



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

---

IIC2343 - Arquitectura de Computadores (I/2020)

I<sub>2</sub>

Floats, circuitos secuenciales y computador básico.

## 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.

## Instrucciones

Esta interrogación es de carácter **INDIVIDUAL**. Consta de 3 preguntas que forman parte de las 9 con las que se calcula el promedio de interrogaciones al final del semestre. Para entregar deberás enviar mediante un [formulario de google](#):

- Un archivo en formato pdf por cada pregunta, con tus respuestas a la pregunta correspondiente.
- En el caso de la pregunta del código de honor, puede ser también en formato imagen.

Debes incluir todo tu procedimiento y/o desarrollo, así como todas tus fuentes en caso de que utilices material externo.

## Código de honor

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

Yo, <tu nombre>, 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>

## 3. Rango y precisión

Explica **clara y detalladamente** por qué, por la convención IEEE754, al sumar  $2^{31} + 1$  obtenemos  $2^{31}$ . Incluye los valores representados en notación científica pero en binario.

Si necesitas algo para orientarte, observa el siguiente código de C y ejecútalo<sup>1</sup> (tienes el archivo aparte en la misma carpeta del enunciado) y mira el output obtenido.

```
1 #include<stdio.h>
2 #include<math.h>
3
4 int main() {
5     float x = (float)pow(2, 31); // hago que me salga como float
6     float y = x + 1.0; // forzamos a que el resultado de la operación
7                         // sea un float. Si no lo hacemos, sería un
8                         // double y el ejemplo no funcionaría.
9
10    if (x == y) // Comparamos los dos valores
11        printf("SON IGUALES! :o\n");
12    else
13        printf("SON DISTINTOS! :)\n");
14    return 0;
15 }
```

La función `pow` de C permite elevar un número a otro y retorna el resultado. Para esto utiliza el tipo de dato `double`. El código está bueno. El output no es resultado de ningún *bug*.

---

<sup>1</sup>Imprime "SON IGUALES! :o".

## 4. Poder de cómputo

A grandes rasgos, diremos que una instrucción se puede ejecutar en un solo ciclo cuando se cumplen las siguientes condiciones:

- Existen las conexiones y componentes necesarios.
- No se está usando algún componente dos (o más) veces para cosas distintas.
- Si se va a escribir en dos componentes de memoria (RAM y registros) distintos, estos deben escribirse en paralelo, nunca en serie.
- La última cosa que puedes hacer con un dato es escribirlo, luego de eso estás en el siguiente ciclo.

Indica cuales de las siguientes operaciones se pueden hacer en un solo ciclo en el computador básico y cuáles no. Justifica a través del circuito del computador básico. Si no es posible realizar alguna de las instrucciones, indica cómo se debe modificar el computador básico para que sea posible. Sé explícito con las señales indicando sus valores y con las conexiones que añadas, no omitas nada.

I.  $\text{Mem}[\text{Lit}] \ += \ 1$

II.  $\text{Mem}[\text{Lit}] \ = \ A \ + \ B$

III.  $\text{Mem}[\text{Lit}] \ = \ 0$  y  $\text{CMP } 1, (\text{Lit})$ : guardo un 0 en memoria y comparo el resultado de 1 con lo que había antes en esa celda de memoria.

Podrás encontrar el diagrama del computador básico con hasta saltos junto a este enunciado para que puedas editarlo con [draw.io](https://draw.io).

## 5. Ruta de acceso a memoria

Dibuja el circuito interno de la memoria de datos del computador básico, identificando las señales y datos de entrada y salida. Explica detalladamente por qué tu circuito es correcto a través de las tablas de verdad de los *flip flops D* y del circuito que diseñes (o sus componentes pero detallando su funcionamiento conjunto). Debe ser genérico para una memoria de  $n$  *bytes*, trabaja la unidad de almacenamiento como una memoria RAM, no como registros individuales.

Tu circuito debe poder:

- Seleccionar la dirección a leer.
- Seleccionar la dirección a escribir.
- Manejar las señales involucradas adecuadamente.

# SOLUCIONES

### 3. Rango y precisión

Primero, recordemos que cuando el exponente es distinto de cero, al frente de la mantisa siempre hay un “1,” implícito llamado *leading bit*, entonces si la mantisa fuera, por ejemplo, 001000000000000000000000 y el exponente fuera 3, el número a representar realmente es  $1,001000000000000000000000 \cdot 2^3$ .

El número  $2^{31}$  se puede representar en notación científica como

$$1,00000000000000000000000000000000.2^{31}$$

Donde los números en verde son la mantisa.

Ahora bien, si le sumamos 1, obtendríamos

[illegible]

Y como se aprecia, el cambio está fuera de la mantisa (bits en rojo) mostrando que el 1 no es lo suficientemente grande como para verse reflejado en la mantisa y por eso es que al final la suma no afecta al resultado y  $2^{31} + 1 = 2^{31}$ . Lo que tenemos es que el exponente es tan grande que la pérdida de precisión se hace evidente.

Corrección - 6 puntos.

- Explicar que el 1 queda fuera de la mantisa por la diferencia de los exponentes (pérdida de precisión): 4 puntos.
- **Mostrar** que el “+1” no cabe en la mantisa: 2 puntos.

## 4. Poder de cómputo

Las señales están en los dibujos (los dibujos para los casos I y II no son necesarios).

- I. Sí se puede hacer sin modificaciones, es cosa de activar la señal W, seleccionar el 1 en el mux A, la entrada DataOut en el mux B y la operación ADD en la ALU. Por último, la salida de la ALU para la entrada del DataIn. En el mux address selecciono el literal que sale de la memoria de instrucciones.

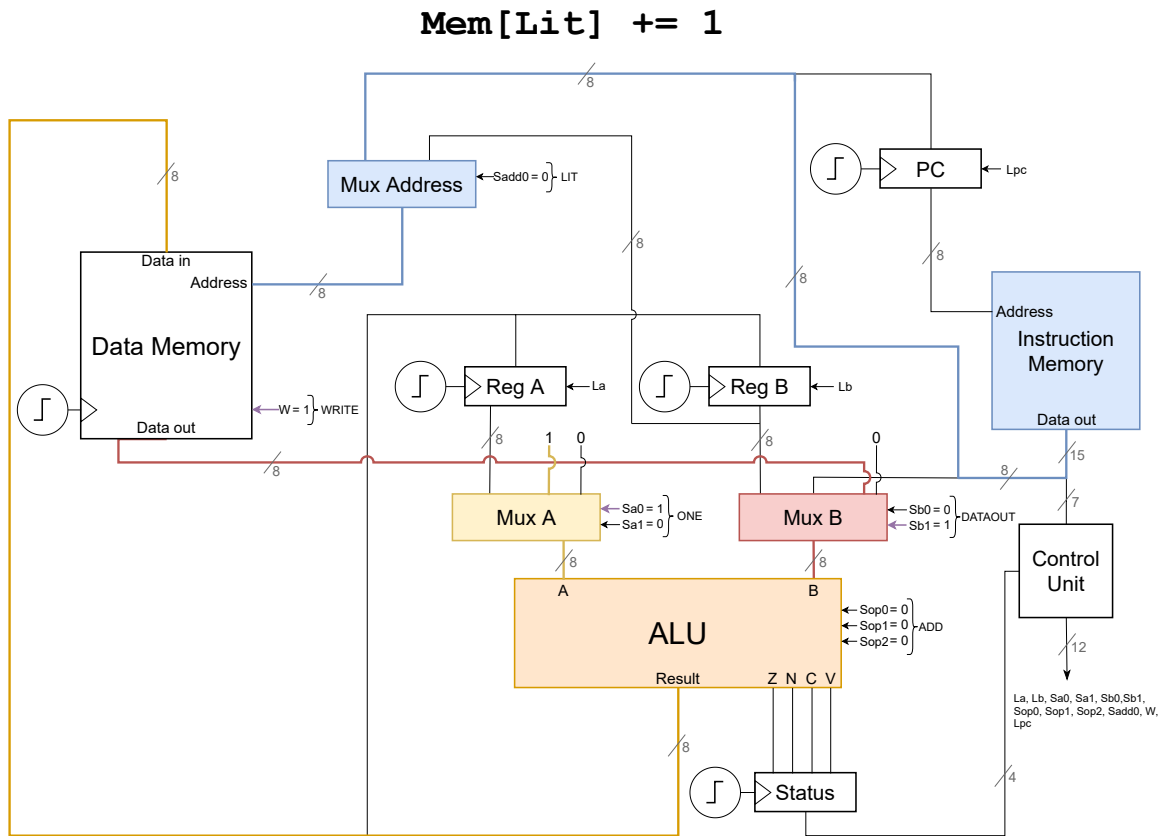


Figura 1: Las señales están en el dibujo.

II. Sí se puede, mantengo todo igual de la instrucción anterior, salvo que tengo que seleccionar el registro A en el mux A en vez del 1 y el B en el mux B en vez del DataOut.

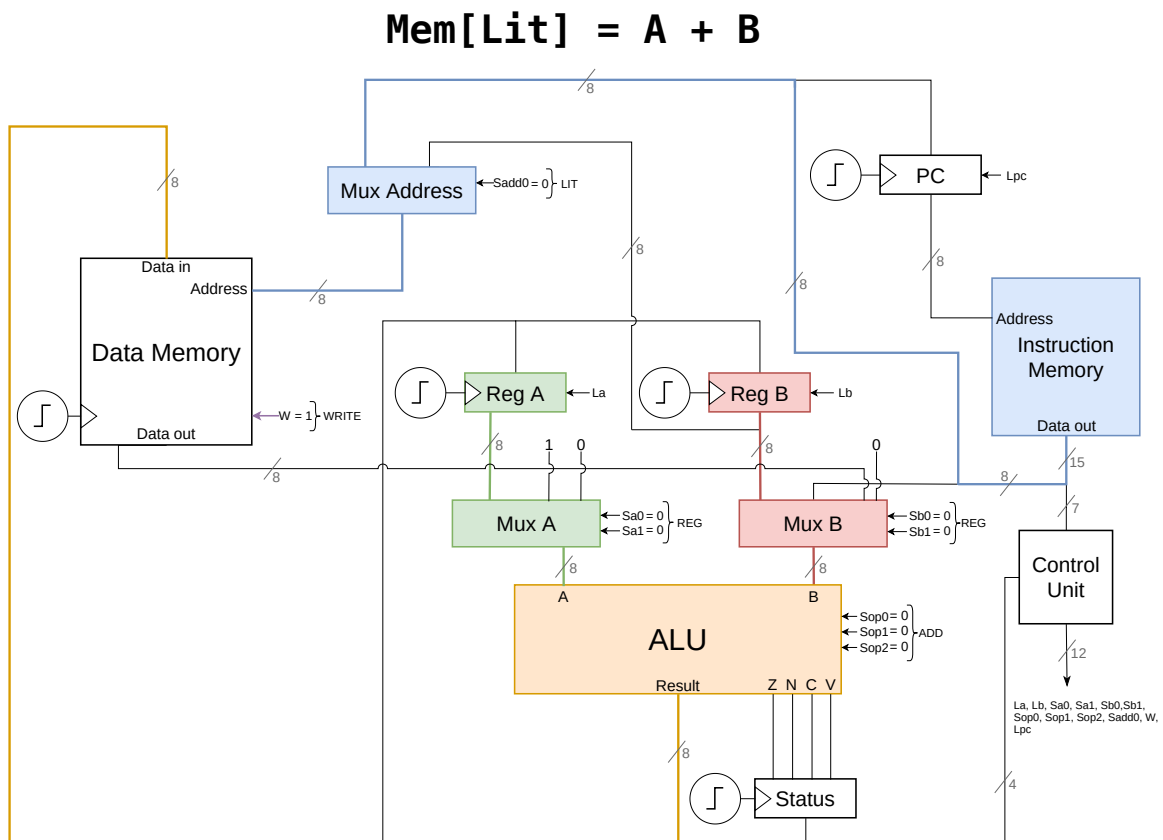


Figura 2: Las señales están en el dibujo.

III. No se puede, ya que necesito la ALU tanto para la comparación como para cargar el 0 en memoria. Tenemos que ponerle un mux datain con su señal de control y un 0 como segunda opción.

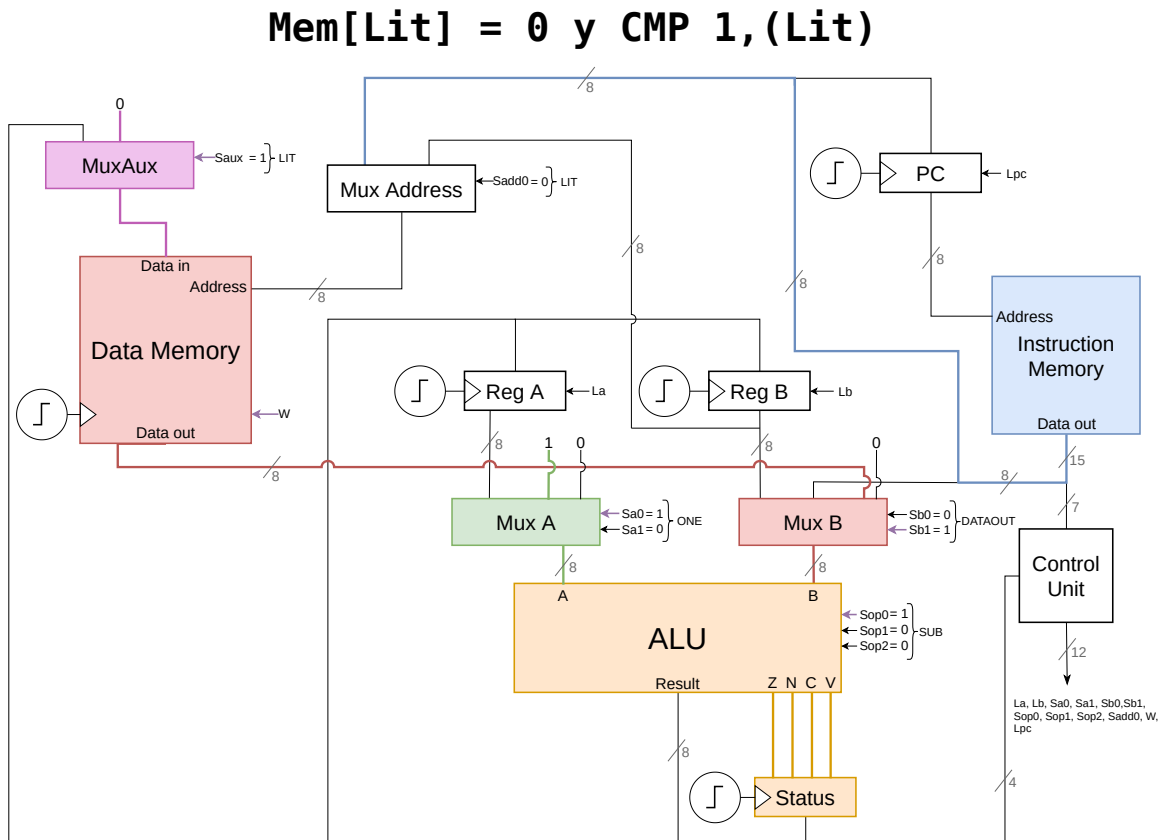


Figura 3: Las señales están en el dibujo.

## Corrección - 6 puntos

Las señales deben ser numéricas.

- Caso I: 1,5 puntos.
- Caso II: 1,5 puntos.
- Caso III: 3 puntos; 1,5 modificación, 1,5 señales.

## 5. Ruta de acceso a memoria

Para *seleccionar* el valor que sale por DataOut es cosa de tener un mux que use la dirección de memoria como un conjunto de señales que dirigirán el valor que deseamos hacia la salida.

Para *escribir*, necesitamos que la señal W sólo llegue a la dirección de memoria específica que deseamos leer, siendo 0 para todas las demás, por lo que usaremos un **demux** para este propósito, donde la dirección de memoria servirá de señales de control para elegir la salida adecuada.

En lo que respecta al DataIn, podemos enviarlo directamente a todas las direcciones de memoria sin comprometer la integridad de la información, gracias a que la señal W solo llegará a una de las direcciones.

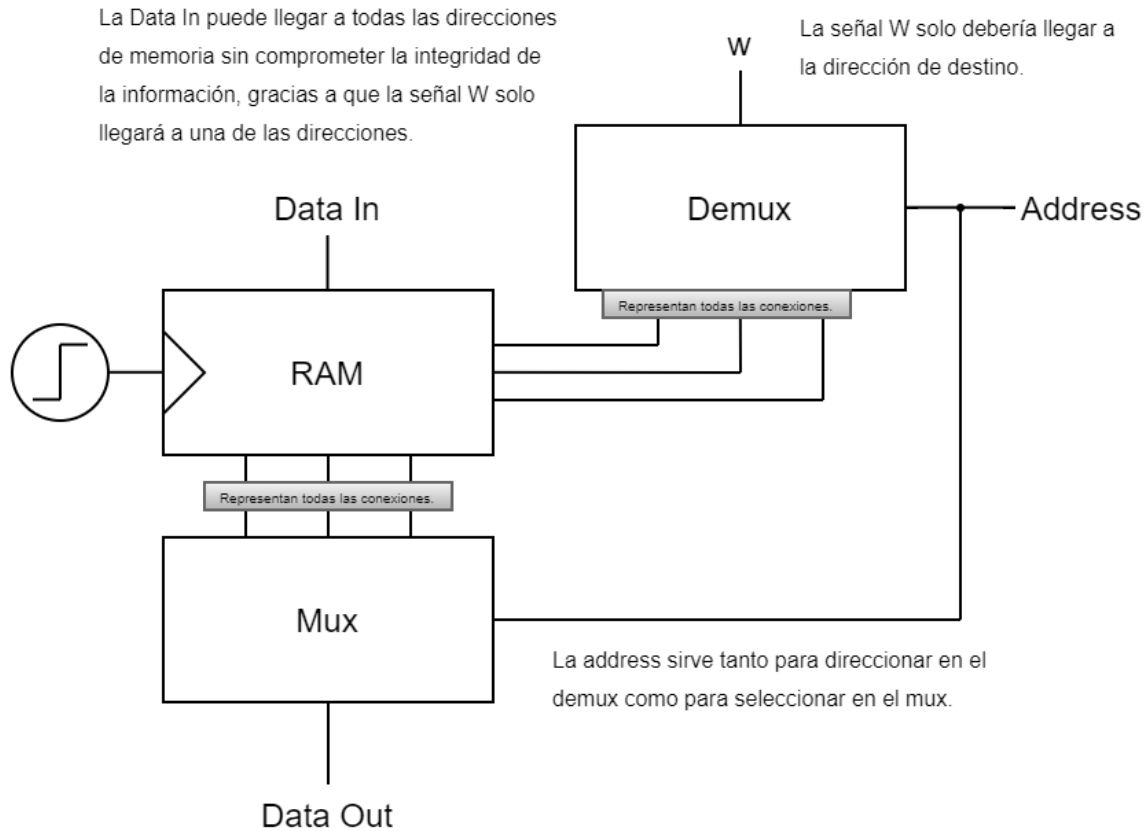


Figura 4: Circuito interno de la *instruction memory*.

### Corrección - 6 puntos

Si el circuito no calza con el propuesto pero está correcto (funciona), los puntajes se adaptan.

- Circuito: 3 puntos.
- Explicación del mux: 1 punto.
- Explicación del demux: 1 punto.
- Explicación de la RAM a través de los *flip flops*: 1 punto.  
Tanto en relación a la señal W como al DataIn.