Clase 11 - Repaso I1

IIC2343 - Arquitectura de Computadores

Profesor:
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl

Resumen de las clases pasadas

¿Qué es un computador?

Introducción:

- Un computador lo definimos como una **máquina programable que ejecuta programas.**
- Para programar necesitamos:
    - Datos: números (enteros, reales) , texto, imágenes, etc
    - Operaciones: suma, resta, multiplicación, división, etc
    - Variables: simples, arreglos
    - Control de flujo: comparaciones, manejo de ciclos-

Representación úmerica de enteros

Sistema posicional de base genérica

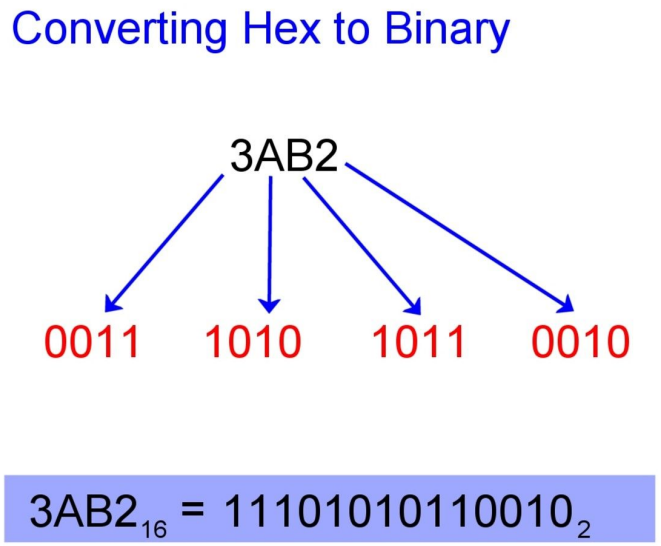
- *s*: símbolo
- *k*: = Posición del símbolo en la secuencia, siendo 0 la posición del extremo derecho.
- *b*: **base**
- *n*: cantidad de símbolos en la secuencia
- Notación típica: (—)b

$$\sum_{k=0}^{n-1} s_k \times b^k$$

Fuente imagen: https://theplctutor.com/decimal.html

Método de conversión: binaria hacia hexa

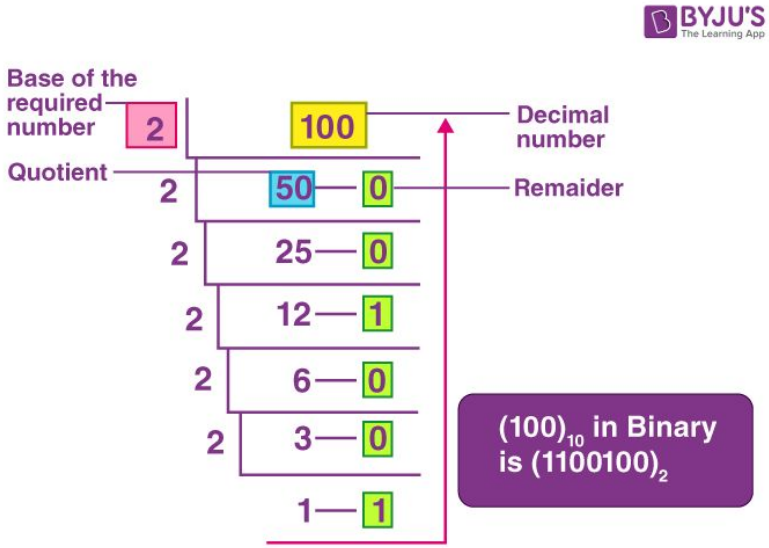
- Ocuparemos un método aprovechando concatenación
- Agrupamos los términos numéricos para obtener el resultado



Fuente imagen: https://owlcation.com/stem/How-to-Convert-Hex-to-Binary-and-Binary-to-Hexadecimal

Método de conversión: decimal hacia binario

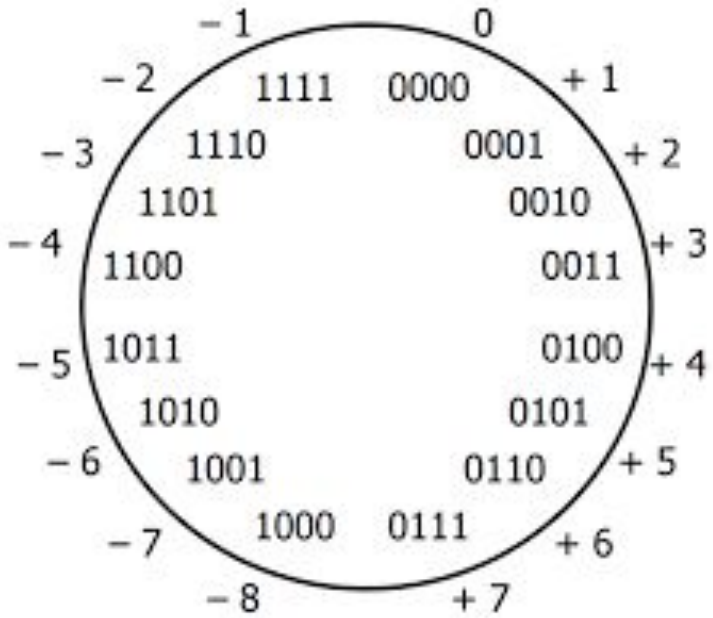
- Se obtiene el resto entre el número en base decimal y el divisor 2.
- Se obtiene el resto entre el número en base decimal y el divisor 2.
- Para obtener el siguiente símbolo de la secuencia, realizar la misma operación con el resultado de la división entera del número



Fuente imagen: https://owlcation.com/stem/How-to-Convert-Hex-to-Binary-and-Binary-to-Hexadecimal

Complemento 2 (C2)

- Sumar una unidad al complemento al C1
- Ahora el cero es intuitivo
- **Contra:** Tenemos una representación desbalanceada
- ***Overflow:*** **Si una operación aritmética resulta en un valor no representable, nos dará un valor erróneo**



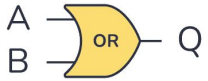
Fuente imagen: https://courses.cs.washington.edu/courses/cse410/17wi/sections/02/IntegersAndBitwiseSol.pdf

string	unsigned	sign & magnitude	1's complement	2's complement
0000	0	0	0	**0**
0001	1	1	1	**1**
0010	2	2	2	**2**
0011	3	3	3	**3**
0100	4	4	4	**4**
0101	5	5	5	**5**
0110	6	6	6	**6**
0111	7	7	7	**7**
1000	8	–0	–7	**–8**
1001	9	–1	–6	**–7**
1010	10	–2	–5	**–6**
1011	11	–3	–4	**–5**
1100	12	–4	–3	**–4**
1101	13	–5	–2	**–3**
1110	14	–6	–1	**–2**
1111	15	–7	–0	**–1**

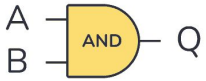
Circuitos Digitales

Compuertas lógicas

A	Q
0	1
1	0



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

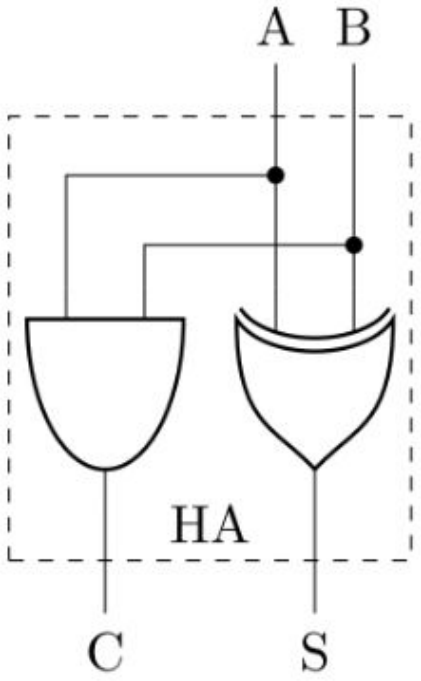
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Algebra booleana

Equivalence	Name of Identity
$p \wedge T \equiv p$ $p \vee F \equiv p$	Identity Laws
$p \wedge F \equiv F$ $p \vee T \equiv T$	Domination Laws
$p \wedge p \equiv p$ $p \vee p \equiv p$	Idempotent Laws
$\neg(\neg p) \equiv p$	Double Negation Law
$p \wedge q \equiv q \wedge p$ $p \vee q \equiv q \vee p$	Commutative Laws
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ $(p \vee q) \vee r \equiv p \vee (q \vee r)$	Associative Laws
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	Ditributive Laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's Laws
$p \wedge (p \vee q) \equiv p$ $p \vee (p \wedge q) \equiv p$	Absorption Laws
$p \wedge \neg p \equiv F$ $p \vee \neg p \equiv T$	Negation Laws

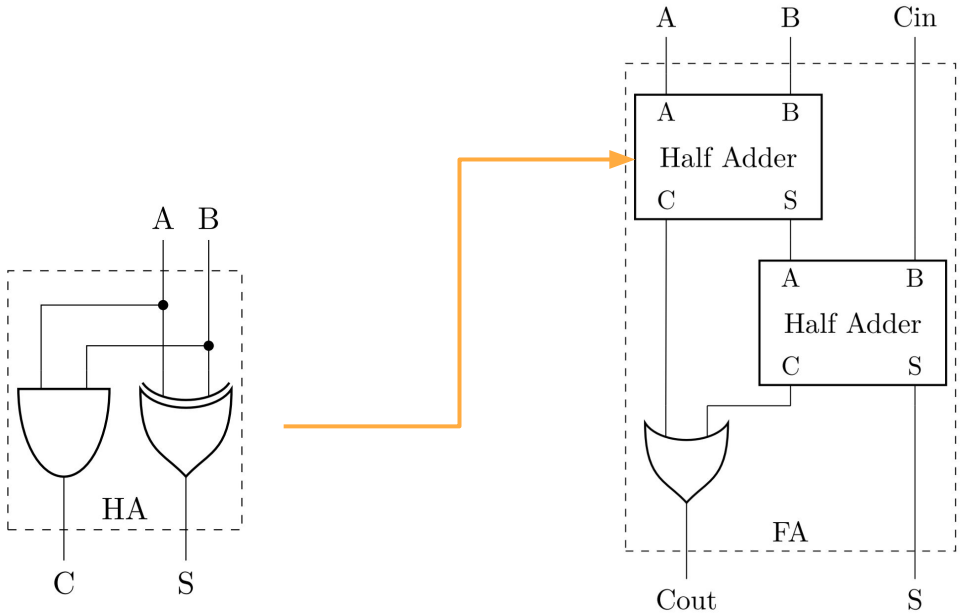
Half Adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



(a) Half Adder

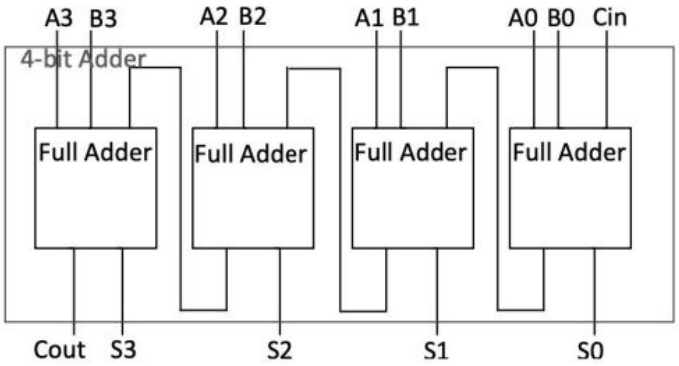
Full Adder

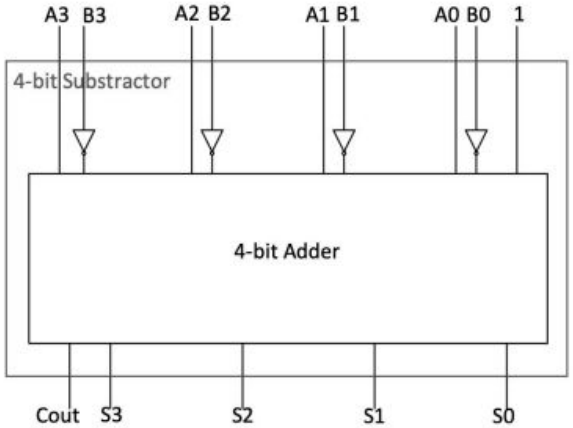


(a) Half Adder

(b) Full Adder

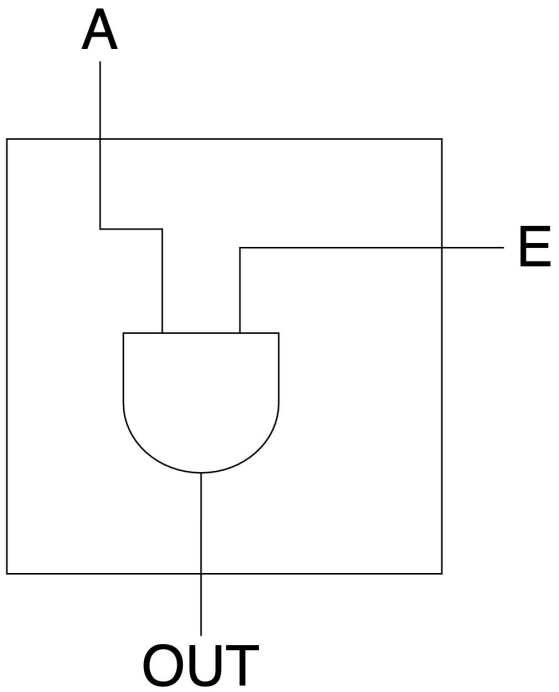
Sumador/Restador





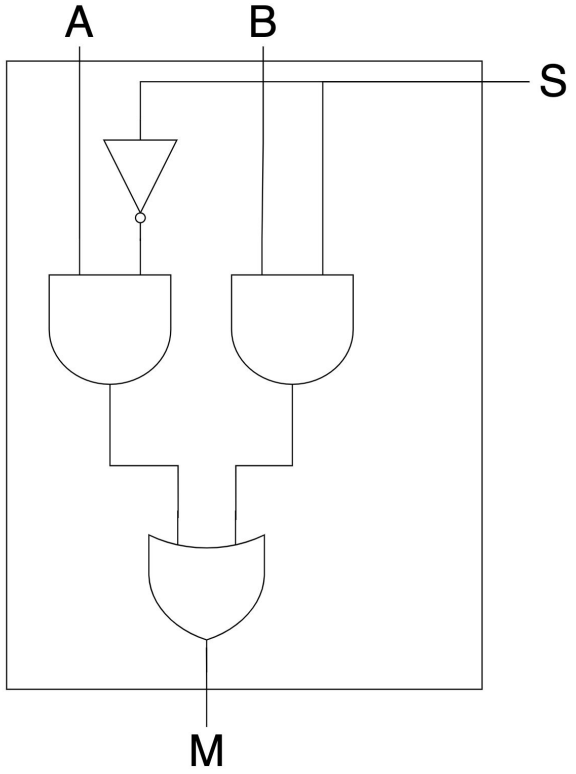
Circuitos de control - Enabler

OUT	E
0	0
A	1



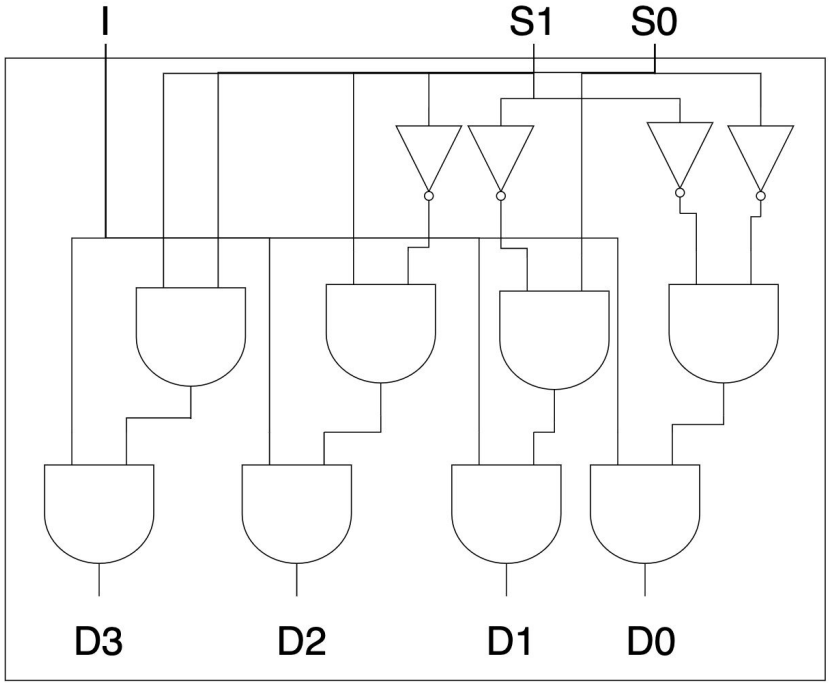
Circuitos de control - Multiplexor

S	M
0	A
1	B



Circuitos de control - Demux

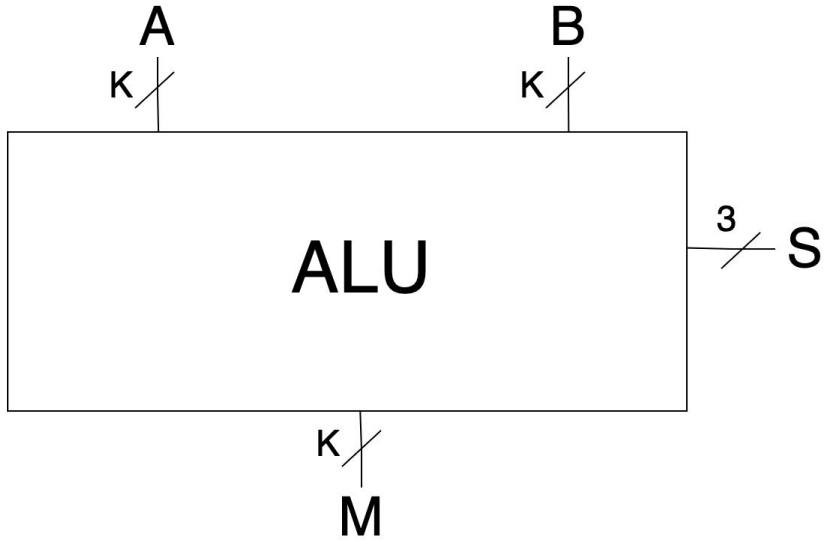
S1	S0	Output
0	0	D0 = I
0	1	D1 = I
1	0	D2 = I
1	1	D3 = I



Unidad Aritmética Lógica: ALU

- **Tabla de valores**:

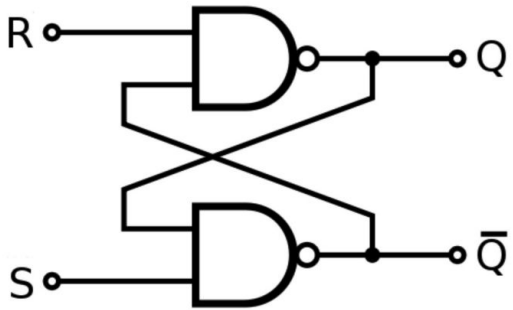
S2	S1	S0	M
0	0	0	Suma
0	0	1	Resta
0	1	0	And
0	1	1	Or
1	0	0	Not
1	0	1	Xor
1	1	0	Shift left
1	1	1	Shift right



Almacenamiento de Datos

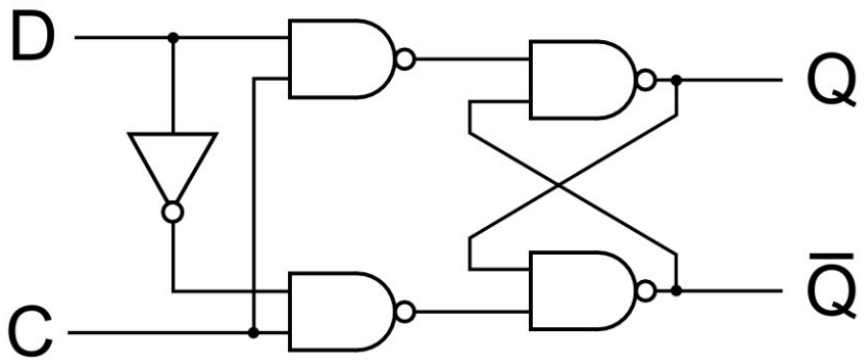
Almacenamiento de datos: Latch RS

S	R	Q(t+1)
0	0	-
0	1	0
1	0	1
1	1	Q(t)



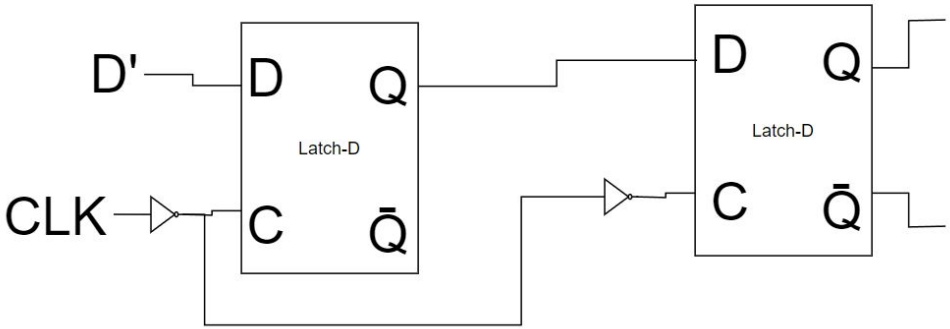
Almacenamiento de datos: Latch D

C	D	Q(t+1)
0	0	Q(t)
0	1	Q(t)
1	0	0
1	1	1



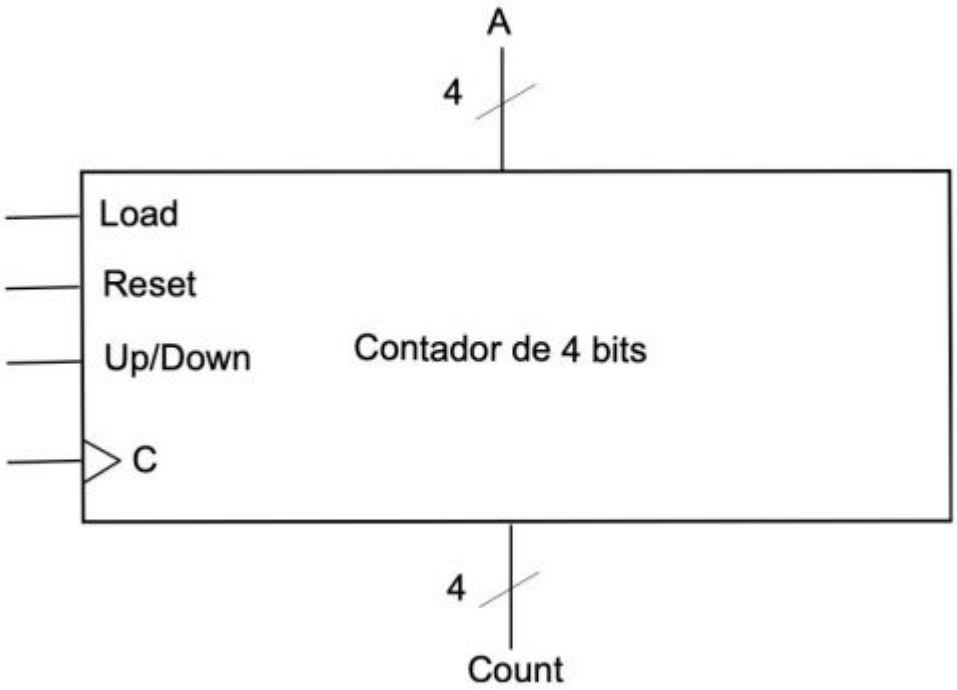
Almacenamiento de datos: Flip-Flop D

C	D	Q(t+1)
↑	0	0
↑	1	1
*	*	Q(t)

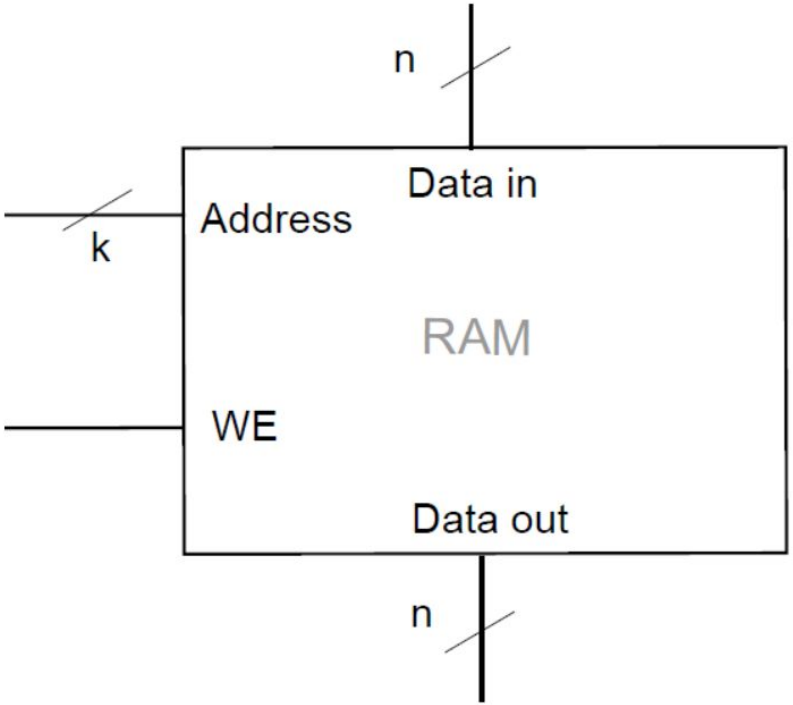


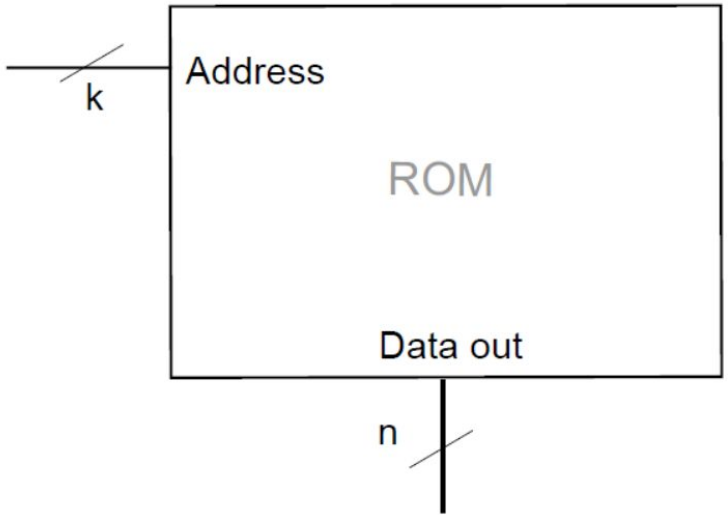
Almacenamiento de datos: Registro Contador

- Con lo anterior se puede generalizar para tener un registro contador
- Una señal Up indicando que el valor almacenado se suma uno
- Una señal Down indicando que el valor almacenado se resta uno



Almacenamiento de datos: Memorias





Programabilidad

Programabilidad: Calculadora Avanzada

- Ahora tenemos control de lo que cuando almacenar el resultado en los registros
- Notamos que también **tenemos control en la operación** ejecutada en la ALU

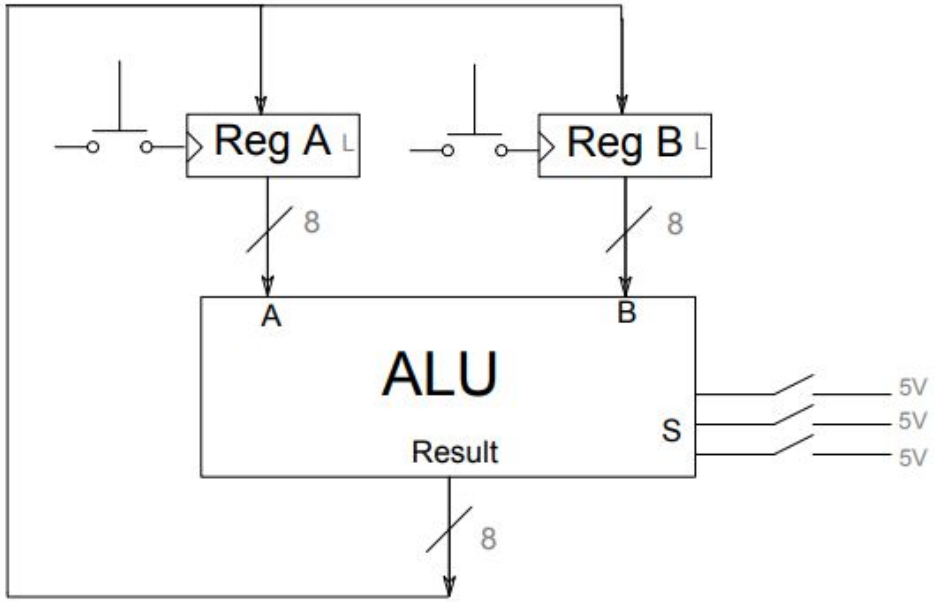


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/4%20-%20Programabilidad.pdf

Computador básico: Extensión - Literales

- Se debe agregar a la máquina programable es la capacidad de operar con **literales**
- Un **literal** se refiere a un valor numérico que se define explícitamente
- Se extiende la capacidad de la memoria de instrucciones

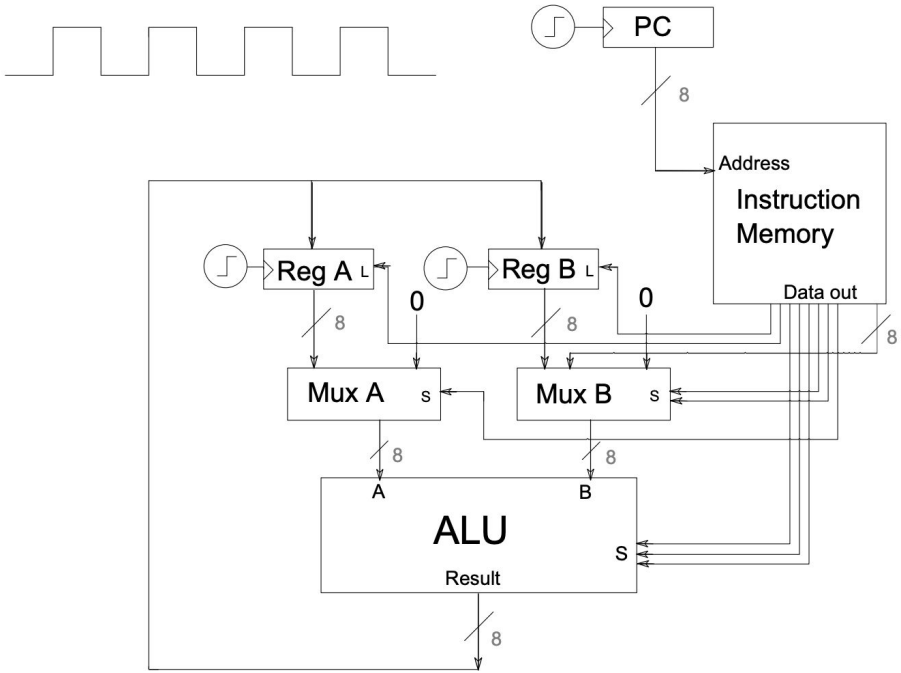


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/4%20-%20Programabilidad.pdf

Computador básico: Opcodes

- Usamos *opcodes* (códigos de operación) que definen la combinación de señales de control que ejecuta una instrucción.
- Ahora, falta un **componente que decodifique** *opcodes*.

Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
000000	1	0	1	0	0	0	0	0	A=B
000001	0	1	0	1	1	0	0	0	B=A
000010	1	0	1	0	1	0	0	0	A=Lit
000011	0	1	1	0	1	0	0	0	B=Lit
000100	1	0	0	0	0	0	0	0	A=A+B
000101	0	1	0	0	0	0	0	0	B=A+B
000110	1	0	0	0	1	0	0	0	A=A+Lit
000111	1	0	0	0	0	0	0	1	A=A-B
001000	0	1	0	0	0	0	0	1	B=A-B
001001	1	0	0	0	1	0	0	1	A=A-Lit
001010	1	0	0	0	0	0	1	0	A=A and B
001011	0	1	0	0	0	0	1	0	B=A and B
001100	1	0	0	0	1	0	1	0	A=A and Lit
001101	1	0	0	0	0	0	1	1	A=A or B
001110	0	1	0	0	0	0	1	1	B=A or B
001111	1	0	0	0	1	0	1	1	A=A or Lit
010000	1	0	0	0	0	1	0	0	A=not A
010001	0	1	0	0	0	1	0	0	B=not A
010010	1	0	0	0	0	1	0	1	A=A xor B
010011	0	1	0	0	0	1	0	1	B=A xor B
010100	1	0	0	0	1	1	0	1	A=A xor Lit
010101	1	0	0	0	0	1	1	0	A=shift left A
010110	0	1	0	0	0	1	1	0	B=shift left A
010111	1	0	0	0	0	1	1	1	A=shift right A
011000	0	1	0	0	0	1	1	1	B=shift right A

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/4%20-%20Programabilidad.pdf

Computador básico: Direccionamiento

- **Direccionamiento Directo:** Se indica la dirección de memoria con un literal
- **Direccionamiento Indirecto:** Se indica la dirección de memoria con el valor de un registro

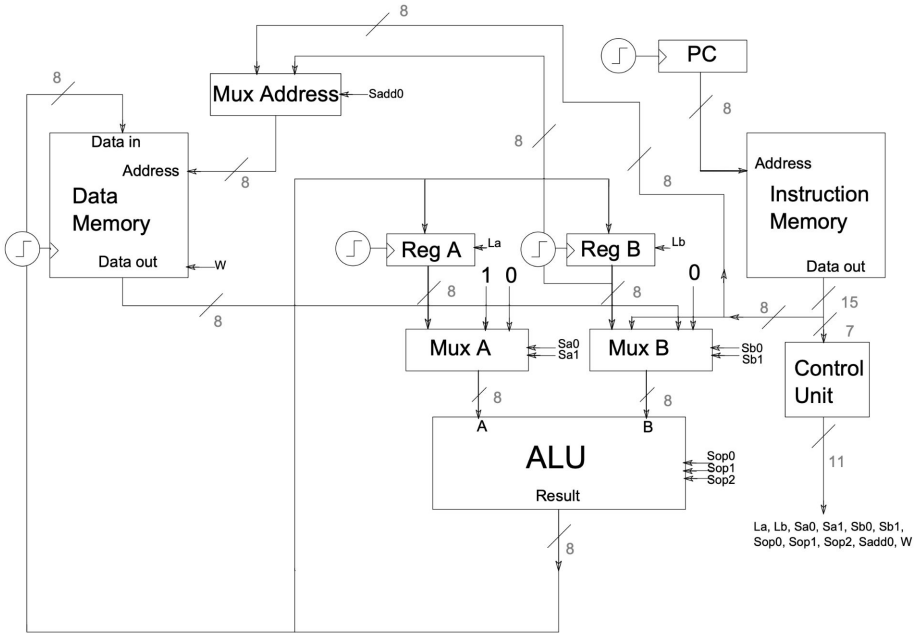


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/4%20-%20Programabilidad.pdf

Computador básico: Variables en Assembly

- Existirá un segmento **DATA** donde manejaremos las variables
- Siempre luego del segmento de DATA se tendrá el segmento de **CODE** donde existirá las instrucciones

Dirección	Label	Instrucción/Dato
	DATA:	
0x00	var0	Dato 0
0x01	var1	Dato 1
0x02	var2	Dato 2
0x03		Dato 3
0x04		Dato 4
	CODE:	
0x00		Instrucción 0
0x01		Instrucción 1
0x02		Instrucción 2
0x03		Instrucción 3
0x04		Instrucción 4

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/4%20-%20Programabilidad.pdf

Saltos

Salto Incondicional: Instrucciones (ejemplo)

- La limitante es el hecho de **no iterar**
- Las instrucciones que nos permiten cambiar la dirección de código las llamaremos **saltos**
- En particular en el ejemplo tenemos un **salto incondicional**

Dirección	Instrucción	Operandos
0x00	MOV	A,0
0x01	MOV	B,1
0x02	ADD	A,B
0x03	ADD	B,A
0x04	JMP	0x02

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto Incondicional: Label

- Es un **indicador** que se puede agregar en una línea del código assembly para referirse a la **dirección de memoria** asociada a esa línea

Dirección	Label	Instrucción	Operandos
0x00		MOV	A,0
0x01		MOV	B,1
0x02	start:	ADD	A,B
0x03		ADD	B,A
0x04		JMP	start

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto Incondicional: Implementación

- La respuesta es modificar el resgistro del **Program Counter**
- Esta conexión va desde el literal que viene de la **Instruction Memory**
- Agregamos una nueva señal de control **Lpc** (Load PC)

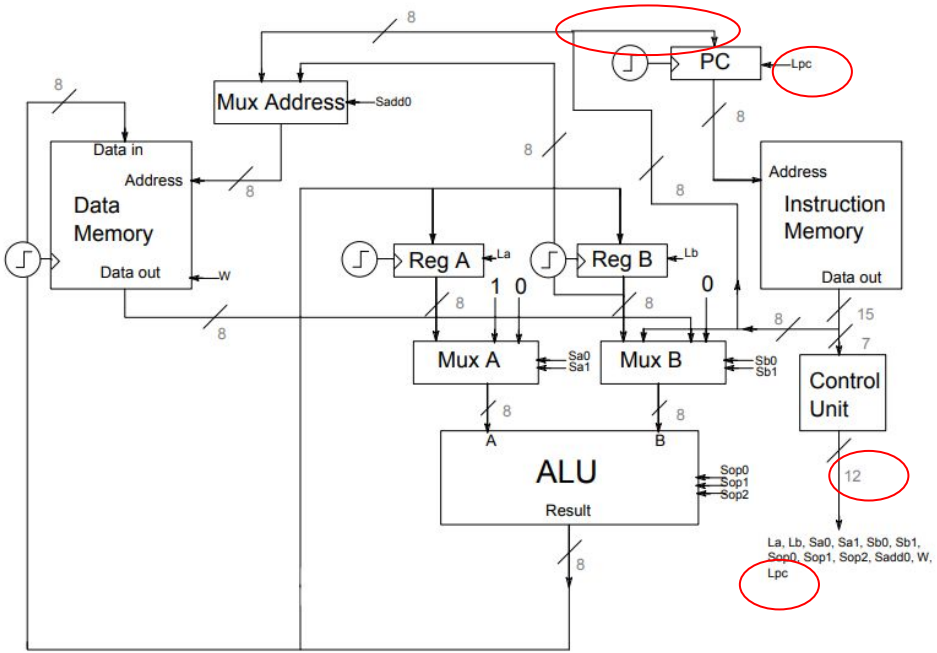


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto condicional: Implementación

- Dado que el salto se realiza con las condiciones ocurridas en la operación anterior
- Se necesita un nuevo registro **Status**
- Su salida se conecta la Unidad de Control

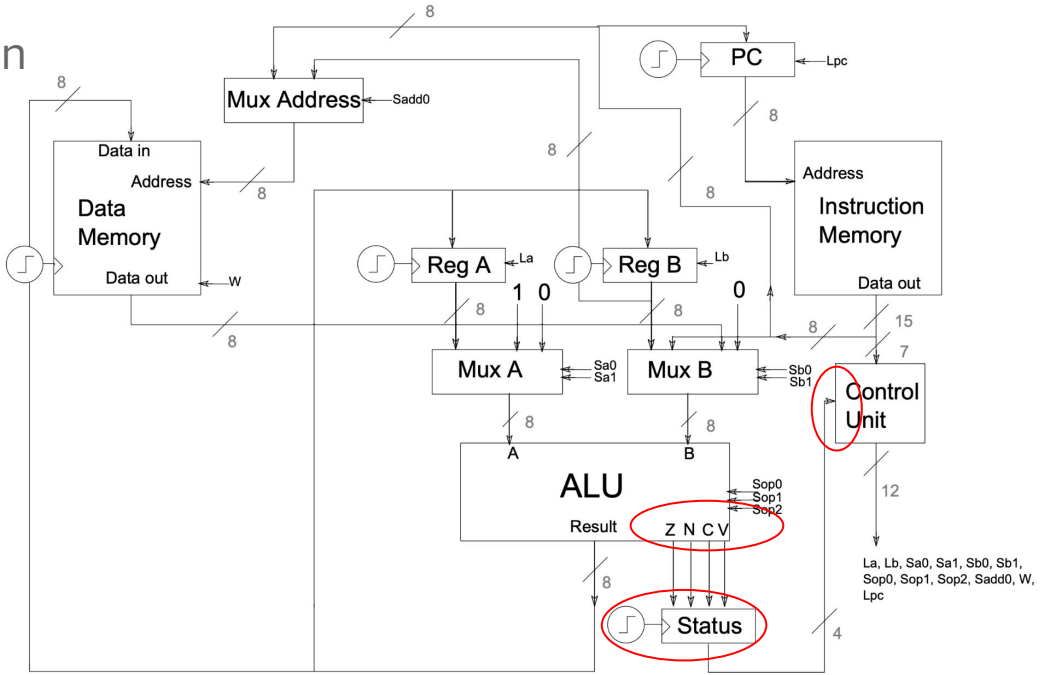


Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Salto Resumen

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CMP	A,B	A-B		
	A,Lit	A-Lit		CMP A,0
JMP	Dir	PC = Dir		JMP end
JEQ	Dir	PC = Dir	Z=1	JEQ label
JNE	Dir	PC = Dir	Z=0	JNE label
JGT	Dir	PC = Dir	N=0 y Z=0	JGT label
JLT	Dir	PC = Dir	N=1	JLT label
JGE	Dir	PC = Dir	N=0	JGE label
JLE	Dir	PC = Dir	Z=1 o N=1	JLE label
JCR	Dir	PC = Dir	C=1	JCR label
JOV	Dir	PC = Dir	V=1	JOV label

Imagen Referencia: https://github.com/IIC2343/Syllabus-anteriores/blob/main/Apuntes%20(viejos)/5%20-%20Saltos%20y%20Subrutinas.pdf

Clase 11 - Repaso I1

IIC2343 - Arquitectura de Computadores

Profesor:
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl