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

IIC2343 – Arquitectura de Computadores (II/2020)

Tarea 2

Aspectos preliminares

El objetivo de esta evaluación es que profundicen en los aspectos prácticos y teóricos de los tópicos de los capítulos 3 y 4 del curso. En esta evaluación hay preguntas teóricas y de programación, que estarán marcadas con **(T)** o una **(P)** al lado del número que las identifica, para diferenciarlas claramente.

La evaluación esta diseñada para que deban buscar material y recursos que no están en el sitio del curso para contestar las preguntas. Tomando esto en consideración, como esta tarea tiene un fin pedagógico, la regla de integridad académica que deben cumplir es la siguiente:

> *Toda respuesta a cualquier pregunta debe ser desarrollada y escrita individualmente. Esto significa que al momento de editar la respuesta, no se debe estar mirando o usando un texto escrito por otra persona, un video hecho por otra persona, o cualquier material que haya sido desarrollado por otra persona. Tampoco es válido memorizar una respuesta obtenida de otra fuente y luego escribirla. En otras palabras, lo que sea que se escriba debe provenir directamente de ustedes. Sin embargo, antes de comenzar a escribir cada respuesta pueden estudiar y comentar la pregunta con otros alumnos (no la respuesta, solo la pregunta), leer libros, mirar videos, o realizar cualquier otra acción que los ayude a responder la pregunta. Esta regla aplica tanto a pregunta teóricas como prácticas.*

Finalmente, la entrega deberá incluir un archivo llamado INTEGRIDAD.txt, en donde afirman que han leído, entendido y respetado la regla de arriba. De no cumplirse esto, la tarea no será corregida y obtendrán la nota mínima.

Entrega

La fecha de entrega es el viernes 23/10 hasta las 23:59. El formulario para la entrega se encuentra en la siguiente dirección `https://forms.gle/U9QPSiBH6qL1iJe37`. En este formulario encontrará la descripción del formato de entrega para cada una de las preguntas. Tenga en consideración que una vez vencido el plazo de entrega, las respuestas solo pueden ser entregadas 1 vez (usando los días de gracia).

1. Arquitecturas

a) (T) Transforme la arquitectura del computador básico (microarquitectura, ISA y assembly) para que esta soporte como tipos de dato nativos números enteros de 8 y 16 bits de manera independiente. **(3.0 ptos.)**

b) (T) Transforme la arquitectura del computador básico (microarquitectura, ISA y assembly) de manera que todas las instrucciones originales sean ahora implementadas con instrucciones de formato homogéneo, es decir, con formato `Instrucción + K` argumentos, donde `K` es fijo e igual para todas las instrucciones: `INST arg1, arg2, ..., argk`. Todas las instrucciones nuevas deben consistir en 1 palabra y solo pueden utilizar 1 ciclo del clock. **(3.0 ptos.)**

2. Arquitectura x86 y convenciones de llamada

a) (T) Indique y detalle cada uno de los elementos que impiden que el computador básico pueda implementar la convención de llamada *stdcall*. **(2.0 ptos.)**

b) (P) Considere una modificación al assembly x86 (x86++), que permite definir y llamar subrutinas con la siguiente sintaxis:

- **Definición:**

```
DEF subroutine(arg1, arg2, ..., arg n)
.
.
.
RET
```

- **Llamado:**

```
CALL subroutine(arg1, arg2, ..., arg n)
```

Además de esta modificación, la nueva sintáxis incorpora de manera transparente el uso de la convención *stdcall*, de modo que no es necesario incluir sus requerimientos tanto antes de llamar a la subrutina, como en el cuerpo de esta. En particular, se considera el manejo de las variables locales y del valor de retorno en *stdcall* como responsabilidad de la subrutina, por lo que x86++ no realiza modificaciones al respecto. Considere además que en x86++ **no debe existir colisión de nombres entre variables globales y los argumentos de las subrutinas**.

Dada esta descripción, escriba un programa en Python que tome código assembly x86++ y lo convierta en código assembly x86 tradicional. Un posible ejemplo de ejecución del programa se muestra a continuación:

- **x86++**

```
JMP main

DEF factorial(arg1):
MOV BX, arg1
CMP BX, 0
JE  set1
DEC BX
CALL factorial(BX)
MOV BX, arg1
MUL BL
JMP endfact
set1:
MOV AX,1
endfact:
RET

main:
CALL factorial(n)
MOV fact, AX
RET

n       dw 5
fact    dw 0
```

- **x86**

```
JMP main

factorial:
PUSH BP
MOV BP, SP
MOV BX, [BP + 4]
CMP BX, 0
JE  set1
DEC BX
PUSH BX
CALL factorial
MOV BX, [BP + 4]
MUL BL
JMP endfact
set1:
MOV AX,1
endfact:
POP BP
RET 2

main:
PUSH n
CALL factorial
MOV fact, AX
RET

n       dw 5
fact    dw 0
```

(4.0 ptos.)

3. I/O

En este ejercicio, construirá componentes en Python para el emulador del computador básico, que simulen el comportamiento de dispositivos de I/O. Luego, utilizando la nueva versión del emulador, interactuará con estos dispositivos para resolver un problema. **Importante:** Antes de empezar a programar, revise la documentación y ejemplos del emulador para familiarizarse con las especificaciones detalladas del funcionamiento, declaración, comunicación e interfaz de los dispositivos de I/O.

a) (P) Construya un componente que emule el comportamiento de un coprocesador matemático, capaz de sumar, restar, multiplicar y dividir números de punto flotante de 16 bits (IEEE754 half precision). El coprocesador debe poder ser controlado a través de *memory mapped I/O* y *port I/O*. **(1.5 ptos.)**

b) (P) Construya un componente que emule el comportamiento de una cinta magnética de almacenamiento, capaz de almacenar 1024 bytes de contenido binario. La cinta debe poder ser controlada a través de *memory mapped I/O*. **(1.5 ptos.)**

c) (P) Construya un componente que emule el comportamiento de un *Address Decoder* para el computador básico, redireccionando las llamadas a las direcciones correspondientes de cada dispositivo mapeado a memoria. **(1.5 ptos.)**

d) (P) Escriba un programa en el assembly del computador básico, que lea el contenido de la cinta magnética, interprete el contenido como números de punto flotante de 16 bits, calcule el promedio de estos y finalmente lo almacene a continuación del último número almacenado. La cantidad de números almacenados en la cinta debe ser especificada por el mismo programa. **(1.5 ptos.)**

IMPORTANTE: Considere que todos los dispositivos de I/O tienen cierta demora entre que reciben un comando y terminan de ejecutarlo (no son acciones instantáneas). Para cumplir esto, simule un *delay* aleatorio por cada llamada, parametrizado de manera adecuada a cada dispositivo.

MÁS IMPORTANTE TODAVÍA: Todos los ítems de esta pregunta deben entregarse juntos en un único archivo `.zip`. Este archivo debe contener lo siguiente:

- 1 archivo `.py` por cada dispositivo de I/O donde se defina su funcionamiento.
- 1 archivo `.ron` con la especificación del dispositivo desde el punto de vista de IO.
- 1 archivo `.txt` con el código assembly de la parte d).
- 1 archivo `.py` con el programa que inicializa y ejecuta todas las interacciones (un `main`).
- 1 *readme*, que explique claramente la lógica, funcionamiento y ejecución de su respuesta.

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.