T para representar un dígito hexadecimal identificador de registro. Las letras X e Y se usan en lugar de dígitos hexadecimales para representar valores o direcciones de memoria.

Instrucciones

1RXY CARGAR el registro R con el patrón de bits que está en la celda de memoria cuya dirección es XY

2RXY CARGAR el registro R con el patrón de bits XY

3RXY ALMACENAR el patrón de bits que está en el registro R en la celda de memoria cuya dirección es XY

40RS COPIAR el patrón de bits que está en el registro R al registro S

5RST SUMAR los patrones de bits de los registros S y T en complemento a 2 y dejar el resultado en el registro R

6RST SUMAR los patrones de bits de los registros S y T en punto flotante y dejar el resultado en el registro R

7RST Disyunción lógica (OR) de los patrones de bits de los registros S y T colocando el resultado en el registro R

8RST Conjunción lógica (AND) de los patrones de bits de los registros S y T colocando el resultado en el registro

9RST Disyunción lógica exclusiva (XOR) de los patrones de bits de los registros S y T colocando el resultado en el registro R

AR0X ROTAR el patrón de bits del registro R un bit a derecha X veces

BRXY SALTAR a la instrucción situada en la celda de memoria cuya dirección es XY si el patrón de bits del registro R es igual al patrón de bits del registro número 0.

C000 PARAR la ejecución

Requerimientos

Se debe desarrollar un ambiente integrado de desarrollo para la máquina descripta que permita

- Editar programas en un lenguaje ensamblador definido para el lenguaje de la máquina con ayuda en línea para la codificación (el lenguaje se puede definir a partir de alguna máquina real, como por ejemplo ver http://opencores.org/project,natalius_8bit_risc)
- Traducir programas ensambladores a programas en código absoluto representado en hexadecimal
- Editar programas de código absoluto representado en hexadecimal con ayuda en línea para la codificación y numeración automática de direcciones de instrucciones (00..FF).
- Ejecutar programas de código absoluto en forma directa
- Ejecutar programas de código absoluto paso a paso mostrando para cada instrucción que se ejecuta el estado de la máquina resaltando los cambios que determina esa ejecución: memoria RAM, CPU (contador de programa, tres registros de instrucción de dos bytes para pipelining –ejecución, decodificación y obtención de instrucciones consecutivas–, 16 registros de un byte, ALU –resaltando overflows en sumas en complemento a dos y pérdidas de precisión en sumas en punto flotante)

Forma de Uso

Se debe poder editar programas en lenguaje ensamblador para luego traducirlos a lenguaje de máquina, o directamente editar programas en lenguaje de máquina. La persistencia de los programas debe realizarse en archivos de texto con extensiones “asm” y “maq” respectivamente.

Las instrucciones en lenguaje de máquina se deberán escribir en hexadecimal siguiendo la siguiente convención:

> A partir del primer renglón del archivo de texto y en tantos renglones como instrucciones tenga el programa se deberá escribir para cada instrucción: la dirección

de memoria en donde se almacenan los primeros 8 bits de la instrucción en hexadecimal, debiendo ser la primera siempre 00 (durante la edición estos números deben generarse automáticamente); la instrucción en hexadecimal separada de la dirección por uno o más espacios en blanco, y opcionalmente, un comentario en lenguaje natural, separado de la instrucción por uno o más espacios en blanco.

El ambiente solicitará el nombre del archivo para la creación de programas nuevos, sean en lenguaje ensamblador o en lenguaje de máquina, manteniéndose el nombre del programa cuando se traduzca de lenguaje ensamblador a lenguaje de máquina. Para abrir programas ya existentes, el ambiente debe mostrar los programas con la extensión correspondiente al del tipo que se pide abrir (asm o maq) en el directorio actual, permitiendo crear nuevos directorios y cambiar de directorio.

Durante la ejecución de un programa, si éste realiza operaciones de entrada de datos, en pantalla aparecerá el mensaje “Entra:” para que a continuación el usuario ingrese un valor de dos dígitos hexadecimales. El ambiente deberá proveer ayuda a solicitud del usuario para escribir números en decimal y obtener su codificación en complemento a dos o en punto flotante para copiarla en el campo de entrada.

Si el programa realiza operaciones de Salida, en pantalla aparecerá el mensaje “Sale:” seguido de dos dígitos hexadecimales. También se debe proveer ayuda a solicitud del usuario para convertir los dígitos hexadecimales a su representación decimal, debiendo indicar el usuario si se convierte a complemento a dos o a punto flotante (el usuario debe saber qué tipo de números produce el programa).

Programas de Prueba a Desarrollar

1. Leer dos números en punto flotante del teclado, calcular el producto del primero por el segundo y escribir el resultado en la pantalla.

2. Leer cuatro patrones del teclado y agruparlos de a dos considerando que conforman, en orden de lectura, las mitades más y menos significativas de un patrón en complemento a dos, hacer la suma aritmética de ambos patrones y escribir el resultado en pantalla. Tener en cuenta que para sumar las mitades menos significativas de ambos números no podrá usarse la suma aritmética en complemento a dos por el problema del acarreo.

3. Leer del teclado una secuencia de números enteros positivos hasta que alguno sea 0, almacenándolos en celdas consecutivas a partir de una a determinar, leer otro patrón de teclado y si el último patrón ingresado es $00 buscar el mínimo de todos ellos y escribirlo en pantalla, o si es $01 buscar el máximo de todos ellos y hacer lo mismo.

Pasos a Seguir y Tiempos Estimados

1. Realizar un diagrama de clases de diseño del ambiente y modelos de interfaz para la ejecución paso a paso de programas y discutirlos con el tutor: dos semanas hasta la muestra más una (si es necesaria) para correcciones u optimizaciones hasta nueva muestra.
2. Demostrar la edición de programas en lenguaje ensamblador y en lenguaje de máquina así como la traducción de unos a otros (programas de prueba): cuatro

semanas para primera demo más dos semanas para correcciones u optimizaciones y nueva demo, si es necesaria.

3. Demostrar la ejecución directa de programas (programas de prueba): dos [más una] semanas
4. Demostrar la ejecución paso a paso de programas: cuatro [más dos] semanas.