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

IIC1001 — Algoritmos y Sistemas Computacionales — 2024-1

# Interrogación 1 - Solución

Miércoles 10-Abril-2024

**Duración:** 120 minutos
**Responder cada pregunta en una hoja separada.**

1. **[20p]** Representaciones y operaciones númericas:

   1.1) **[12p]** Complete la siguiente tabla usando conversiones numéricas. No necesitan detallar el procedimiento, pero si lo hacen pueden y el resultado no es correcto, podrían obtener puntaje parcial:

| decimal | binario | hexadecimal |
|---|---|---|
| 1340 | | |
| 4096 | | |
| 3072 | | |
| 3392 | | |
| | 0110 0001 | |
| | 0101 1100 | |
| | 1001 1001 1100 | |
| | 1100 0000 1001 | |
| | | CA |
| | | 7E |
| | | 822 |
| | | AD3 |

**R.** 1 pto por cada línea correcta. 0,5 pto si respuesta es incorrecta pero hay un error de procedimiento leve.

| decimal | binario | hexadecimal |
|---|---|---|
| 1340 | **101 0011 1100** | **53C** |
| 4096 | **1 0000 0000 0000** | **1000** |
| 3072 | **1100 0000 0000** | **C00** |
| 3392 | **1101 0100 0000** | **D40** |
| **97** | 0110 0001 | **61** |
| **92** | 0101 1100 | **5C** |
| **2460** | 1001 1001 1100 | **99C** |
| **3081** | 1100 0000 1001 | **C09** |
| **202** | **1100 1010** | CA |
| **126** | **0111 1110** | 7E |
| **2082** | **1000 0010 0010** | 822 |
| **2771** | **1010 1101 0011** | AD3 |

1.2) **[8p]** Efectúe las siguientes operaciones de números hexadecimales **explicitando su procedimiento**. Puede hacerlo en notación hexadecimal o binaria, pero **no es válido convertirlas a decimal** (salvo si quiere comprobar un resultado):

**R.** Se muestran los resultados tanto en binario como en hexadecimal, pero solo se requería usar una de ellas. 1 pto por cada respuesta correcta. 0,5pto si el resultado está incorrecto pero hay un error leve de procedimiento. El resultado final debe estar en hexadecimal. Si solo está en binario, se considera "error leve".

a) `0x32` + `0x0E`

**R.**

| | |
|---|---|
| `0x32` | 0011 0010 |
| `0x0E` | 0000 1110 |
| `0x40` | 0100 0000 |

b) `0x333` + `0xA2`

**R.**

| | |
|---|---|
| `0x333` | 0011 0011 0011 |
| `0x0A2` | 0000 1010 0010 |
| `0x3D5` | 0011 1101 0100 |

c) `0xD315` + `0x8`

**R.**

| | |
|---|---|
| `0xD315` | 1101 0001 0101 |
| `0x0008` | 0000 0000 1000 |
| `0xD31D` | 1101 0001 1101 |

d) `0x1111` + `0x9999`

**R.**

| | |
|---|---|
| `0x1111` | 0001 0001 0001 |
| `0x9999` | 1001 1001 1001 |
| `0xAAAA` | 1010 1010 1010 |

e) `0x7` − `0x5`

**R.** Para ejecutar esta suma, se debe representar los números usando el primer bit para el signo, y el resto para el valor absoluto. Los números negativos deben representarse como complemento de 2.
En el caso `0x5`, el valor binario es 0101. En complemento de 2, es 1011.

| | |
|---|---|
| `0x7` | 0111 |
| `-0x5` | 1011 |
| `0x2` | 0010 |

f) `0x10` − `0x4`

**R.**
Para `0x04`, su valor binario es 0000 0100, y su complemento de 2 es 1111 1100.

| | |
|---|---|
| `0x10` | 0001 0000 |
| `-0x04` | 1111 1100 |
| `0x0C` | 0000 1100 |

g) `0x17` − `0x22`

**R.**
Para `0x22`, su valor binario es 0010 0010, y su complemento de 2 es 1101 1110.

| | |
|---|---|
| `0x17` | 0001 0111 |
| `-0x22` | 1101 1110 |
| `-0x0B` | 1111 0101 |

El resultado 1111 0101 es negativo. Para obtener el valor positivo, aplicamos complemento de 2, y se obtiene 0000 1011 que es $B$. Por lo tanto el resultado hexadecimal es `-0x0B`.

h) `0x40EC` − `0x403B`

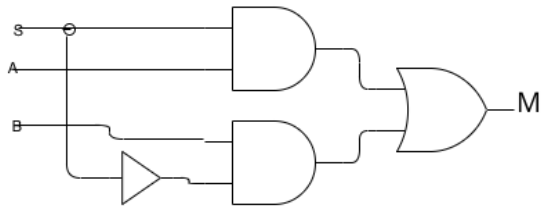
**R.**
Para `0x403B`, su valor binario es 0100 0000 0011 1011, y su complemento de 2 es 1011 1111 1100 0101.

| | |
|---|---|
| `0x40EC` | 0100 0000 1110 1100 |
| $-$`0x403B` | 1011 1111 1100 0101 |
| `0x00B1` | 0000 0000 1011 0001 |

En este caso se puede notar que los dos primeros dígitos de cada número hexadecimal son iguales, y que el problema se puede reducir a la diferencia $\texttt{0xEC} - \texttt{0x3B}$. Más aún, se puede calcular cada diferencia fácilmente. En la columna de la derecha $\texttt{0xC} - \texttt{0xB} = \texttt{0x1}$, y en la segunda columna, $\texttt{0xE} - \texttt{0x3} = \texttt{0xB}$.

2. **[12p]** Para las siguientes construcciones con celdas lógicas:

   2.1) **[8p]** Construir una tabla con todas las salidas posibles. Los valores de entrada son $S, A, B$, y la salida es $M$.




   Bonus (2pto): ¿Qué rol tiene el valor de entrada $S$ respecto a $M$? Por ejemplo, qué pasa con $M$ cuando $S = 0$ y cuando $S = 1$.

   **R.**

   La tabla que se debe entregar es la siguiente. 1 pto por cada línea correcta.

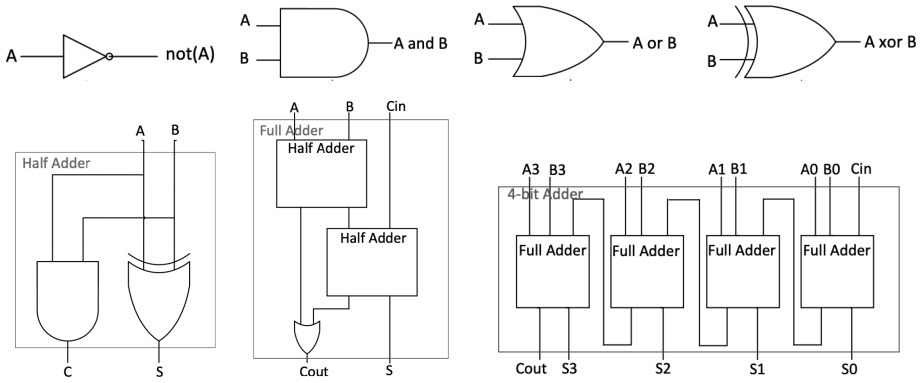
| $S$ | $A$ | $B$ | $M$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

   Opcionalmente se pueden incluir valores intermedios para facilitar el cálculo.

| $S$ | $A$ | $B$ | $S$ AND $A$ | NOT $S$ AND $B$ | $M$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |

   *Bonus*: 2pts. Lo que se puede notar es que, cuando $S = 0$, $M$ entrega el mismo valor que $B$. Por otro lado, cuando $S = 1$, $M$ entrega el mismo valor que $A$. Adicionalmente se podría mencionar que $S$ permite "seleccionar" si $M$ refleja el valor de $B$ ó el de $A$.

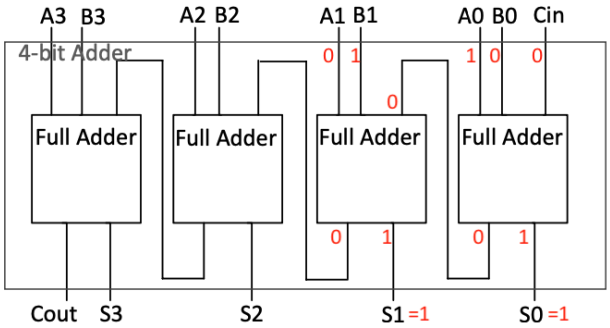
2.2) **[4p]** Ejecute la suma de los valores hexadecimales: $0x1$ y $0x2$ usando un full-adder de 2 bit. Muestra la entrada y salida del full-adder. Como referencia se agregan los diagramas vistos en clases.




**R.** La suma binaria que se espera debería ser equivalente a esta:

| | | |
|---|---|---|
| `0x1` | $A_1A_0$ | 01 |
| `0x2` | $B_1B_0$ | 10 |
| `0x3` | $S_1S_0$ | 11 |

Para usar el *full adder* con de 1 bit, necesitamos dos *full adder*. Los valores de cada bit están dados por $A = A_1A_0 = 01$, y $B = B_1B_0 = 10$. El valor del *carry* de entrada debe $C_{in} = 0$. El *full-adder* queda de la siguiente manera:




Opcionalmente se puede usar el *full-adder* de 4 bit, y completar las otras entradas ($A_3$, $A_2$, $B_3$, $B_2$) con 0

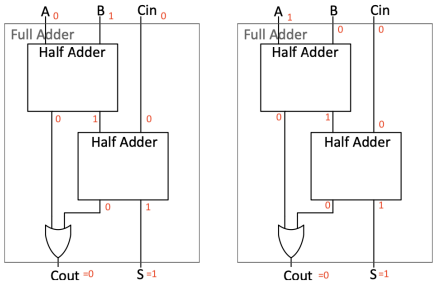
1 pto por indicar correctamente las entradas $A_1$ y $A_0$

1 pto por indicar correctamente las entradas $B_1$ y $B_0$

1 pto por obtener correctamente las salidas $S_1$ y $S_0$

1 pto por especificar que $C_{in}$ es 0.

Opcionalmente se puede agregar las salidas de los *full-adder* de 1 bit:



3. **[16p]** Una manera de codificar palabras es escribir una secuencia en que primero se escribe la cantidad de letras que tiene la palabra ($L$), y a continuación $L$ pares $(C_i, P_i)$, donde cada $C_i$ es una letra y $P_i$ es la posición donde va esa letra. Las letras se escogen **primero en orden alfabético**. Si hay una letra que aparece en mayúscula y en minúscula, la mayúscula va primero.

   Por ejemplo, la palabra `Hola` se escribe mediante la secuencia: $4, \texttt{a}, 3, \texttt{H}, 0, \texttt{l}, 2, \texttt{o}, 1$. Esto es porque hay $4$ letras, a continuación el par $(\texttt{a}, 3)$ indica que la letra `a` va en la posición 3, luego el par $(\texttt{H}, 0)$ indica que la letra `H` va en la posición 0, etc.

   Construya un archivo, escrito en bytes en representación hexadecimal. Cada línea debe tener máximo 12 byte, y luego se escribe en la línea siguiente.

   - 9 Byte, donde cada byte contiene un **caracter** ASCII del texto `I1-2024-1`
   - 1 Byte con el valor numérico $(23)_{10}$ si usted está en la sala B23, o el valor numérico $(24)_{10}$ si usted está en la sala BC24.
   - 1 Byte con el valor numérico que representa la cantidad de Byte (el valor $X$) usados en la siguiente parte.
   - $X$ Byte con la codificación de la palabra `FiuYPangUI` de acuerdo a lo explicado. Cada elemento de la secuencia debe almacenarse en 1 Byte. Las letras se escriben en ASCII, y las posiciones como valor numérico.
   - Cerrar con un Byte con el valor numérico $(255)_{10}$
   - Rellenar la última línea con el valor numérico $(0)_{10}$ hasta completar los 12 byte de la línea.

   **R.**

   **4 pts.** La primera parte corresponde a los valores ASCII de cada caracter. Corresponde a: `49 31 2D 32 30 32 34 2D 31`. -1 pto por cada valor incorrecto.

   **1 pto.** La segunda parte, corresponde al valor numérico 23 escrito en hexadecimal que corresponde a `17`. Si el valor numérico es 24, entonces debería ser `18`.

   **4 pts.** Para codificar la palabra `FiuYPangUI`, se debe codificar la siguiente secuencia: 10, `a`, 5, `F`, 0, `g`, 7, `I`, 9, `i`, 1, `n`, 6, `P`, 4, `U`, 8, `u`, 2, `Y`, 3. La codificación, considerado que las letras van en ASCII, y el valor numérico se convierte a su valor hexadecimal es: `0A 61 05 46 00 67 07 49 09 69 01 6E 06 50 04 55 08 75 02 59 03`. 1 pto por incluir la longitud (primer byte), 2 pts si están las secuencias correctas. 1 pto por respetar el orden de mayúsculas y minúsculas.

   **1 pto.** La cantidad de byte usados en la parte anterior son 21. Por lo tanto $X = 21$, y el byte que corresponde a la tercera parte es `15`.

   **1 pto.** El byte que cierra es `FF`.

   **1 pto.** Si hay que rellenar, se rellena con el byte `00`.

   **4 pts.** Consolidando cada parte, el archivo queda (suponiendo sala B23) de la siguiente manera. Debe haber 12 byte por línea (-2pto si no cumple). **DEBE** verse el archivo consolidado con líneas de 12 bytes. No basta haber respondido cada parte anterior por separado, aunque cada una esté correcta. Si no está el archivo consolidad, son 0 pts en parte.

   ```
   49 31 2D 32 30 32 34 2D 31 17 15 0A
   61 05 46 00 67 07 49 09 69 01 6E 06
   50 04 55 08 75 02 59 03 FF 00 00 00
   ```