IIC2343 – Arquitectura de Computadores

Pauta Control 1

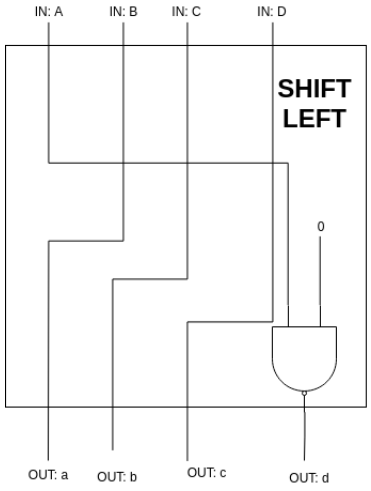
Profesor: Yadran Eterovic

1. Lógica Digital (3 puntos)

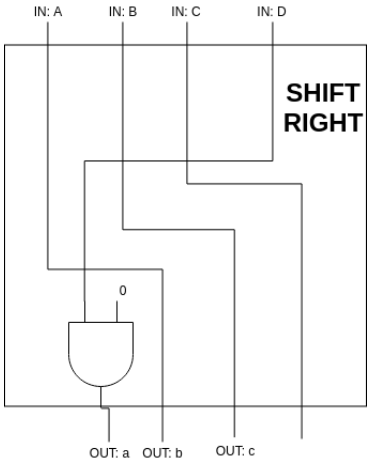
1. Diseñe un componente de 4 bits que según una señal de control multiplique o divida el valor de 4 bits de entrada por 2. Sí, solo por 2. Hint: Los multiplexores existen **(1.5 puntos)**

   Respuesta:

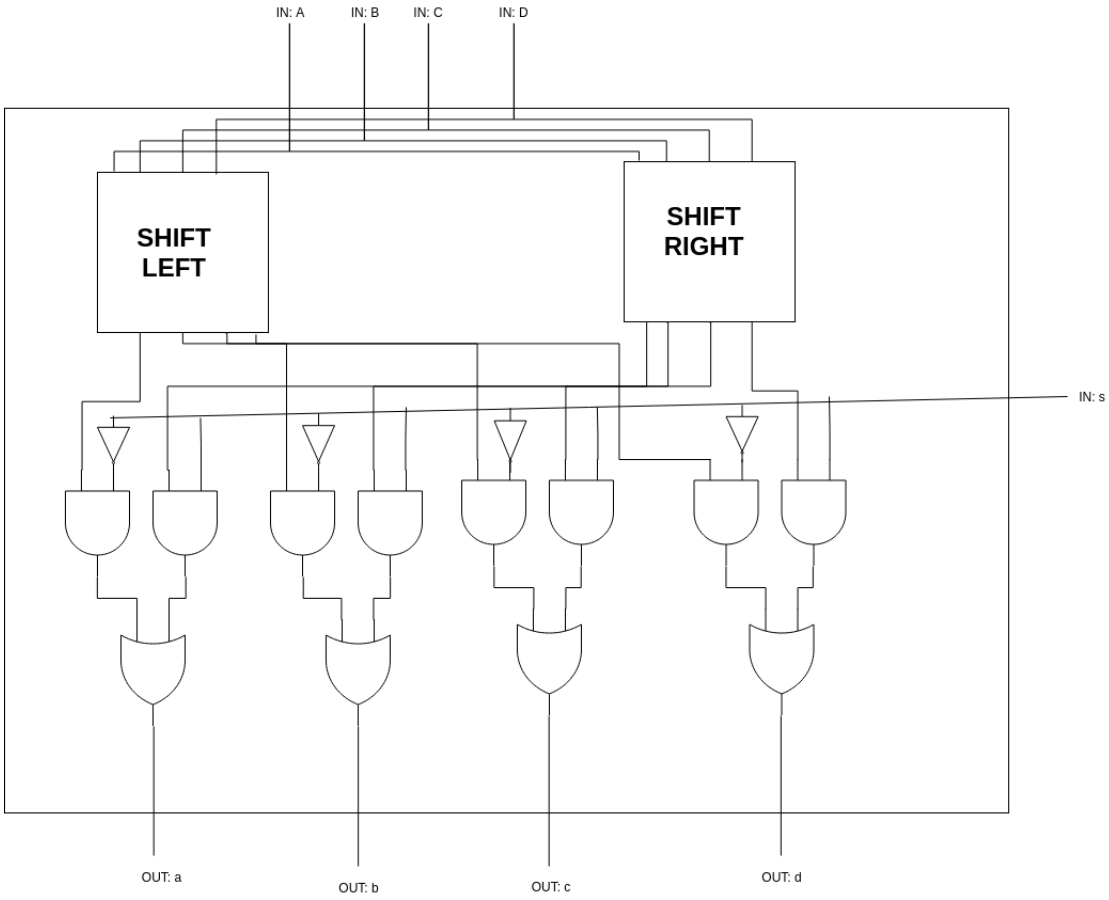
   Primero debemos construir un componente que multiplique por dos, y eso lo podemos conseguir con un shift-left. (**NOTA:** para la construcción se asumirá misma cantidad de bits de inputs que de output, y no se hará uso del carry de la multiplicación ni la división). El shift-left se describe a continuación:



   En segundo lugar debemos construir un componente que divida por dos, y eso lo podemos conseguir con un shift-right, como se describe en la siguiente imagen:



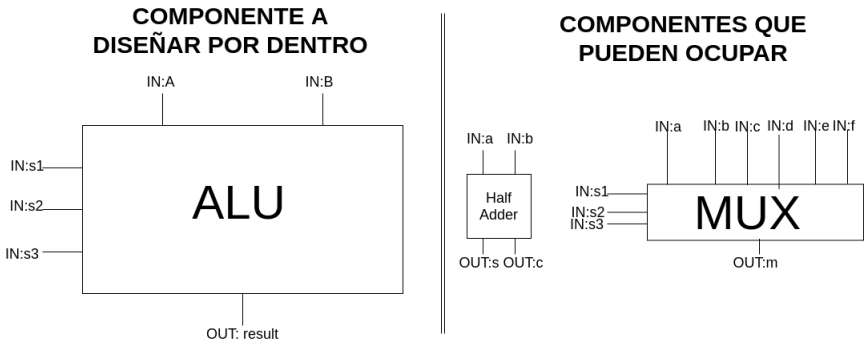
Finalmente conectamos ambos componentes a un multiplexor (**NOTA:** no es necesario hacer una descripción exhaustiva del multiplexor como en la imagen, con la mención de este basta).



Distribución de puntos:

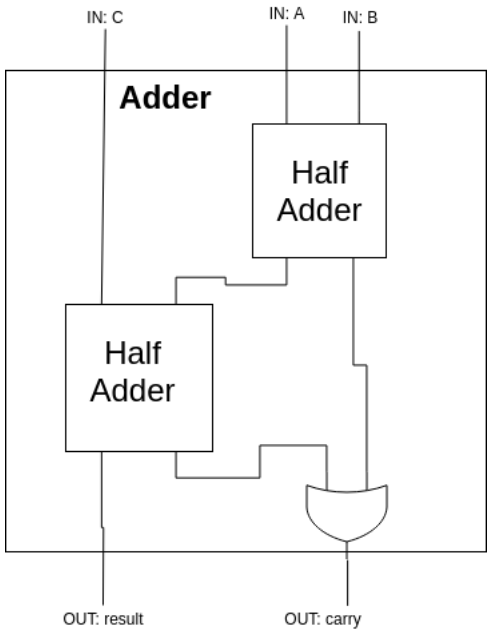
- 0.25 pts. por saber que multiplicar es hacer un shift left.
- 0.25 pts. por saber que dividir es hacer un shift right.
- 0.5 pts si shift left funciona.
- 0.5 pts si shift rigth funciona.
- -0.2 pts si se rige mal la operación.

2. Construya el interior de una ALU de 1-bit que acepte como input entradas A y B. La ALU debe ser capaz de ejecutar las operaciones AND, OR, XOR entre los dos valores de entrada. También debe ser capaz de realizar NOT sobre la entrada A. Finalmente, debe poder sumar y restar entre ambas entradas (el *carry* se ignora en la construcción de esta ALU). Para más información se tiene el siguiente esquema con los componentes que puede asumir poseer y no construir.**(1.5 puntos)**

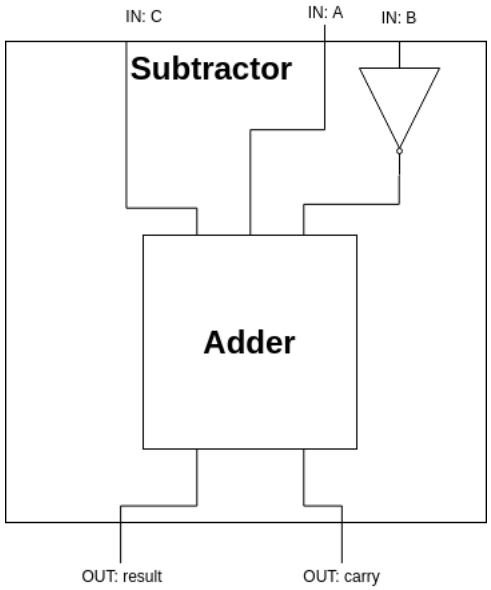


Respuesta:

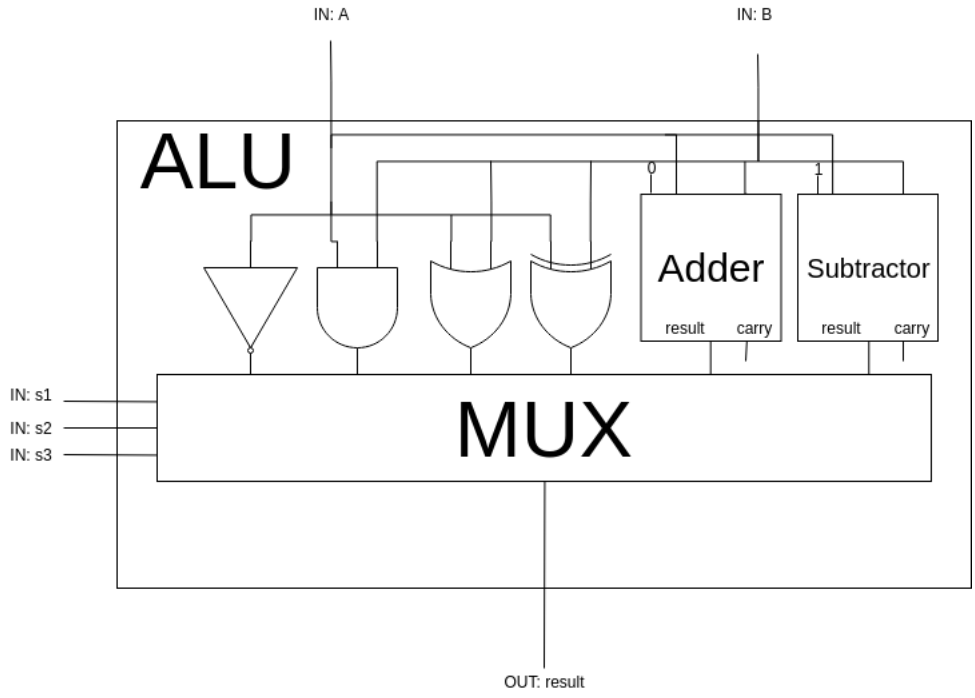
Para la construcción de nuestra ALU de un bit necesitamos un Full Adder como el que se muestra en la imagen:



También necesitamos un componente para restar (**NOTA:** se puede hacer ambos en un solo componente.)



Finalmente con la ayuda del multiplexor y agregando y conectando las compuertas lógicas correctamente tenemos el siguiente diagrama de la ALU pedida:



Distribución de puntos:

- 0.5 pts correcta implementación instrucciones AND, OR, XOR, NOT.
- 0.4 pts correcta implementación de la suma.

- 0.4 pts correcta implementación de la resta.
- 0.2 pts correcta selección de la salida.

2. Assembly (3 puntos)

1. Utilizando el lenguaje *assembly* visto en clases (ver Anexo), escriba un programa que dado un valor k ($k \geq 0$), guardado en la dirección **0x0000** de la memoria, guarde en las direcciones siguientes los k primeros números de *Fibonacci* (partiendo en 0). Considere los 32 registros de la arquitectura MIPS, respetando la función de cada uno de ellos, y asuma que estos poseen la cantidad de bits necesarios para representar el valor k y los k primeros números de la serie de *Fibonacci*.

   Serie de Fibonacci:

   $$f_k = f_{k-1} + f_{k-2}$$

Respuesta Propuesta 1:

```
# $t0: contador; $t1: k ----------------------------
    addi $t0, $zero, 0     # $t0: cantidad de numeros generados
    lw   $t1, 0($zero)     # $t1: cantidad de numeros a generar
# -------------------------------------------------
# cargar los primeros valores a memoria
    addi $t2, $zero, 0     # $t2 = fib(1) = 0
    addi $t3, $zero, 1     # $t3 = fib(2) = 1
    beq  $t0, $t1, end     # chequear si $t0 == $t1
    sw   $t2, 4($zero)     # Memory[0 + 4] = $t2
    addi $t0, $t0, 1       # $t0 += 1
    beq  $t0, $t1, end     # chequear si $t0 == $t1
    sw   $t3, 4($zero)     # Memory[0 + 8] = $t3
    addi $t0, $t0, 1       # $t0 += 1
# -------------------------------------------------
# definir variables
    addi $t5, $zero, 4     # $t5 = ubicacion de k - 2
    addi $t6, $zero, 8     # $t6 = ubicacion de k - 1
    addi $t7, $zero, 12    # $t7 = (proxima) ubicacion de k
# -------------------------------------------------
loop:
    beq  $t0, $t1, end      # todos los n meros generados si $t0 == $t1
    # generar n mero y guardarlo
    lw   $t2, 0($t5)        # $t3: valor almacenado en k - 2
    lw   $t3, 0($t6)        # $t2: valor almacenado en k - 1
    add  $t4, $t2, $t3      # $t4 = $t2 + $t3 (k = (k - 2) + (k - 1))
    sw   $t4, 0($t7)        # Memory[$t7] = $t4
    # ---------------------------------------------
    # sumar 1 a contadores
    add  $t5, $t5, 4        # $t5 += 4 (posicion k - 2)
    add  $t6, $t6, 4        # $t6 += 4 (posicion k - 1)
```

```
    add   $t7, $t7, 4        # $t7 += 4 (posicion k)
    add   $t0, $t0, 1        # $t0 += 1 (contador)
    # ------------------------------------------
    j loop
end:
    # listo!
```

- 0.25 pts. por Manejar $k = 0$.
- 0.25 pts por Manejar $k = 1$.
- 1 pt. por guardar k valores en memoria (terminar oportunamente).
- 1 pt. por guardar los valores correctos.
- 0.5 pts. por uso correcto de assembly.