Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (II/2017)

Enunciado Tarea 03

Programación Recursiva en Assembly, 50 % de la entrega 03

Fecha de entrega: 23 de Octubre del 2017, 17:00 hrs.

1. Motivación

Ahora que ya tenemos un computador programable y medios para transformar Assembly a lenguaje de nuestro computador básico, el objetivo del siguiente desafío es hacer programas complejos sobre nuestra arquitectura. En la versión física de esta entrega añadiremos un stack, lo que nos permite hacer llamadas a métodos y, por lo tanto, ejecutar algoritmos recursivos. Además, se añadirá el manejo de listas, algo que usaremos para manejar arreglos y matrices.

2. Descripción

Al igual que en las tareas anteriores, vamos a programar. La diferencia es que ahora será en un lenguaje de bajo nivel: Assembly. Para esta tarea van a tener que generar dos programas en Assembly y uno en C como bonus que serán descritos a continuación:

2.1. Primer Algoritmo: K-mínimo y K-máximo factorial

Para esta tarea deberán generar una programa que dada una lista de números, encuentre el K-mínimo y K-máximo de la lista y luego les aplique factorial a dichos números.

K-mínimo: Devuelve el k-ésimo menor valor de un conjunto de datos. K puede ir desde 1 hasta el largo de la lista. Ejemplo:

```
lista = [1,3,4,5,7,9,10,23] #Ordenada para visualizar mejor el K-mínimo
1-mínimo = 1
2-mínimo = 3
3-mínimo = 4
4-mínimo = 5
...
8 mínimo = 23
```

K-máximo: Devuelve el k-ésimo mayor valor de un conjunto de datos. K puede ir desde 1 hasta el largo de la lista. Ejemplo:

```
lista = [1,3,4,5,7,9,10,23] #Ordenada para visualizar mejor el K-mínimo
1-máximo = 23
2-máximo = 10
```

```
3-máximo = 9
4-máximo = 7
...
8 máximo = 1
```

Finalmente deberán aplicar factorial **de forma recursiva** al K-mínimo y K-máximo encontrado. Puede utilizar el algoritmo mostrado en esta páginaUsando la lista anterior, si solicitamos el 4-mínimo y 4-máximo, la respuesta será 5! y 7! respectivamente, es decir, 120 y 5040 (78_{16} y $13B0_{16}$ en base hexadecimal).

Inputs y Output

Para este programa el input serán 5 variables:

1. Mínimo: Esta variable deberá tener el mínimo factorial calculado. Parte con valor 0.
2. Máximo: Esta variable deberá tener el máximo factorial calculado. Parte con valor 0.
3. K: Esta variable indica el K-mínimo y K-máximo a buscar. Pueden suponer que siempre estará entre 1 y el largo de la lista
4. largo: Esta variable indicará el largo de la lista. Pueden suponer que siempre será mayor a 0
5. lista: Esta variable apuntará al primer valor de la lista

En Assembly este input se verá literalmente así:

```
1  DATA:
2  minimo 0
3  maximo 0
4  K 4
5  largo  8
6   lista  1
7          12
8          2
9          3
10         21
11         6
12         9
13         20
14   otras_variables....
```

Aclaraciones:

1. No puede suponer que la lista vendrá ordenada
2. Puede crear toda la variable que quiera pero siempre debajo de la lista, es decir, donde dice "otras_variables..."
3. No se preocupes por el *overflow* al momento de calcular el factorial
4. Deben usar **recursion** para calcular el factorial, no será considerado otra forma.

El output será editando las variables "mínimo" y "máximo" en donde deberán poner la respuesta en dichas variables, es decir, hacer `MOV (mínimo), resultado` y hacer `MOV (máximo), otro_resultado`.

2.2. Segundo Algoritmo: Grafo Cíclico

Luego de trabajar con listas, daremos un avance para trabajar con matrices[1]. Para este algoritmo deberán implementar lo necesario para verificar si un grafo es cíclico o no.
Se define grafo cíclico como aquel que posee un camino cerrado en el que no se repite ningún vértice a excepción del primero que aparece dos veces como principio y fin del camino, es decir, existe un camino donde partes de un nodo y llegas a ese mismo.

Para poder trabajar con grafos, utilizaremos su representación matricial. En particular, el grafo será codificado utilizando su matriz de adyacencia (revisar este link), que será almacenada en orden de filas y comenzará en la dirección de memoria asociada al label "matriz". La variable "columnas" indica el número de columnas de la matriz. El resultado del algoritmo (si es cíclico o no) debe indicarse en la dirección asociada al label "ciclico". El resultado debe ser "0" si es que no hay ciclo y "1" si es que hay.

```
1  DATA:
2   ciclico  0
3  columnas: 3
4  matriz 0
5          0
6          1
7          0
8          0
9          1
10         0
11         1
12         0
13   otras_variables....
```

Para ayudar la visualización, la matriz del ejemplo se vería así:

```
0 0 1
0 0 1
0 1 0
```

Para ayudar, la siguiente **imagen** muestra un grafo junto con su matriz de adyacencia.
Interpretación:

1. El nodo 1 está conectado con el nodo 3.
2. El nodo 2 está conectado con el nodo 3
3. El nodo 3 está conectado con el nodo 2
4. Hay ciclo entre el nodo 2 y nodo 3.

Aclaraciones:

- Puede existir ciclos de largo 1, es decir, un nodo puede estar conectado a sí mismo.
- El grafo es dirigido [2]
- La matriz siempre es cuadrada.
- Puede crear toda la variable que quiera pero siempre debajo de la lista, es decir, donde dice "otras_variables..."

[1] Lo cual no es tan fácil como es en Python
[2] `https://es.wikipedia.org/wiki/Grafo_dirigido`

2.3. Bonus: Segundo Algoritmo en C 1.5 pts

Para "aterrizar" el uso de Assembly es que presentamos el siguiente desafío: Implementar el mismo algoritmo de matrices en C[3]. Para esto deberá crear un archivo .c en donde el "main" deberá ejecutar todo lo necesario para encontrar un ciclo.

El input será:

```
int main(int argc, char const *argv[]) {
  int* matriz = [0, 0, 0, 0, 0, 1, 0, 1, 0];
  int columnas = 3;

  // Código tuyo

  // imprimir si el grafo tiene o no un ciclo.
}
```

El output debe ser mediante un `printf` que diga si hay ciclo o no. Ejemplo: `printf(''El grafo no tiene ciclo'');`

Tenga siempre presente que C funciona con punteros y variables tal como lo hace Assembly, por lo que recomendamos fuertemente investigar el uso de punteros en C. Por otro lado C es un lenguaje que debe ser compilado (con GCC o make) y deben asegurarse que sea posible compilar. Recomendamos utilizar el tutorial creado por el curso de Estructura de Datos y Algoritmos. También pueden utilizar esta página que permite programar C de forma online si es que tienen windows menor a la versión indicada en el tutorial.

3. Corrección

Para ingresar los valores, los ayudantes modificaremos manualmente las variables de cada algoritmo de Assembly y el algoritmo bonus.

Los algoritmos serán evaluados en base a diferentes tests. Cada test tiene asociado un puntaje binario, o sea "logrado" o "no logrado". Para la ejecución, se correrán sus programas en el Assembler del curso que está subido en el repositorio.

Nota: Este Assembler sólo funciona en el Sistema Operativo Windows. El algortimo bonus será evaluado ejecutando el código con gcc en consola[4].

4. Contacto

- Francesca Lucchini: flucchini@uc.cl
- Felipe Pezoa: fipezoa@uc.cl
- Hernán Valdivieso: hfvaldivieso@uc.cl
- Luis Leiva: lileiva@uc.cl

[3]Consideren esto como la oportunidad de aproximarse a C, lenguaje de programación ocupado en Sistemas Operativos y Estructura de Datos y Algoritmos

[4]Si su código en C se llama tarea.c, haremos `gcc tarea.c -o ejecutable` y despues `./ejecutable`

5. Integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1,1 en el curso y se solicitará a la Dirección de Docencia de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona.

Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente.

Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.