
LABORATORIOS 1 Y 2

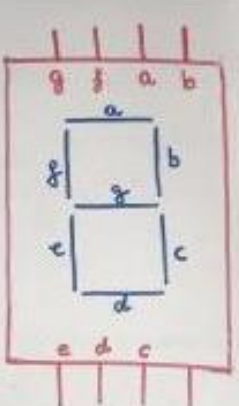
Fundamentos de computación
Ana Robledano Abasolo 1ºB

17 DE DICIEMBRE DE 2021
UFV

LABORATORIO 1

1. Conversor de 7 segmentos

Diagrama de un display de 7 segmentos:



Los segmentos están etiquetados como a, b, c, d, e, f, g. Los pines de conexión están etiquetados como g, f, a, b en la parte superior y e, d, c en la parte inferior.

TABLA DE VERDAD

$x_3 x_2 x_1 x_0$	a	b	c	d	e	f	g
0 0000	1	1	1	1	1	0	0
1 0001	0	1	1	0	0	0	0
2 0010	1	1	0	1	1	0	1
3 0011	1	1	1	1	0	0	1
4 0100	0	1	1	0	0	1	1
5 0101	1	0	1	1	0	1	1
6 0110	1	0	1	1	1	1	1
7 0111	1	1	1	0	0	0	0
8 1000	1	1	1	1	1	1	1
9 1001	1	1	1	0	0	1	1

Análisis Combinacional

Archivo Editar Proyecto Simular Ventana Ayuda

Entradas Salidas Tabla Expresión Minimizado

Salida: a

$\overline{x_3} \overline{x_2} \overline{x_0} + \overline{x_3} x_1 + \overline{x_3} x_2 x_0 + x_3 \overline{x_2} x_1$

$\sim x_3 \sim x_2 \sim x_0 + \sim x_3 x_1 + \sim x_3 x_2 x_0 + x_3 \sim x_2 \sim x_1$

Limpiar Recargar Intro

Crear Circuito

Salida:

$$\overline{x_3} \overline{x_2} + \overline{x_3} \overline{x_1} \overline{x_0} + \overline{x_2} \overline{x_1} + \overline{x_3} x_1 x_0$$

$\sim x_3 \sim x_2 + \sim x_3 \sim x_1 \sim x_0 + \sim x_2 \sim x_1 + \sim x_3 x_1 x_0$

Limpiar

Recargar

Intro

Salida:

$$\overline{x_2} \overline{x_1} + \overline{x_3} x_0 + \overline{x_3} x_2$$

$\sim x_2 \sim x_1 + \sim x_3 x_0 + \sim x_3 x_2$

Limpiar

Recargar

Intro

Salida:

$$\overline{x_3} \overline{x_2} \overline{x_0} + \overline{x_3} \overline{x_2} x_1 + \overline{x_3} x_1 \overline{x_0} + \overline{x_3} x_2 \overline{x_1} x_0 + x_3 \overline{x_2} \overline{x_1}$$

$\sim x_3 \sim x_2 \sim x_0 + \sim x_3 \sim x_2 x_1 + \sim x_3 x_1 \sim x_0 + \sim x_3 x_2 \sim x_1$
 $x_1 x_0 + x_3 \sim x_2 \sim x_1$

Limpiar

Recargar

Intro

Salida:

$$\overline{x_2 x_1 x_0} + \overline{x_3 x_1 x_0}$$

$\sim x_2 \sim x_1 \sim x_0 + \sim x_3 x_1 \sim x_0$

Limpiar

Recargar

Intro

Salida:

$$\overline{x_3 x_1 x_0} + \overline{x_3 x_2 x_1} + \overline{x_3 x_2 x_0} + \overline{x_3 x_2 x_1}$$

$\sim x_3 \sim x_1 \sim x_0 + \sim x_3 x_2 \sim x_1 + \sim x_3 x_2 \sim x_0 + x_3 \sim x_2$
 $\sim x_1$

Limpiar

Recargar

Intro

Salida:

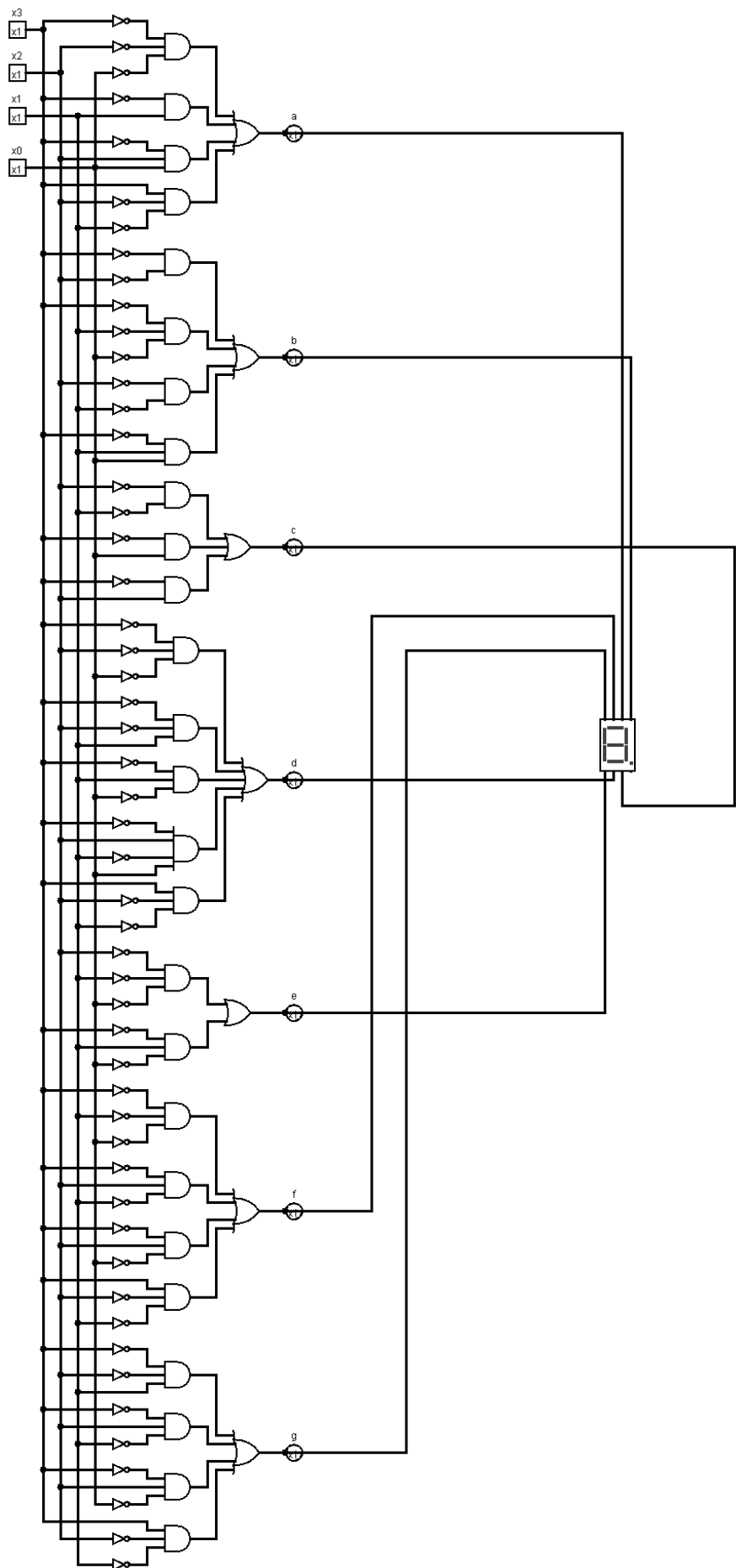
$$\overline{x_3 x_2 x_1} + \overline{x_3 x_2 x_1} + \overline{x_3 x_2 x_0} + \overline{x_3 x_2 x_1}$$

$\sim x_3 \sim x_2 x_1 + \sim x_3 x_2 \sim x_1 + \sim x_3 x_2 \sim x_0 + x_3 \sim x_2$
 $\sim x_1$

Limpiar

Recargar

Intro



2. Conjunto universal de operadores

2.1.

Leyes de Boole

OR (+)	AND (·)
$A + 0 = A$	$A \cdot 0 = 0$
$A + 1 = 1$	$A \cdot 1 = A$
$A + A = A$	$A \cdot A = A$
$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$

X	Y	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

$\bar{\bar{X}} = X$
 $\overline{X + Y} = \bar{X} \cdot \bar{Y}$
 $\overline{X \cdot Y} = \bar{X} + \bar{Y}$

NOT: $\bar{X} = \{x = x + x\} \Rightarrow \overline{x + x}$

AND: $X \cdot Y = (x + x) \cdot (y + y) = (\overline{x + x}) \cdot (\overline{y + y}) = \overline{(x + x) + (y + y)}$

OR: $X + Y = \overline{\overline{x + y}} = \overline{(\overline{x + y}) \cdot (\overline{x + y})} = \overline{(\overline{x + y}) + (\overline{x + y})}$

NOT: $x \rightarrow \bar{x}$ \rightarrow $x \rightarrow \overline{x + x}$

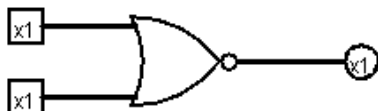
AND: $x, y \rightarrow x \cdot y$ \rightarrow $x, y \rightarrow \overline{(x + x) + (y + y)}$

OR: $x, y \rightarrow x + y$ \rightarrow $x, y \rightarrow \overline{(\overline{x + y}) + (\overline{x + y})}$

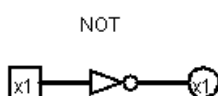
2.2.

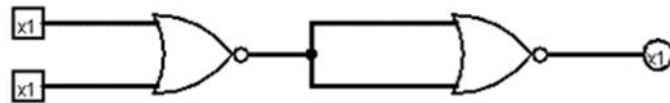
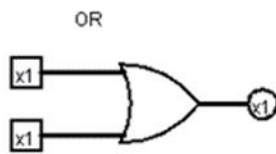
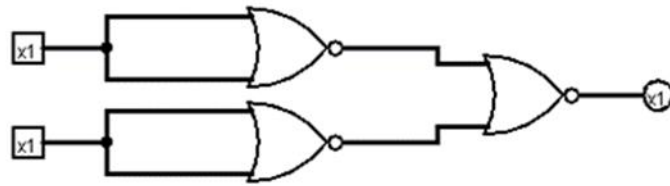
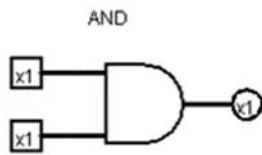
Puerta nor:

NOR



Operadores lógicos not, and, or, expresados únicamente con puertas nor:





LABORATORIO 2:

3.1 Para poder ser ejecutado, el programa requiere ser almacenado en memoria (lo que realiza habitualmente el sistema operativo). ¿Cuántos bytes de memoria requiere este programa?

4bytes/instrucción

8bytes/dato, cada dato se guarda en una posición de memoria de 8bytes. Y hay 4 variables de datos.

1byte/resultado.

Por lo tanto, se necesitan 73 bytes.

$$4 * n^{\circ} \text{instrucciones}(10) + 8 * n^{\circ} \text{datos}(4) + 1 * \text{resultados}(1) = 73$$

3.2 Carga el programa en QTSPim y observa la memoria de datos. ¿Cómo interpretas lo que aparece ahí: a24bf618?

Son los datos declarados en nuestro programa, los datos iniciales, que aparecen en hexadecimal, junto a la dirección de memoria [10000000] .

De derecha a izquierda, se distribuyen de la siguiente manera:

- 18: byte de dirección 10000000
- f6: byte de dirección 10000001

- 4b: byte de dirección 10000002
- a2: byte de dirección 10000003

	a 2	4 b	f 6	1 8
Código	Registro	Registro	Número a sumar al registro	
101000	10010	01011	1111 0110 0001 1000	

3.3 ¿En qué dirección se guardará el resultado?

En la dirección de memoria [00400044]. Escribe el resultado en el R15 y añade 5 al R8.

3.4 ¿Qué hace este programa? Ejecútalo y comprueba el resultado.

Cada registro obtiene los siguientes valores:

- **R9**, la var1 24, en hexadecimal (18).
- **R10**, la var2 -10, en hexadecimal (ffffff6).
- **R11**, la var3 75, en hexadecimal (4b).
- **R12**, la var4 -94, en hexadecimal (ffffffa2).
- **R13**, la suma de los registros R9 y R10 (e).

$$24 + (-10) = 14$$

(las operaciones las hace el procesador en binario complemento a dos, y se muestran en hexadecimal en pantalla)

- **R14**, la suma de los registros R11 y R12 (ffffffed).
 $75 + (-94) = -19$
- **R15**, la resta del registro R13 menos el R14 (21).
 $14 - (-19) = 33$

(este resultado se guarda en la variable 5 y se suma 5 al registro 8, 10000000) RAM[5+10000000]

- **R31**, el resultado final.

En conclusión, las operaciones que realiza el programa son:

$$(\text{var1} + \text{var2}) - (\text{var3} + \text{var4}) = \text{var5}$$

$$(24 + (-10)) - (75 + (-94)) = 33$$

El programa lee las variables 1,2,3,4 y las almacena en memoria (cada una en un registro), después de operar con ellas y almacenar los resultados, escribe el resultado final, guardado en la variable5.

3.5 Durante la ejecución, el programa debe acceder numerosas veces a memoria, pues cada instrucción debe ser leída previamente a su ejecución, los datos iniciales también deben ser leídos de memoria y el resultado final debe ser escrito en la memoria. ¿Cuántos accesos en total se producen durante la ejecución de este programa en concreto? Escribe la secuencia de direcciones accedidas.

Durante la ejecución se producen 18 accesos a memoria.

[00400004] [00400008] [0040000c] [00400010] [00400014] [00400024]
[00400028] [0040002c] [00400030] [00400034] [00400038] [0040003c]
[00400040] [00400044] [00400048] [00400018] [0040001c] [00400020].

Hay 10 instrucciones, el programa tiene que ir a memoria 10 veces para leerlas y después ejecutarlas.

Pero también hay algunas instrucciones en las que se tiene que ir a memoria para leer o escribir el dato.

3.6. Como verás en la anterior secuencia, muchas veces se accede a posiciones consecutivas en memoria. ¿Se te ocurre cómo podría el procesador aprovechar esta característica para reducir el número de accesos a memoria y hacer así más eficiente la ejecución?

Como en el programa se repite la operación suma dos veces, creo que se podría utilizar un bucle de repetición, para poder ir a memoria una sola vez (para leer la instrucción add) y hacer dos sumas.

3.7 ¿Cuál es el código binario que le corresponde a la instrucción add? ¿Y a la instrucción sb?

Para hallar el código binario de la instrucción add, he ido a la dirección de memoria (en text):

[00400038] 012a6820 add \$13, \$9, \$10

En binario

000000 0100 1010 100110100000100000

Los primeros 6 dígitos son el código en binario de la instrucción add.

Para la hallar el de la instrucción sb:

[00400044] a10f0005 sb \$15, 5(\$8)

101000 0100 0011 11000000000101

Los primeros 6 dígitos son el código en binario de la instrucción sb.

En conclusión:

- Add: 000000
- Sb: 101000

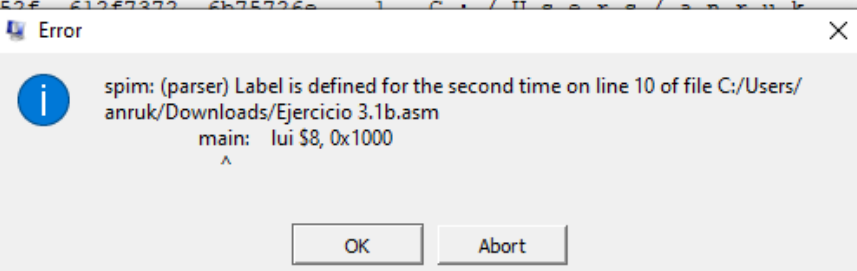
3.8 Considera ahora este programa, que hace lo mismo que el anterior, pero sobre datos de tipo Word en vez de byte. Observa que el primer dato ha sido declarado como número real y no entero, es un error introducido a propósito para ver cómo funciona el programa. ¿Qué crees que va a pasar? Ejecuta el programa y comprueba qué pasa exactamente. Explícalo. ¿Coincide con lo que habías pensado?

Creo que se producirá un error, ya que las operaciones son de tipo entero, pero el dato es float.

Salta un error y no se ejecuta el programa.

Esto se debe al overflow que se produce al almacenar el dato tipo float.

```
ab 7fffffa8d 7fffffa64 7fffffa46 . . . . . d . . . F . . .
db 7fffff9c4 7fffff9b0 7fffff9a1 . . . . .
8b 7fffff964 7fffff93e 7fffff92f . . . . d . . . > . . . / . . .
11 7fffff902 7fffff8e7 7fffff8d5 . . . . .
00 31280000 73612e29 2e33006d . . . . . ( 1 ) . a s m . 3 .
31 65735525 612f7372 6b75726e 1 C : / U s e r s / a n r u k
2f 616f6
69 6e697
49 55005
4c 73555
6b 414e5
00 4d4f4
49 49464
53 49414
50 65735
5c 61746
65 504d4554 5c3a433d 72657355 e m p . T E M P = C : \ U s e r
73 5c6b7572 44707041 5c617461 s \ a n r u k \ A p p D a t a \
4c 65545c6c 5300706d 65747379 L o c a l \ T e m p . S y s t e
6d 3a433d74 4e49575c 53574f44 m R o o t = C : \ W I N D O W S
```

An error dialog box titled "Error" with a close button (X) in the top right corner. It contains an information icon (i) and the following text: "spim: (parser) Label is defined for the second time on line 10 of file C:/Users/anruk/Downloads/Ejercicio 3.1b.asm". Below this, it shows the assembly instruction "main: lui \$8, 0x1000" with a caret (^) under the '8' in the register field. At the bottom, there are "OK" and "Abort" buttons.

Error

spim: (parser) Label is defined for the second time on line 10 of file C:/Users/anruk/Downloads/Ejercicio 3.1b.asm

main: lui \$8, 0x1000
^

OK Abort