DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (I/2020)

Tarea 2

Assembly del computador básico, arreglos, control de flujo y punteros.
Fecha de entrega: 20 de mayo, 2020. 18:00 horas.

Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes 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. 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 (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), 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 (grupo) copia un trabajo, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir reprobación del curso y un procedimiento sumario. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.

Objetivos

- Aprender a programar en el lenguaje assembly del computador básico.
- Aprender sobre el manejo de punteros.
- Aprender el manejo de arreglos y el manejo del stack con subrutinas.
- Investigar sobre un algoritmo de ordenamiento sencillo.
- Investigar sobre las distintas formas de representar los *strings* en la memoria de un computador.
- Investigar sobre algunos de los distintos tipos de *encodings*.

Entrega y evaluación

Esta tarea es de carácter estrictamente **INDIVIDUAL**. Tiene dos tipos de preguntas evaluadas: programadas en *assembly* y teóricas de alternativas o similar disponibles en *canvas*. Para la pregunta del código de honor y las preguntas programadas, deberás enviar mediante un formulario de google:[1]

- Una imagen o PDF con tu respuesta a la pregunta del código de honor.
- Tus códigos en archivos `.txt` independientes con *encoding* **UTF-8 sin BOM**. Si no respetas el *encoding* no podremos corregirte.

Debes referenciar correctamente[2] en caso de utilizar material externo. Tus programas serán corregidos vía *tests* en el emulador del computador básico del curso, disponible junto con el enunciado.

Importante: el emulador solo corre en Windows de manera nativa. Para Mac y Linux puedes utilizar herramientas como WINE[3] que permiten correrlo. ACTUALIZACIÓN: estamos revisando alternativas para los que tienen mac. La solución que más a mano tenemos por el momento es que utilicen una máquina virtual, pronto haremos un foro para discutir más este tema, porque es algo de lo que también podemos aprender.

Código base para cada pregunta

En GitHub encontrarás el código base para cada pregunta. Puedes agregar nuevas variables entre los comentarios de control `//!begin-data` y `//!end-data`, y tu código debe ir entre los comentarios de control `//!begin-code` y `//!end-code`. Si modificas o borras estos comentarios especiales, no podremos aplicar los tests y tampoco corregir tu tarea. Nada del código que agregues fuera de estas zonas delimitadas por los comentarios contará como parte de tu entrega.

No puedes asumir que habrá una variable con el mismo nombre que el código base a menos que se indique el nombre de esta explícitamente en el enunciado de la pregunta. Tampoco asumas que estarán en el mismo orden que en el que aparece el código base.

Puedes utilizar subrutinas auxiliares y *labels* adicionales en cualquiera de las tres preguntas.

[1] Link completo:
https://docs.google.com/forms/d/e/1FAIpQLSf77OliFI7h7hF1blT1hGr_ct5qDt_LwC8x7C_78VepucBuqw/viewform?usp=sf_link
[2] Si tienes dudas, usa el formato APA.
[3] https://www.winehq.org/

Código de honor

Escribe a mano en un papel el siguiente texto y fírmalo. A continuación tómale una foto o escanéalo y súbelo al formulario. **No hacerlo equivale a un 1 en todas las preguntas de la tarea.**

> Tarea 2 - IIC2343
>
> Yo, <tu nombre>, afirmo que respetaré el código de honor de la universidad y no cometeré ninguna falta a la ética ni a la política de integridad académica.
>
> <número de alumno>
> <firma>

1. Comparación con signo

Restar dos números con signo para determinar desigualdades no es infalible. De hecho, si quisiéramos comparar 100 y -100, obtendríamos:

$$100 - (-100) = 200$$

Pero como estamos usando 8 *bits* **con signo** (complemento a 2), en realidad tendremos que

$$200 \Rightarrow -56$$

Esto produce las señales: $N = 1$, $Z = 0$, $C = 0$ y $V = 1$. Si entonces, nuestra condición de salto fuera $100 > -100$, no saltaríamos.

Teniendo en cuenta esto, crea un programa que compare los registros A y B y retorne **correctamente**:

- 1 si $A > B$
- 0 si $A = B$
- -1 si $A < B$

En específico, tu programa debe tener las siguientes características:

Nombre del *label*: `comparar`.

Argumentos que recibe: un número en A y otro en B.

Retorna: en la variable `res`, 0 si son iguales, 1 si A es mayor que B y -1 si B es mayor que A.

Forma de correr tu programa: usando un `JMP` para saltar a la *label* `comparar`.

2. *Selection Sort*

Existen numerosos algoritmos de ordenamiento y uno de ellos es *selection sort*[4], que destaca por su simplicidad. En esta pregunta deberás realizar una implementación de este algoritmo en el assembly del computador básico del curso.

[4]Esperamos que lo busques por tu cuenta o preguntes si estás demasiado perdido. De todas formas dejamos este vídeo disponible: link ☺

Se te dará el puntero a un arreglo de **hasta** 30 números en total (puede estar vacío), el puntero a su largo y el puntero a la base de un nuevo arreglo. Debes construir un programa que, utilizando *selection sort*, tome todos los elementos del primer arreglo y los coloque ordenados en el arreglo 2.

Los valores enteros estarán en el rango $[-60, 60]$. Tu programa deberá cumplir las siguientes características:

Nombre del *label*: `selection_sort`.

Argumentos que recibe: el puntero al arreglo en A y el largo del mismo en B.

Retorna: una copia del arreglo pero ordenado en la variable `arreglo_ordenado`.

Forma de correr tu programa: usando un `JMP` para saltar a la *label* `selection_sort`.

Restricciones: el arreglo puede estar vacío o contener hasta 30 elementos. Todos los números estarán entre -60 y +60, incluyéndolos.

3. Palíndromo

Un palíndromo es una palabra (o expresión[5]) que se lee igual tanto de izquierda a derecha como de derecha a izquierda. Para esta pregunta deberás determinar si el *string* codificado en ASCII que recibes es palíndromo o no, y para esto tendrás que implementar **dos** subrutinas:

- Una para calcular el largo de un *string NULL-Terminated*
- Una para determinar si un *string* es un palíndromo y que utilice la de subrutina para calcular el largo.

Debes dejar el *stack pointer* en la posición en la que estaba antes de llamar a la subrutina. Además, tus subrutinas deben cumplir con las siguientes características:

Calcular el largo

Nombre del *label*: `calcular_largo`.

Argumentos que recibe: el puntero al *string* en A.

Retorna: en la variable `len`, el largo del *string*.

Forma de correr tu programa: llamando con `CALL` a tu subrutina `calcular_largo`.

Restricciones: un *string NULL-Terminated* de entre 0 y 30 elementos (incluyendo a ambos) en codificación ASCII.

Determinar palíndromo

Nombre del *label*: `determinar_palindromo`.

Argumentos que recibe: el puntero al *string* en A y el puntero a la variable en que retorna en B.

Retorna: en la variable cuyo puntero le pasamos en B, 1 si es un palíndromo y 0 si no lo es.

Forma de correr tu programa: llamando con `CALL` a tu subrutina `determinar_palindromo`.

Restricciones: un *string NULL-Terminated* de entre 0 y 30 elementos (incluyendo a ambos) en codificación ASCII. Solo se utilizarán caracteres representables en esta codificación y no debes preocuparte de ignorar los espacios.

[5] Como el ejemplo clásico: *Anita lava la tina*, solo que no tienes que preocuparte de ignorar los espacios.