IIC2343 – Arquitectura de Computadores

Examen

Profesor: Yadran Eterovic

27 de Junio de 2019 - 9:00 – 11:30

Instrucciones:

1. **Escribe tu nombre y número de alumno en cada hoja que utilices.**
2. En cada hoja puedes responder a lo más una pregunta.
3. Si no respondes alguna pregunta, entrega una hoja con **tu nombre y tu número alumno** para esa pregunta de todas formas.
4. El incumplimiento de cualquiera de las instrucciones **podría significar una penalización en tu nota de 5 décimas.**

Pregunta 1

a) Transforma 0,1 de base 10 a punto flotante en un computador de 16 bits según la convención IEEE754 (1 bit de signo y 5 bits de exponente). Repite el proceso con el decimal 0,2. Luego, suma 0,1 + 0,2 como punto flotante y transforma el resultado a base 10. ¿Es igual a 0,3? ¿Por qué?

Respuesta:
La composición de un número de punto flotante de 16 *bits* según la convención es la siguiente:

1 *bit* de signo + 5 *bits* de exponente (desplazado en 15) + 10 *bits* de significante

Primero hay que transformar el $0,1$ a binario, el que queda como $0,0\overline{0011}$, lo que es equivalente a $1,1\overline{0011} \times 2^{-4}$. Por lo que el $0,1$ se escribe de la siguiente forma:

- Signo: $0 = (0)_2$
- Exponente: $15 - 4 = 11 = (01011)_2$
- Significante: $(1001100110)_2$

Para el 0,2 solamente cambia el exponente.

Luego, los números pedidos quedan:

$0,1 = (0010111001100110)_2$
$0,2 = (0011001001100110)_2$

Finalmente la suma:

$$(0,00011001100110)_2 + (0,00110011001100)_2 = (0,0100110011001)_2 < 0,3$$

La razón por la que la suma de 0,1 y 0,2 no es igual, es porque los números pedidos no son representables de manera exacta como punto flotante y por lo tanto, se trunca el significante.

b) ¿Cuánto es $(0xFE + 0x1) \times -0x4C$, suponiendo 8 bits y complemento de 2?

Respuesta:

$$(0xFE + 0x1) \times -0x4C = 0xFF \times -0x4C = 0x4C$$

Puede expresar el valor como lo estime conveniente.

Pregunta 2

a) ¿Por qué no se puede distinguir, en general, datos de instrucciones en la memoria en un computador Von Neumann? **(1/4 pregunta = 1,5 puntos)**

Respuesta:

Porque se pueden guardar programas en memoria y ser utilizados como si fuesen datos y luego querer ejecutarlos. Del mismo modo, puede ejecutar el contenido de un archivo de texto, imagen o demás, como si fuese un programa.

Si ponen que un valor de datos puede ser igual a un opcode, mitad de puntaje (**0,75**), ya que puede estar rodeado de valores que no sean opcodes y uno puede ver perfectamente que en ese caso es un dato y no un opcode.
Si ponen que instrucciones y datos tienen el mismo largo, **0,75 puntos.** (Es equivalente, en parte, a lo anterior).
Si ponen solo que no se puede distinguir porque están en la misma memoria, **0,375**.
Si aluden a la autoprogramabilidad, 1,125 (falta el concepto de multiprogramación y ejecución de datos como código).

Pueden argumentar que es posible si se siguen ciertas convenciones/protocolos fijos pero la escencia es la misma y se considera correcto junto a los argumentos correspondientes.

b) ¿Cómo se podría hacer para construir un computador en el que se distingan datos de instrucciones en la memoria tomando en cuenta que el computador **debe** ser Von Neumann? **(1/4 pregunta)**

Respuesta:

Hay varias respuestas correctas para esta pregunta, a continuación se enumeran algunas.

Si propone "separar datos de instrucciones en memoria", pero no indica el cómo: 0,75.

Si olvida los literales (por ejemplo, intercalando instrucciones y datos en toda la memoria), 0 puntos.

Si proponen no-execute bit, pero les falta el concepto de página/marco físico, **1,125**.

Si proponen un bit extra por cada dirección de memoria, 0,75. Es una mala solución. A la hora de escribir en memoria, además, habría que estar actualizando esos bits y no podríamos saber a priori con qué valor setearlos, etc. Tiene muchos problemas. **Pero** si da más detalles respecto a la actualización/preservación de esos valores y menciona algún componente asociado que deba ser modificado y qué modificación debe hacerse (a grandes rasgos), pero no funciona para el caso de la multiprogramación, **1,125** puntos.

Caché split: todo el puntaje.

Si su solución es tener la mitad de la memoria con instrucciones y la otra con datos, medio puntaje, ya que no se hace cargo de la multiprogramación. Tampoco reconoce que puedo tener un programa guardado como datos y luego querer ejecutarlo.

Si entrega más de una solución y alguna de ellas dice cosas incorrectas (no que no resuelvan el problema, sino que estén mal), no recibe puntaje.

c) Para cada uno de los siguientes programas escritos en assembly del computador básico, indica si durante la ejecución se produce el salto o no. No hay pipeline. Justifica con los valores de las señales Z, N, C y V entregados por la unidad aritmética lógica. Recuerda que el computador básico es de 8 bits y su arquitectura es Harvard. **(1/2 pregunta)**

I.-
```
MOV A,3
CMP A,4
JMP Fibonacci
...
```

II.-
```
MOV A,1
MOV B,-9
SHR A, A
JEQ Arquiyudantes
...
```

III.-
```
MOV A,5
MOV B,-5
XOR A,B
JNE BonusLealtad
...
```

IV.-
```
MOV A,7
MOV B,-2
INC B
JLT YadranEsGenial
...
```

V.-
```
MOV A, 12 ; sin signo
SUB A, 255 ; sin signo
JCR JurgenMandaSaludos
...
```

```
                                    ADD A,127                   ...
VI.- MOV A,127 ; con signo          JOV Germoji
```

Todo comienza con saber de que los saltos condicionales no utilizan un compare directo entre los registros A y B del computador, sino que se guía explícitamente por las señales de control `Z`, `N`, `C` y `V` liberadas por la ALU hacia el registro `STATUS`. La instrucción `CMP A,B` y su variante `CMP A,Lit` solo fuerzan un valor de

Si intentan comparar A y B cuando no corresponde o realizan mal la operación, no reciben puntaje.

Tener mal las señales (equivocarse en la letra), medio puntaje.

Errores menores en las operaciones (que no puedan ser confundidos con una coincidencia), medio puntaje.

Mal manejo de señales no recibe puntaje.

Si menciona bit de signo: -0,25 al puntaje obtenido

I.- Al ser un salto incondicional, salta sin importar los valores de las señales de control.

II.- Z = 1 ya que la ultima instrucción es 1 >> 1 lo cual es igual a 0. Por lo tanto, Salta.

III.- -5 xor 5 = 0xFE. Z = 0 y JNE Salta.

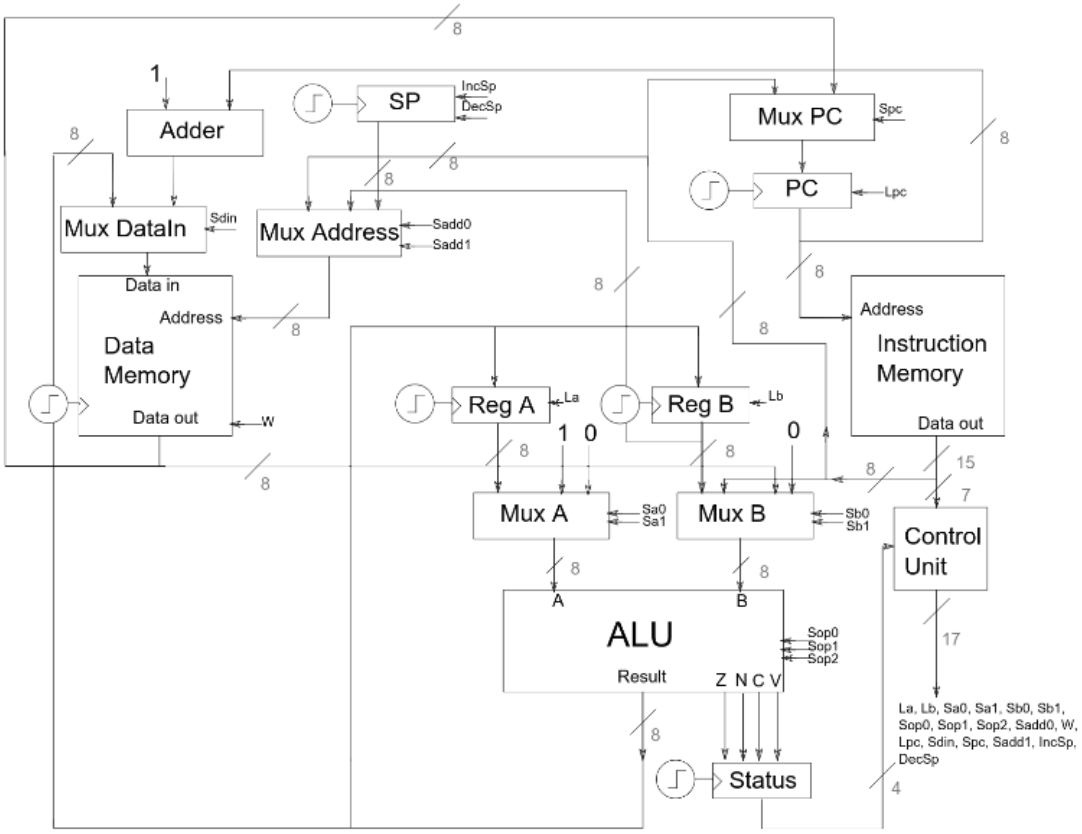
IV.- -2 + 1 = -1, N = 1 y Z = 0. JLT Salta.

V.- 12 - 255 = -243. Pero, la ALU hace 12 + 1 = 13. Por lo tanto, hay carry (ya que el valor real queda cortado por el límite de 8 bits). Si alguien pone que es -13, mitad de puntaje.

VI.- suma de dos números positivos acaba en número negativo (0b11111110), por lo tanto hay overflow. Sí salta

Pregunta 3

Determina si las siguientes instrucciones son soportadas por el computador básico en su versión final (sin pipeline, el diagrama se encuentra a continuación). En caso afirmativo indica las señales de control de los componentes involucrados; de lo contrario, **justifica**.



a) Una instrucción `SET Lit` que actualiza en un mismo ciclo los valores de los registros A y B con el valor del literal.**(3pts)**

Respuesta:

Puede ser soportada si las señales de control se asegura de tener: El resto de señales no me interesan, por lo

La	Lb	Sa	Sb	Sop
1	1	10	01	000

que pueden ser *Don't Care* (X)

b) Ejecutar las instrucciones `MOV B, (Dir)` y `ADD A,B` en el mismo ciclo de modo que en el registro A quede la suma de `A + B` y en B el valor de `MEM[Dir]`**(3pts)**

Respuesta:

Es imposible ejecutar esta instrucción ya que para cargar en un registro un valor guardado en memoria ,este debe pasar por la ALU, sin embargo esta ya estar ıa siendo utilizada para guardar en A el valor de A + B.

Pregunta 4

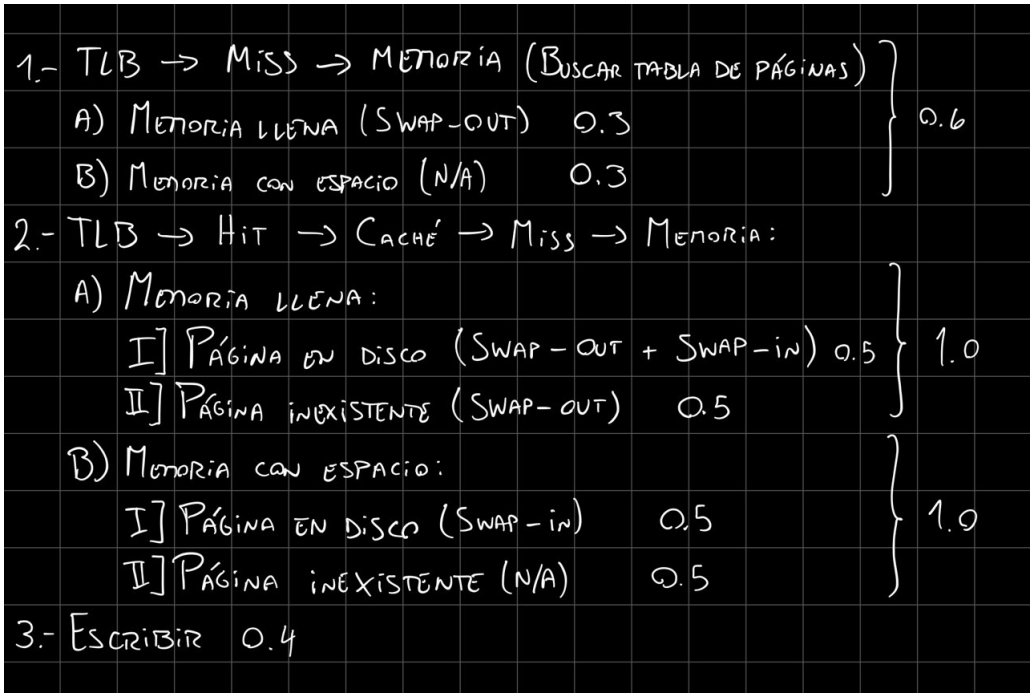
a) Dibuja la `TLB` en el diagrama del computador básico (Arquitectura Harvard). Notar que para posicionar la `TLB` tendrás que usar el diagrama anexo al final del enunciado y completarlo.

Respuesta:

La forma de contestar bien esta pregunta era interceptar la TLB entre el *Mux Address* y la *Data Memory*. Sólo importaba la traducción, no la actualización de la TLB, por lo que si está conectada de forma que ingresa la salida del *Mux Address* y su *output* es el *Address* de la *Data Memory*, la respuesta es correcta (3 pts).

b) Un programa que se encuentra en el disco de un computador comienza a ser ejecutado. Considera que la primera instrucción del programa escribe en memoria el dato de un registro. ¿Cuántos ciclos tomará la ejecución de dicha instrucción? Explica **detalladamente**.

—



Como no sabemos si estamos escribiendo un valor perteneciente previamente al programa o no, debemos cubrir ambos casos (página en el disco o página inexistente). También tomar en cuenta de que el programa comienza a ser ejecutado, por lo que la TLB estará vacía. Los errores (ciclos extra) serán penalizados con $-0{,}2pts$. No es necesario (de hecho, es imposible) dar la cantidad exacta de ciclos (no sabemos cuántos toma swap out ni swap in), pero se deben dejar como incógnitas (la respuesta esperada no es de caracter numérico, más bien es de caracter analítico). Hay una modificación en los puntajes: 1.a vale 0.2 y 1.b vale 0.4.

Pregunta 5

a) Tenemos un computador muy especial, al que decidimos llamar "HarvardSix", ya que su arquitectura es tipo Harvard y su pipeline consta de las siguientes seis etapas:

`IF | ID | MEM | EX | MEM | WB`

Además el computador presenta un tipo de memoria de datos que permite dos accesos simultáneos en cada ciclo. La primera etapa MEM utiliza el flanco de subida y la segunda etapa MEM el flanco de bajada.

A partir de esta información, responde lo siguiente:

I.- Escribe dos instrucciones que son soportadas por este computador y que no lo son por el computador básico con pipeline; considera que las instrucciones pueden aprovechar la etapa de MEM previa a EX.

Respuesta:

Se esperan instrucciones del tipo MEM[A] = MEM[A] + Lit ó A = MEM[A] + MEM[B], Ahí ya hay al menos dos ejemplos de instrucciones que son soportadas por este computador y que no lo son por el computador básico con pipeline.

0.35 puntos por cada intrucción

II.- Indica los tipos de hazards que se pueden producir y los que no.

Respuesta:

Pueden ocurrir cualquier tipo de *hazards* excepto los de tipo estructural.

0.3 puntos por mencionar los hazard que pueden ocurrir

0.3 puntos por mencionar que no se puede hazard estructural

III.- ¿Qué tipo de problemas puede generar este pipeline de seis etapas para las instrucciones de saltos condicionales?

Respuesta:

Básicamente, como el salto condicional se resuelve en EX, cuando se llegue a la segunda etapa MEM se podría decidir que la predicción de salto fue errónea y habría que hacer flush de la primera etapa MEM. Esto podría ser complicado, en el sentido de que si se alteró un valor utilizado por la segunda etapa de MEM se terminan ocasionando incoherencias en las mismas ejecuciones, por lo que el *flush* podría ser más profundo o combinarlo con *stalling*, en cualquiera de los casos mi número de ciclos totales se verán afectados.

0.7 por mencionar y explicar bien la problemática

b) De acuerdo con lo enseñado en clases, describe y explica qué características es deseable que tenga un set de instrucciones con *pipelining*.

Respuesta:

Es deseable que el set de instrucciones sea de tipo RISC, y de ser necesario instrucciones CISC, estas sean descompuestas en instrucciones RISC más pequeñas. Esto debido a que una instrucción del tipo RISC son pequeñas y simples, enfocadas más en el desarrollo de software (la escritura del programa) que en desarrollo de hardware (como es el caso de instrucciones CISC).

El tener instrucciones simple permite que una instrucciones ocupe una menor cantidad de etapas dentro del pipeline, y por tanto el rendimiento de la misma sea mejor.

Distribución de puntos:

- Describir instrucciones RISC sobre CISC (no es necesario mencionarlas en nombre, pero si estar bien descritas) **(1 punto)**
- Explicación correcta (como se menciona arriba como afecta en el rendimiento y las etapas es un buen ejemplo, aunque se puede mencionar otras como el tiempo por instrucción) **(1 punto)**

c) ¿Por qué queremos *pipeline* si podemos tener varios núcleos? No omitas detalles en tu respuesta.

Respuesta:

Porque si aumentamos demasiado la cantidad de núcleos, se congestiona el acceso a memoria y el rendimiento acaba disminuyendo. Entonces eventualmente es necesario el paralelismo a nivel de instrucciones si necesitamos aumentar todavía más el rendimiento.

Distribución de puntos:

- Mencionar problemática de varios núcleos **(1 punto)**
- Mencionar paralelismo como medio clave para el rendimiento **(1 punto)**

Pregunta 6

Un nuevo dispositivo Tesla diseñado para automatización vehicular es accesible mediante I/O *memory mapped*. Este automóvil se mueve como si estuviera en un espacio cuadriculado infinito, donde cada celda puede estar vacía o contener un muro. El dispositivo tiene comandos para ser encendido, apagado, avanzar una celda hacia adelante, girar a la izquierda en 90^o y examinar lo que hay en frente.

a) Describe el mapa de memoria necesario para manejar el automóvil.**(1 pto)**

Respuesta:

Dirección	Contenido/Función Asociada	puntaje
0	Dirección ISR de manejo del Automóvil	0.3
1	Registro de comandos del Automóvil	0.35
2	Registro de estado del Automóvil	0.35
3-...	Memoria de uso libre	0.2 (bonus)

b) Define el formato de los comandos que recibirá el automóvil para llevar a cabo las acciones, y el de los datos que entregará para informar su estado. **(2 pts)**

Respuesta:

Ubicación	Comando/Estado	Valor
Reg. Comandos	Encender	255
Reg. Comandos	Apagar	0
Reg. Comandos	Avanzar	1
Reg. Comandos	Girar Izq.	2
Reg. Comandos	Examinar	4
Reg. Estado	Recién encendido	0
Reg. Estado	Nada que informar	255
Reg. Estado	Espacio libre adelante	1
Reg. Estado	Muralla adelante	2

Distribución de puntos:

- Parte como base con 2 pts
- Por cada comando faltante, **(-0.25 pts)** (hasta llegar a 0)

c) Escribe en *assembly* x86 la ISR asociada al control del automóvil, siguiendo el siguiente comportamiento: El auto avanza hasta encontrar un muro, en cuyo caso girará a la izquierda hasta encontrar un espacio vacío para avanzar, teniendo la precaución de que el automóvil no retroceda. Asume que el espacio ha sido diseñado para que el automóvil no se quede pegado girando eternamente.**(3 pts)**

Respuesta:

```
ISR_auto:
    MOV Bx, 0x0002
    CMP [Bx], 0x00
    JEQ inicializar
    CMP [Bx], 0xFF
    JEQ examinar
    CMP [Bx], 0x01
    JEQ check_retroceso
    JMP girar_izq

inicializar:
    MOV Bx, 0x0003 ; inicializamos variable en
    MOV [Bx], 0x00 ; direccion de memoria 3

examinar:
    MOV Bx, 0x0001
    MOV [Bx], 0x04
    JMP end_isr

check_retroceso:
    MOV Bx, 0x0003
    CMP [Bx], 0x02 ; verificamos direccion de retroceso
    JEQ girar_izq

avanzar:
    MOV [Bx], 0x00
    MOV Bx, 0x0001
    MOV [Bx], 0x01
    JMP end_isr

girar:
    MOV Bx, 0x0003
    ADD [Bx], 0x01
    MOV Bx, 0x0001
    MOV [Bx], 0x02
end_isr:
    IRET
```

Distribución de puntos:

- ISR_auto o equivalente **(1 pto)**
- Por cada una de las 5 subrutinas faltantes, **(0.4 pts)**
  (puntajes parciales a criterio del corrector)

Nombre de Alumno: _____ **Nº Alumno:**_____

Diagrama para la pregunta 4.a.

¡¡¡ENTREGAR ESTA HOJA ES OBLIGATORIO!!!