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

---

**IIC2343 - Arquitectura de Computadores (I/2020)**

**Ayudantía 2**

***Flats*, circuitos lógicos secuenciales y computador básico**

# 1. *Floats*

Describe y respalda de manera teórica la solución para la pregunta 5 de la tarea 1.

# 2. Circuitos secuenciales

Dibuja el circuito interno del *program counter* y analiza: ¿por qué funciona?

# 3. Computador básico

Describe cómo son soportadas las siguientes instrucciones en el computador básico:

1. `MOV A,B`
2. `CMP A,(Dir)`

# 4. Ejercicio propuesto (sin solución en este enunciado)

Con los registros usados en la solución de la pregunta 2, tenemos que se escriben de manera automática al llegar al flanco de subida siempre. Modifícalos para añadirles una señal que permita controlar si queremos que se escriban o no.

# Soluciones

## 1. *Floats* (solución)

La convención IEEE754 establece que tomaremos los 32 bits y usaremos el más significativo como bit de signo, los siguientes 8 como exponente (desplazados en 127) y los siguientes 23 bits como mantisa. Además, si el número está normalizado (exponente no es 0) entonces hay un 1 implícito delante de la mantisa.

Dicho esto, podemos situarnos en casos:

### 1.1. Racional positivo mayor a 0

Si tenemos un número racional positivo y mayor a cero, podemos intuir que si aumentamos el valor de la mantisa en 1, obtendremos el sucesor del número, pero si la mantisa ya tenía su valor máximo, la pasamos a 0 y lo que aumentamos en 1 es su exponente. Esto equivale a interpretar el patrón de bits como un entero y simplemente sumar 1.

Por otro lado, sabemos que si decrementamos en 1 la mantisa tendremos el antecesor del número, y si esta tiene un valor de 0, la pasamos a `0x7FFFFF` y lo que decrementamos es el exponente. Esto equivale a interpretar el patrón de bits como un entero y simplemente restar 1.

### 1.2. Racional negativo menor a -0

Si por un momento ignoramos el signo y aumentamos el valor numérico (exponente + mantisa) nos dará el sucesor como si fuera un positivo, pero al tener que es negativo, este es en realidad el antecesor del número.

Y luego, si decrementamos el valor numérico, obtenemos esta vez el sucesor del número.

Recuerda que (esto en los enteros) -3 es sucesor de -4 y a la vez antecesor de -2.

### 1.3. Tabla resumen

| Dato | Antecesor | Sucesor |
|---|---|---|
| $x > 0$ | $bits(x) - 1$ | $bits(x) + 1$ |
| $x < -0$ | $bits(x) + 1$ | $bits(x) - 1$ |
| $x = 0$ | $-0$ | $bits(x) + 1$ |
| $x = -0$ | $bits(x) + 1$ | $0$ |

El código está en el repositorio de github, en la carpeta `/Tareas/T1/Solución/`.

## 2. Circuitos secuenciales (solución)

Un *program counter* es un circuito secuencial que almacena un valor (en el caso del computador básico es de 8 bits) y en cada ciclo entrega el valor que almacena (diremos que a un registro X cualquiera) e incrementa en 1 el valor que almacena.

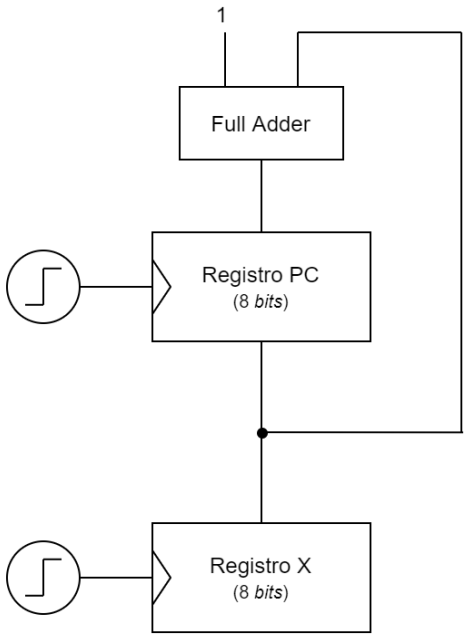



Figura 1: *program counter*.

Hay dos puntos cruciales en los que hay que tener cuidado a la hora de determinar si esto es posible:

1. ¿Cómo puede sobreescribir su valor a la vez que entrega el valor viejo?
2. ¿Cómo alcanza a escribir en dos registros en el mismo ciclo del *clock*?

Esto se responde si miramos la tabla de verdad de los *flip flops D*:

| C | D | Q |
|---|---|---|
| 0/1/↓ | x | Q |
| ↑ | 0 | 0 |
| ↑ | 1 | 1 |

Q es el valor que sale, C es la señal del *clock* y D es el valor que le damos como "entrada". Notemos que el valor de la salida del registro no va a cambiar hasta que lleguemos al flanco de subida.

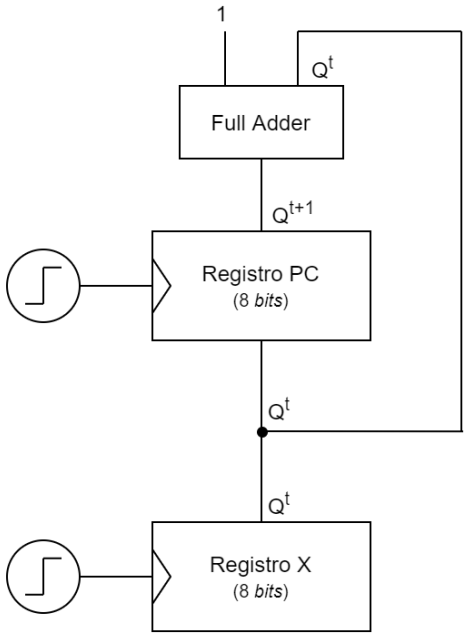



Figura 2: *program counter* con la propagación de los valores.

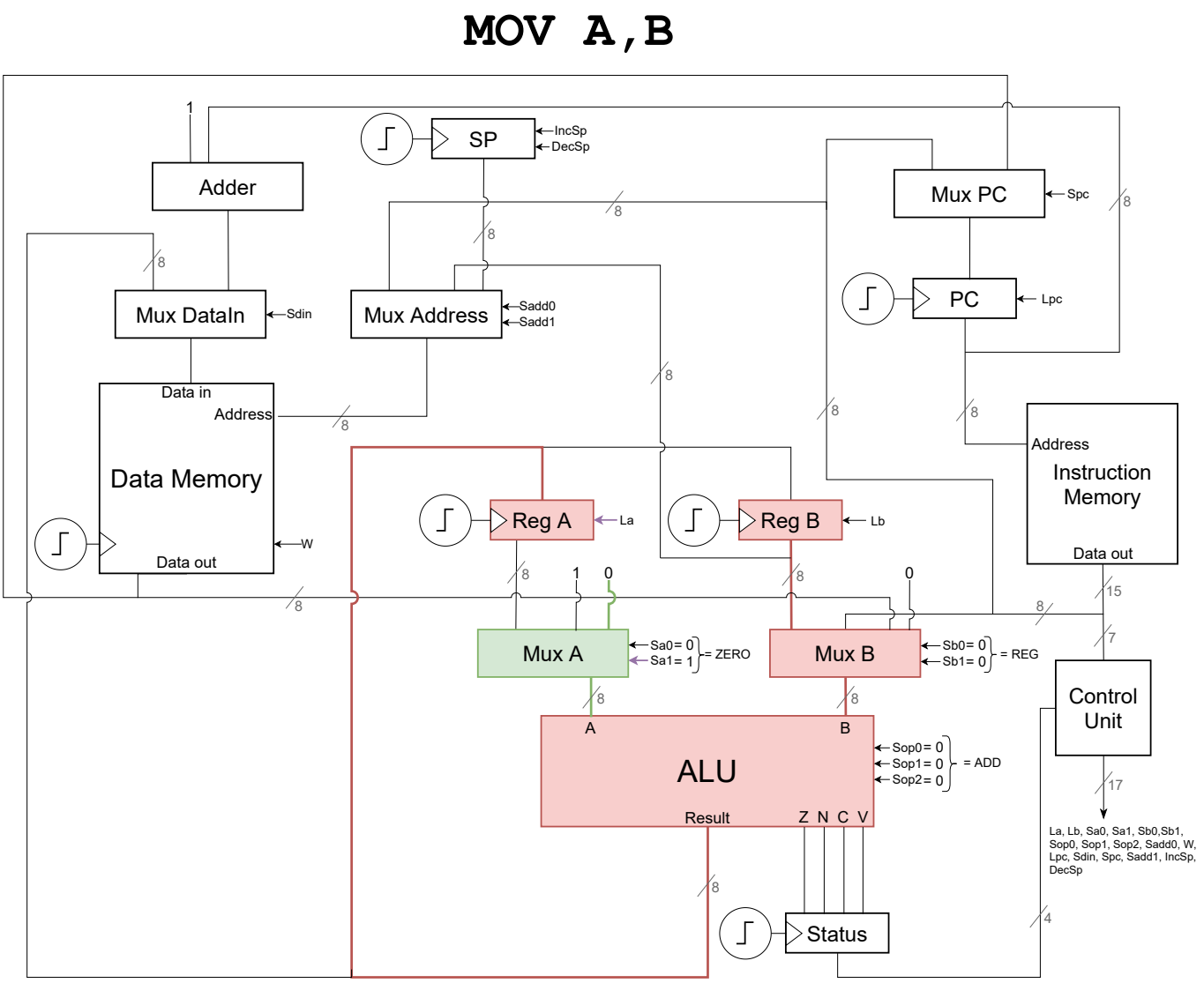
Si $Q^t$ es el dato del registro PC en el tiempo t, $Q^{t+1}$ es el dato del registro PC en el tiempo t + 1. Lo que está pasando es que el valor que se está propagando por el circuito antes de llegar al momento del flanco de subida es $Q^t$ y, cunado finalmente llegamos al flanco, el valor que intenta "entrar" en nuestro registro X es $Q^t$, mientras que el que intenta entrar en el registro PC es $Q^{t+1}$. Almacenándose estos valores los registros.

Debemos recordar algo baastante importante, además: el concepto de tiempo de propagación.

Cuando tenemos un circuito real, las señales y valores tardan en recorrerlo y llegar a destino. La frecuencia del *clock* se ajusta en base a esta información. Si la frecuencia es muy alta (va muy rápido), podría pasar que los valores no tengan tiempo para propagarse y el circuito se comporte de forma errática.

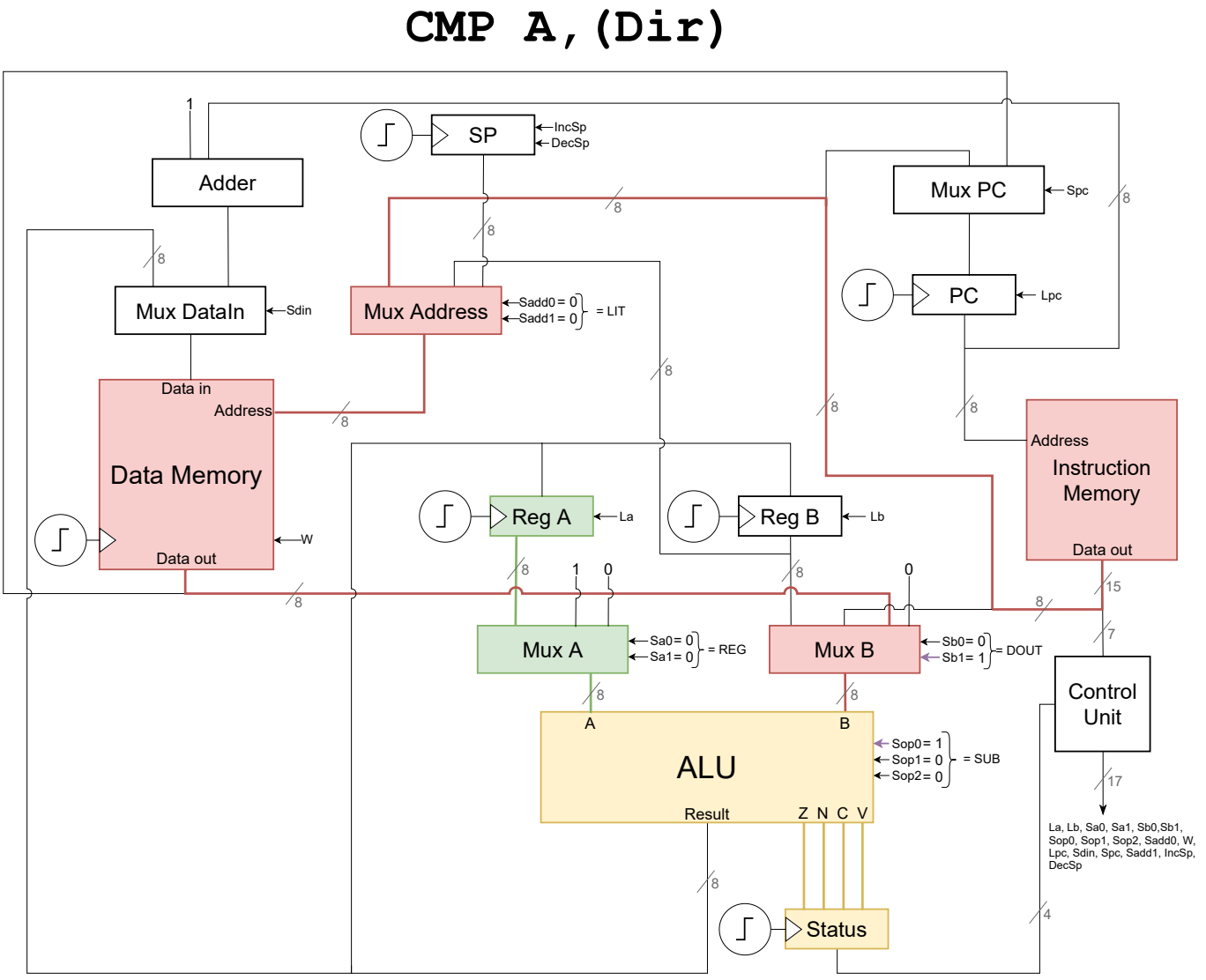
# 3. Computador básico (solución)

## 3.1. MOV A,B




Del registro B sale el valor y es seleccionado por el mux B, mientras que en el mux A seleccionamos el valor 0. Con esto seteamos la operación en ADD y al calcular 0 + B, el valor de la salida es B, que será escrito en el registro A.

### 3.2. CMP A, (Lit)




Del registro A sale el valor y es seleccionado por el mux A, mientras que de la *Instruction Memory* sale el literal que va hasta el mux address y allí es seleccionado para usarse como dirección para la *Data Memory*, donde se lee la posición `Lit` de la memoria y el valor allí almacenado sale y llega hasta el mux B, donde se selecciona y entra por la entrada B de la ALU, cuya operación seleccionada es SUB para la resta y se guardan los *condition codes* en el registro *status*.