Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática
CI-0118 Lenguaje Ensamblador, grupo 01
Fecha: 29/abril/2019, I ciclo lectivo 2019

Tarea de Programación # 2: subrutinas y pila

1. Procedimiento para simular la ejecución de instrucciones

En esta tarea vamos a diseñar una máquina cuyo hardware no existe y vamos a simular la ejecución de algunas pocas instrucciones. Cada instrucción se implementará como un procedimiento `instr` que es llamado desde el programa principal o desde el mismo procedimiento.

La CPU tendrá 4 registros enteros de 64 bits numerados del R0 al R3. La memoria RAM se divide en dos bancos: una memoria de datos que se visualiza como un arreglo de 1024 bytes (solo 1 KB) y una memoria de instrucciones ejecutables que no vamos a implementar por ahora.

Las instrucciones de esta CPU tienen el siguiente formato:
`cod.Op, T1, Op1, T2, Op2, T3, Op3, vector de memoria de datos, vector de registros`
En donde

- cod. Op es el código de operación
- Ti es el tipo de este operando: operando de registro, de memoria o inmediato
- Opi es la dirección del operando.
- vector de memoria de datos es la dirección del inicio del arreglo de 1024 bytes
- vector de registros es la dirección del inicio de de una tabla que contiene el arreglo de registros. En total, esta tabla consta de 4 * 8 = 32 bytes de memoria.

Las operaciones se realizan entre el operando 2 y el operando 3, y el resultado se almacena en el operando 1, el cual nunca puede ser inmediato. Como se ve, vamos a simular los 4 registros de la CPU mediante 4 quadwords de memoria en la máquina real.

Cuando el simulador comienza, toda la memoria de datos y los registros de la CPU ya vienen inicializados en ceros binarios.

De momento, para esta tarea, solo 2 operaciones de la CPU van a ser definidas: ADD (4) y SUBTRACT (SUB) (8). Si el programa principal estuviese escrito en lenguaje C, los siguientes serían llamados válidos para una secuencia de instrucciones:

```
instr(ADD, R, R0, I, 100, I, 0, memarray, registers); pone R0 = 100
instr(4, 1, 0,  3, 100, 3, 0, memarray, registers); codificación de la instr. anterior

instr(ADD, M, 200, R, 0, I, 50, memarray, registers); memarray[200] = R0 + 50 = 150
instr(4, 2, 200, 1, 0, 3, 50, memarray, registers); codificación

instr(SUB, M, 40, I, 1000, M, 200, memarray, registers); memarray[40] = 1000 - 150 = 850
instr(8, 2, 40, 3, 1000, 2, 200, memarray, registers); codificación
```

Todos los argumentos se pasan por valor. `Memarray` y `registers` se pasan por referencia, pero aún estos se puede considerar que se pasan por valor, puesto que son el valor de una dirección.

La instr. `SUB` se debe implementar mediante un cambio de signo al tercer operando, para luego invocar a instr con el operador `ADD`.

Los resultados de la ejecución de estas instrucciones se observarán a través del debugger `GDB`, en el cual se ejecuta el programa y al finalizar se hace un vaciado de las áreas de memoria de interés.

Esta tarea se puede resolver en grupos de 2 personas. Entregar en Moodle el viernes 10 de mayo. Suba un archivo comprimido con los documentos en LaTeX, código fuente en ensamblador, imágenes que comprueben la ejecución del programa y el pdf respectivo.