

Clase 05 - Programabilidad

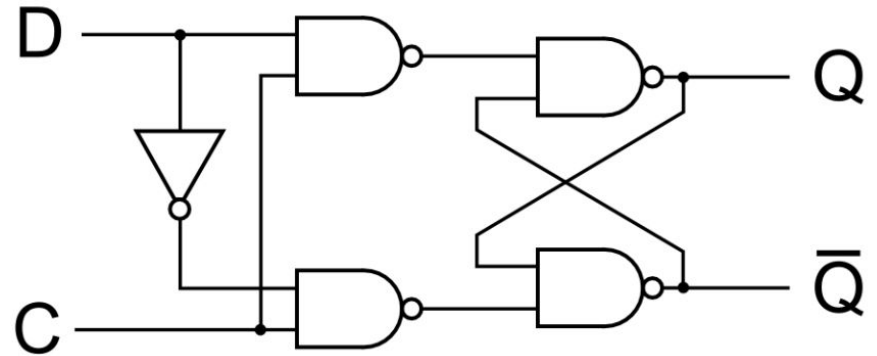


Profesor: **IIC2343 - Arquitectura de Computadores**
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl

Resumen de la clase pasada

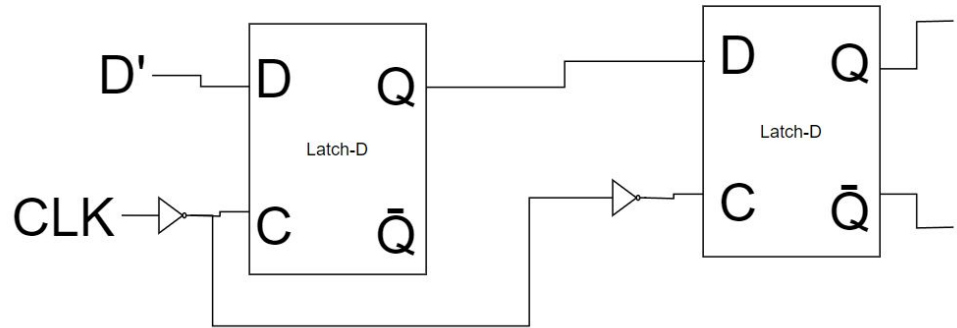
Almacenamiento de datos: Latch D

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



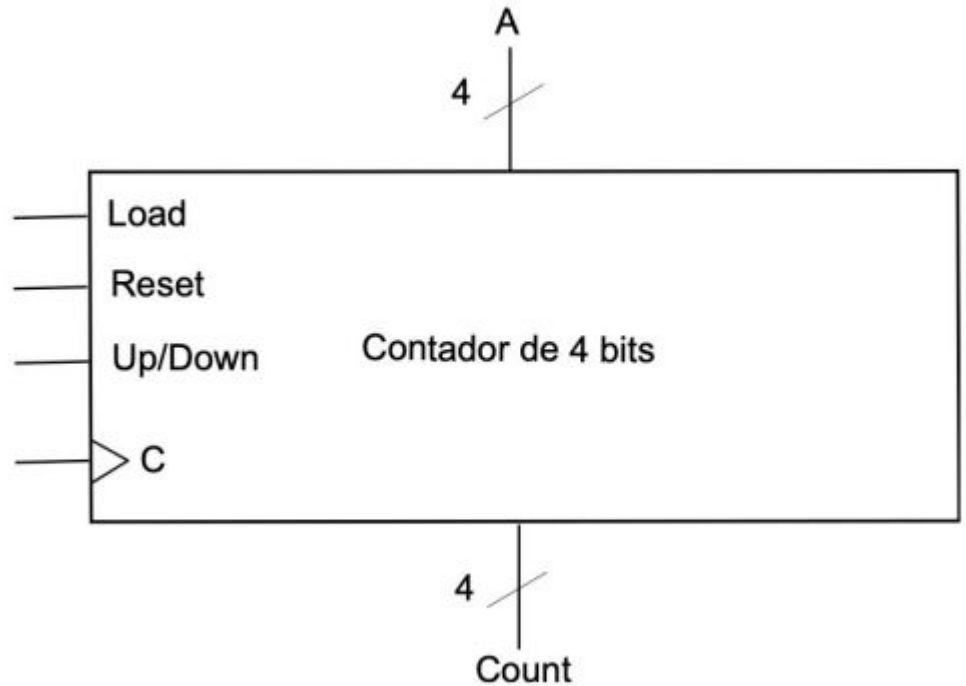
Almacenamiento de datos: Flip-Flop D

C	D	Q(t+1)
↑	0	0
↑	1	1
*	*	Q(t)

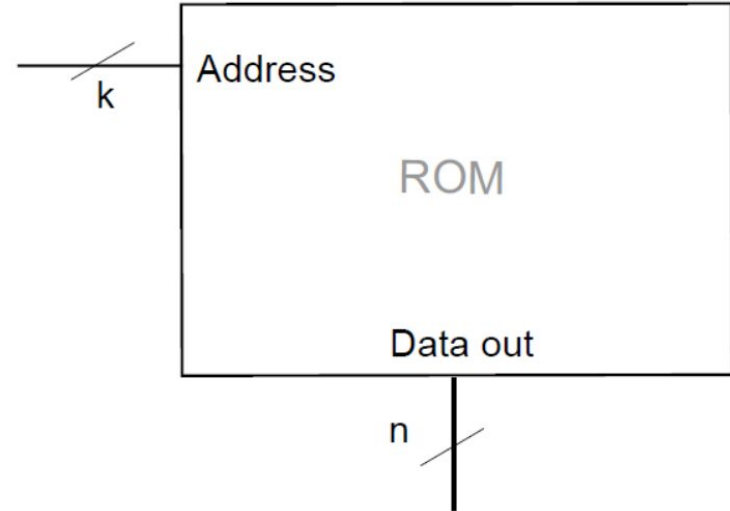
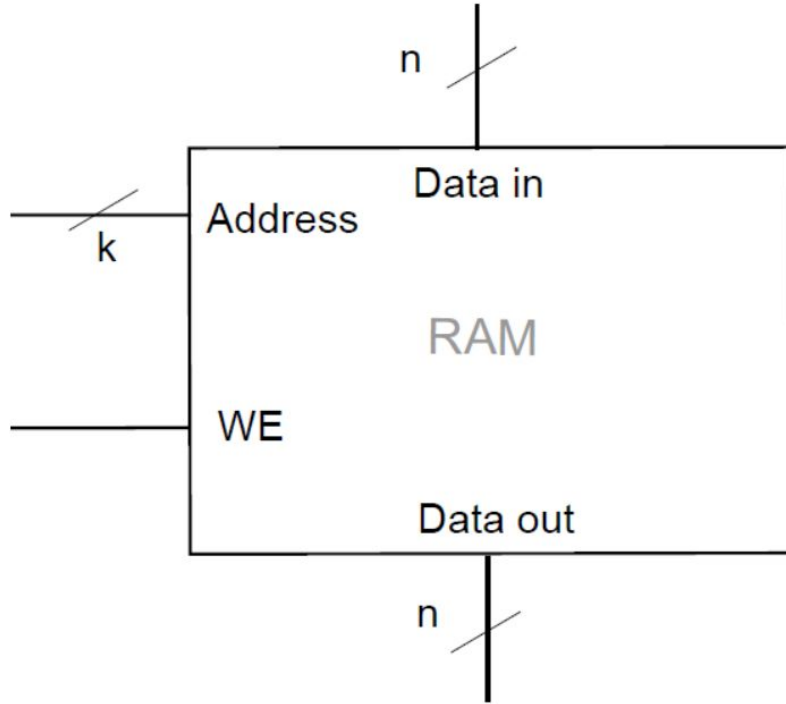


Almacenamiento de datos: Registro Contador

- 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



¿Dudas?

Introducción del curso:

- 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 
- Para poder tener esta última parte requerimos de comenzar el capítulo de **¡programabilidad!**

Programabilidad: Calculadora Simple

select	s2	s1	s0	operación
0	0	0	0	Suma
1	0	0	1	Resta
2	0	1	0	And
3	0	1	1	Or
4	1	0	0	Not A
5	1	0	1	Xor
6	1	1	0	Shift Left A
7	1	1	1	Shift Right A

Tabla 1: Operaciones de la ALU.

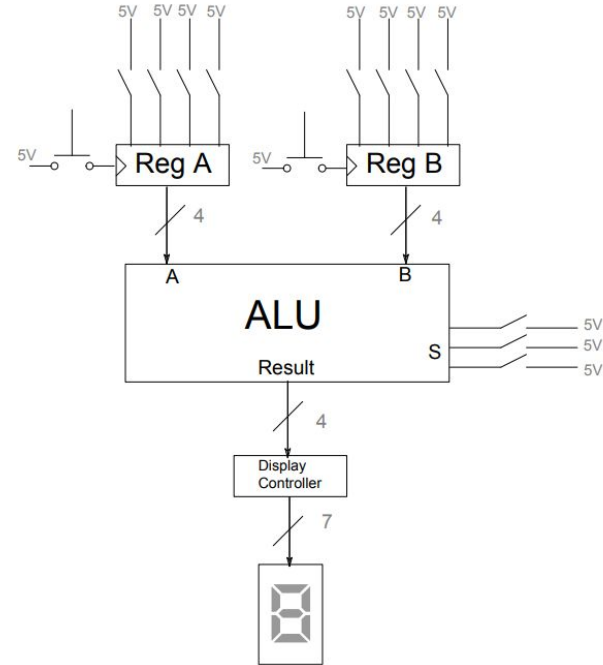


Figura 1: Calculadora de 4 bits.

Programabilidad: Calculadora Simple

- Con lo visto hasta ahora podemos construir una calculadora
- Esto es una versión simplificada de lo hecho en laboratorio
- **¿Qué limitaciones tiene?**

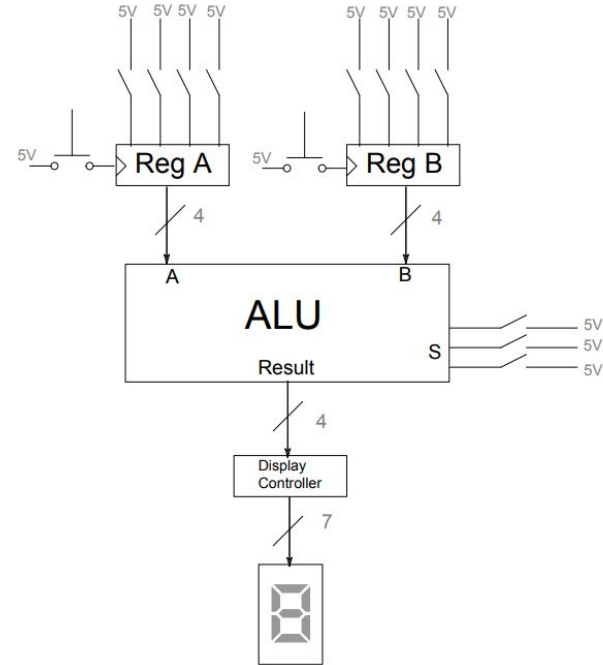
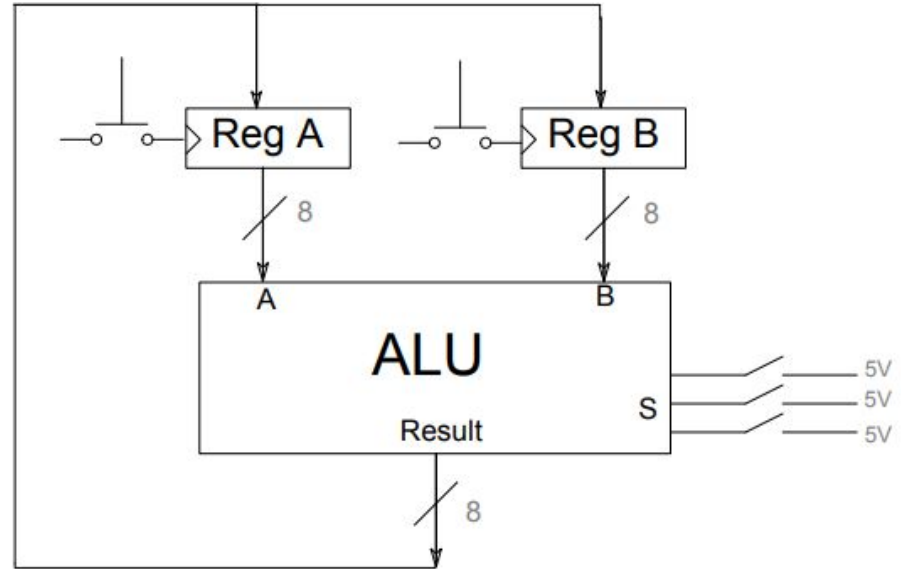


Figura 1: Calculadora de 4 bits.

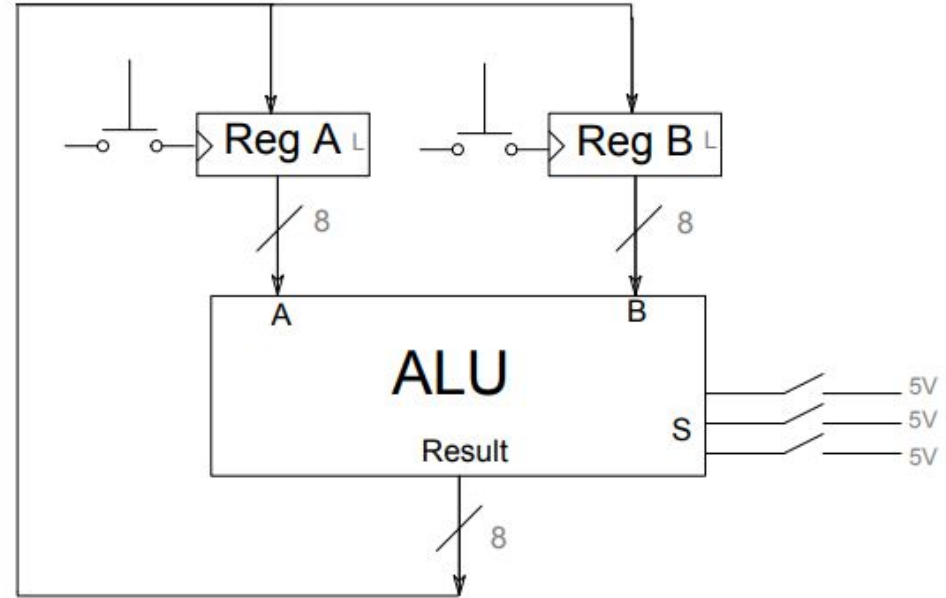
Programabilidad: Calculadora Avanzada

- La respuesta a la pregunta anterior es el que **no podemos usar el resultado anterior**
- Expandimos eso usando el valor de resultado como valor
- Notar que **no tenemos control en cuando se cargan los registros**



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



Programabilidad: Señales de Control

- A estas señales que nos permiten decidir qué operación y cuando almacenar las llamaremos **señales de control**
- Las secuencias de señales de control descritas en la tabla las llamaremos **instrucciones**

la	lb	s2	s1	s0	operación
1	0	0	0	0	$A=A+B$
0	1	0	0	0	$B=A+B$
1	0	0	0	1	$A=A-B$
0	1	0	0	1	$B=A-B$
1	0	0	1	0	$A=A \text{ and } B$
0	1	0	1	0	$B=A \text{ and } B$
1	0	0	1	1	$A=A \text{ or } B$
0	1	0	1	1	$B=A \text{ or } B$
1	0	1	0	0	$A=\text{not } A$
0	1	1	0	0	$B=\text{not } A$
1	0	1	0	1	$A=A \text{ xor } B$
0	1	1	0	1	$B=A \text{ xor } B$
1	0	1	1	0	$A=\text{shift left } A$
0	1	1	1	0	$B=\text{shift left } A$
1	0	1	1	1	$A=\text{shift right } A$
0	1	1	1	1	$B=\text{shift right } A$

¿Dudas?

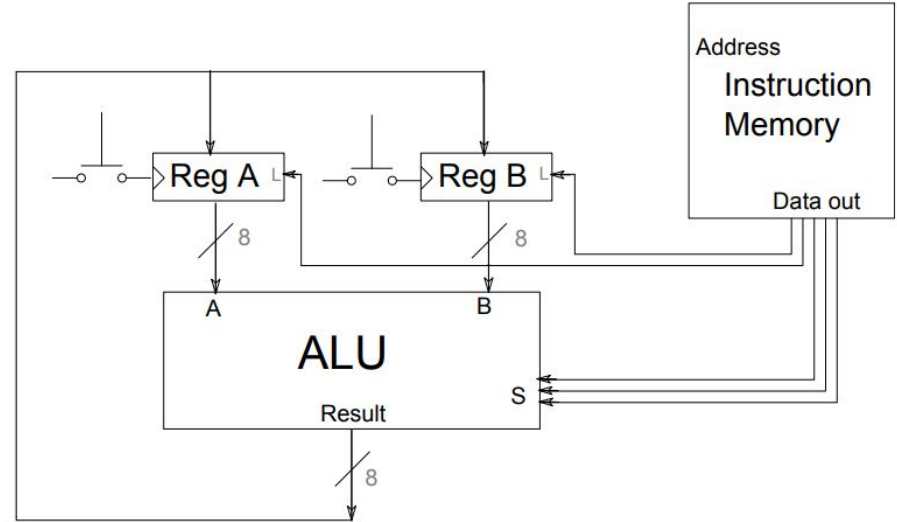
Programabilidad: Instrucciones (ejemplo)

- A partir de un conjunto de instrucciones podemos construir un **programa**
- En la tabla podemos tener un ejemplo de calcular números de Fibonacci (asumimos que registro B parte en 1)

la	lb	s2	s1	s0	operación	A	B
0	0	-	-	-	-	0	1
1	0	0	0	0	A=A+B	1	1
0	1	0	0	0	B=A+B	1	2
1	0	0	0	0	A=A+B	3	2
0	1	0	0	0	B=A+B	3	5
1	0	0	0	0	A=A+B	8	5
0	1	0	0	0	B=A+B	8	13

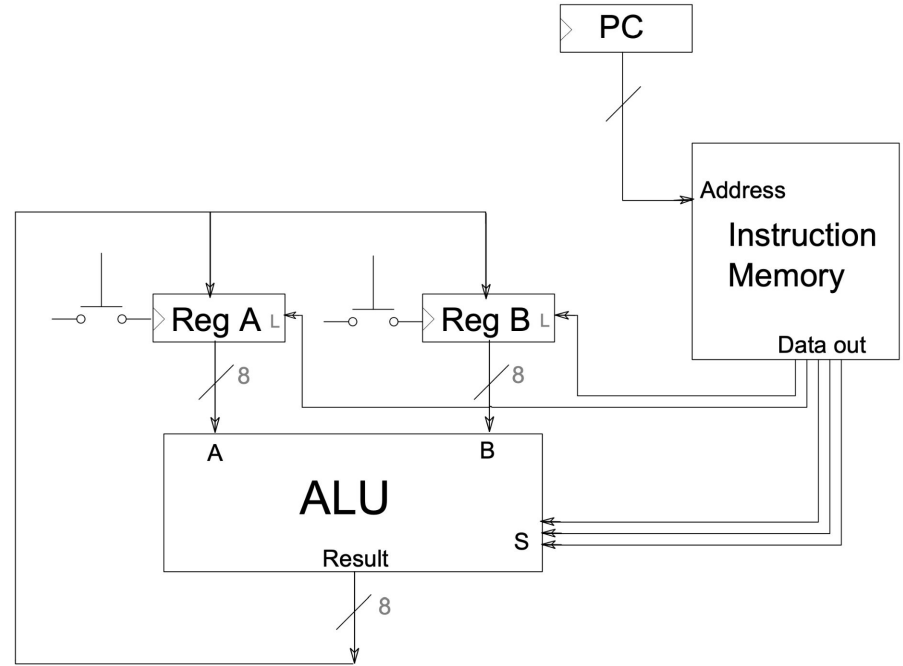
Programabilidad: Almacenamiento de Instr.

- A partir de un conjunto de instrucciones podemos construir un **programa**
- En la tabla podemos tener un ejemplo de calcular números de Fibonacci (asumimos que registro B parte en 1)



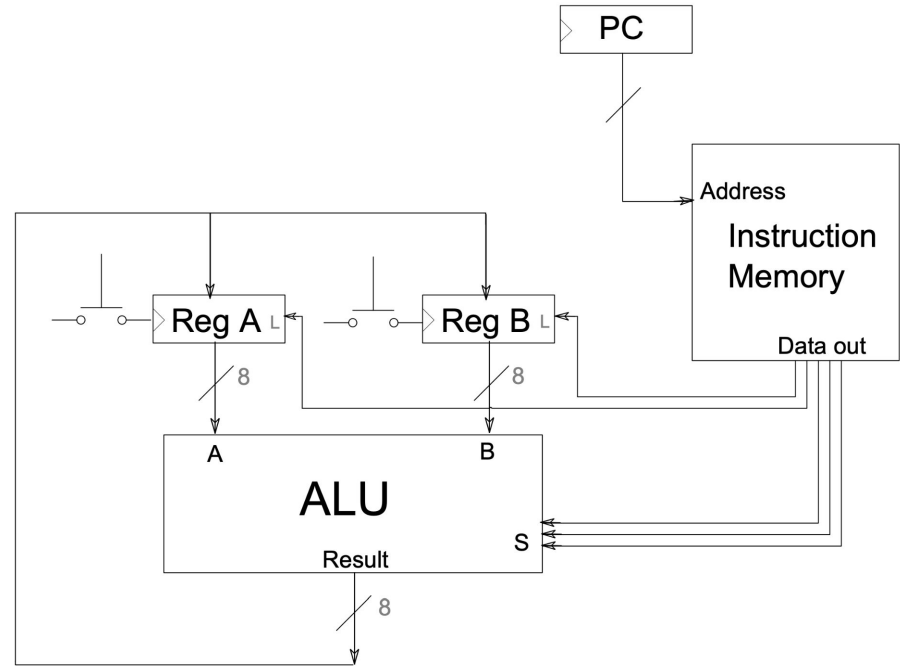
Programabilidad: Direcccionamiento de Instr.

- Podemos automatizar la lectura de instrucciones con registro un **contador**
- Este registro lo llamaremos **Program Counter (PC)**



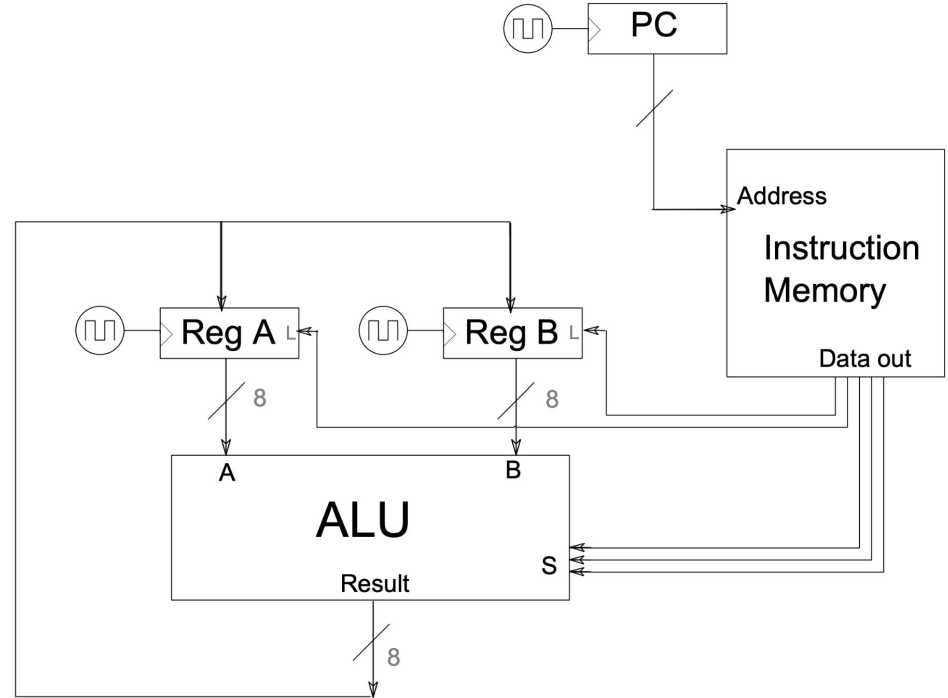
Programabilidad: Direccionamiento de Instr.

PC	Inst	Operación
0000	10000	$A = A + B$
0001	01000	$B = A + B$
0010	10000	$A = A + B$



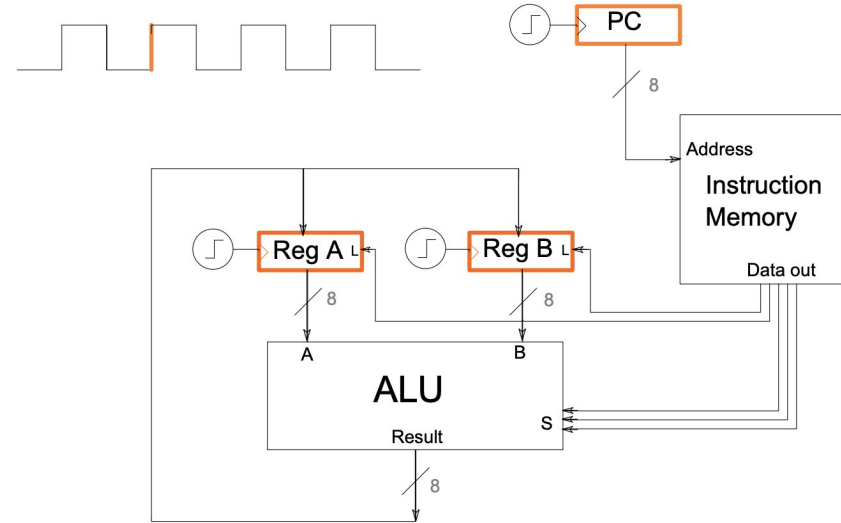
Programabilidad: Sincronización

- Para sincronizar todo el proceso ocupamos un **único reloj**
- Todos los registros dentro de un mismo computador operan bajo el mismo reloj



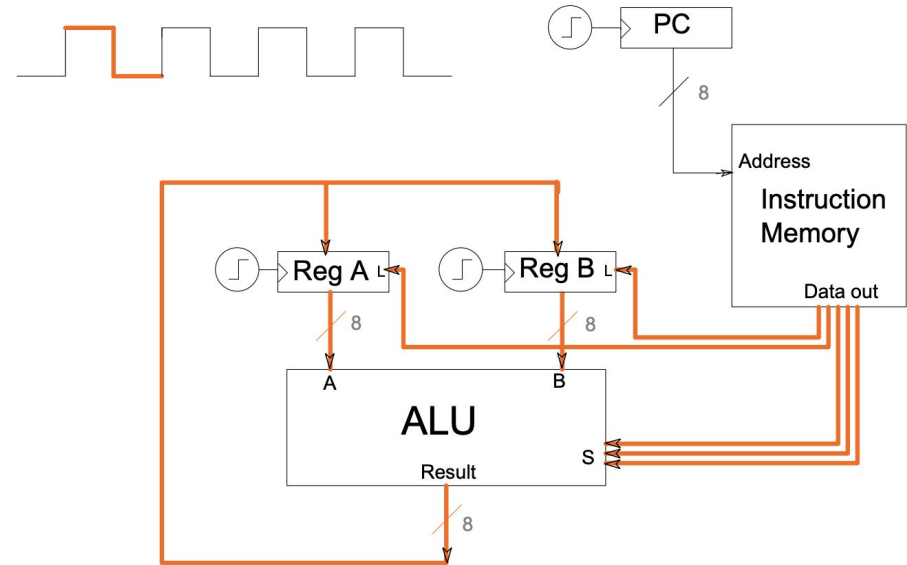
Programabilidad: Sincronización (ejemplo)

- Los registros son activados en flanco de subida, guardando los resultados si corresponden
- Recordar que dentro de los registros tenemos Flip Flop D que funcionan en base al **flanco de subida**



Programabilidad: Sincronización (ejemplo)

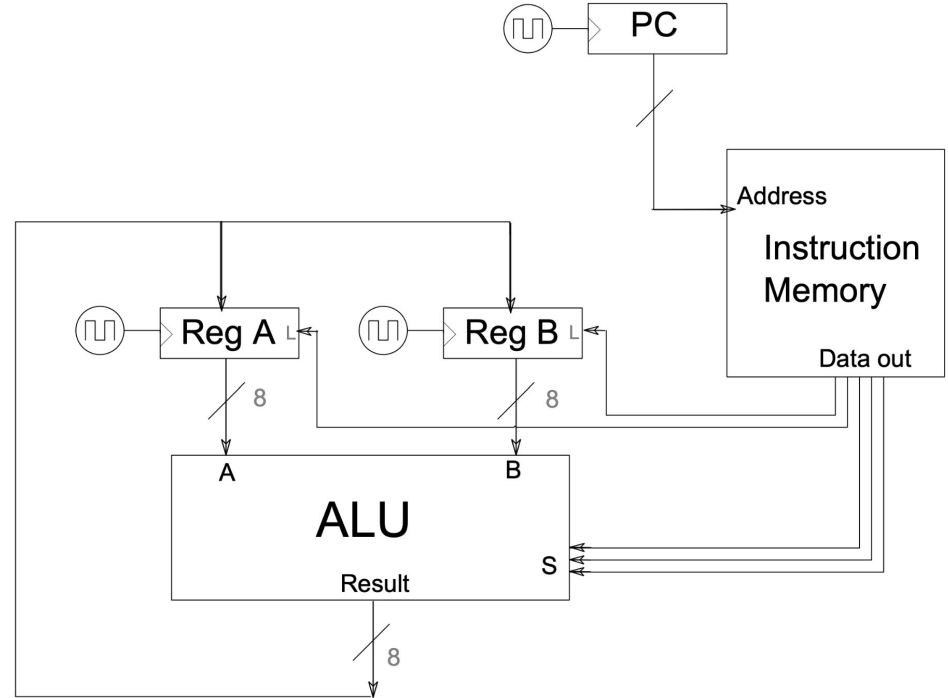
- Cuando el clock subió y luego cuando baja, el program counter
- Los registros no se actualizan, dejando tiempo para procesar la instrucción y ejecutarla



¿Dudas?

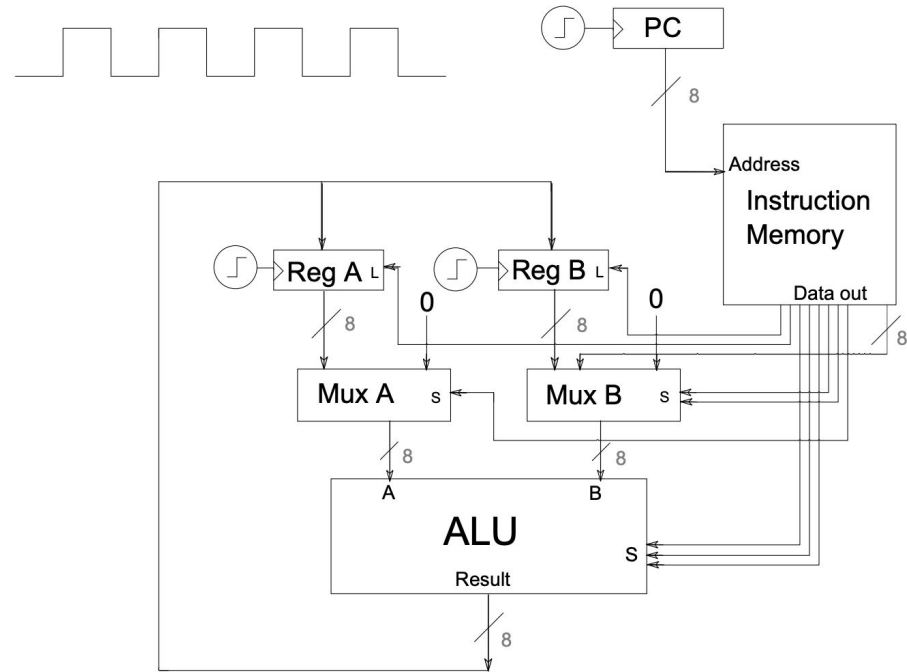
Computador básico: Limitaciones

- No podemos inicializar los registros con valores iniciales. Hasta ahora asumimos que parten con un valor
- No podemos almacenar más de dos datos simultáneamente.
- **¿Cómo se soluciona?**



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



Computador básico: Señales de control

La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
1	0	1	0	0	0	0	0	A=B
0	1	0	1	1	0	0	0	B=A
1	0	1	0	1	0	0	0	A=Lit
0	1	1	0	1	0	0	0	B=Lit
1	0	0	0	0	0	0	0	A=A+B
0	1	0	0	0	0	0	0	B=A+B
1	0	0	0	1	0	0	0	A=A+Lit
1	0	0	0	0	0	0	1	A=A-B
0	1	0	0	0	0	0	1	B=A-B
1	0	0	0	1	0	0	1	A=A-Lit
1	0	0	0	0	0	1	0	A=A and B
0	1	0	0	0	0	1	0	B=A and B
1	0	0	0	1	0	1	0	A=A and Lit

La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
1	0	0	0	0	0	1	1	A=A or B
0	1	0	0	0	0	1	1	B=A or B
1	0	0	0	1	0	1	1	A=A or Lit
1	0	0	0	0	1	0	0	A=not A
0	1	0	0	0	1	0	0	B=not A
1	0	0	0	0	1	0	1	A=A xor B
0	1	0	0	0	1	0	1	B=A xor B
1	0	0	0	1	1	0	1	A=A xor Lit
1	0	0	0	0	1	1	0	A=shift left A
0	1	0	0	0	1	1	0	B=shift left A
1	0	0	0	0	1	1	1	A=shift right A
0	1	0	0	0	1	1	1	B=shift right A
-	-	-	-	-	-	-	-	---

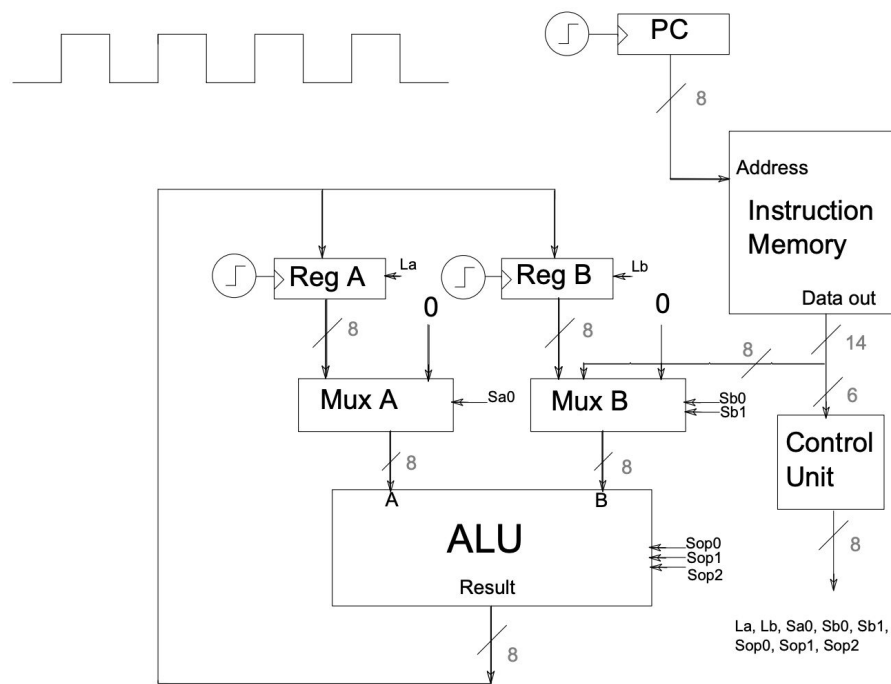
Computador básico: Opcodes

- 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

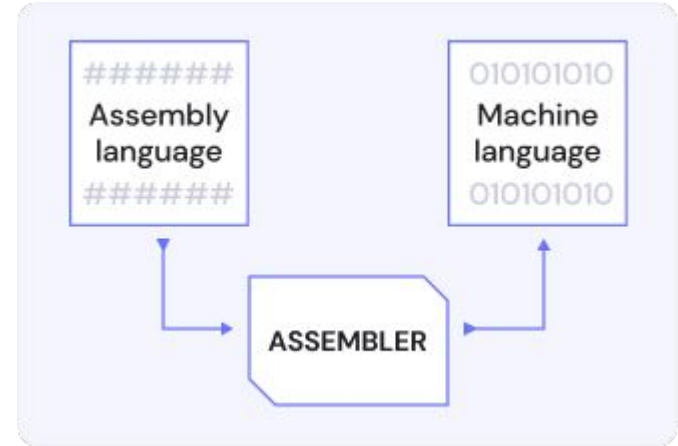
Computador básico: Unidad de Control

- El componente que se encarga de la decodificación se llamará **Unidad de Control**
- Se encargará de indicar directamente las señales de control



Computador básico: Assembler

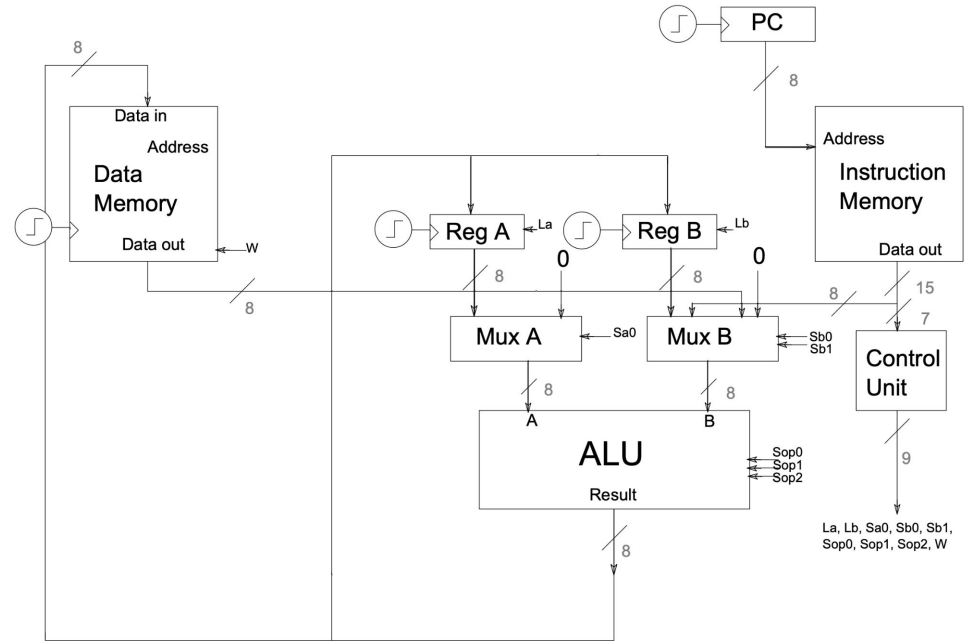
- La conversión entre las palabras del assembly y los opcodes la realiza el programa denominado **Assembler**
- Un **compilador** será el programa encargado de convertir un lenguaje de alto nivel en el assembly de la máquina



¿Dudas?

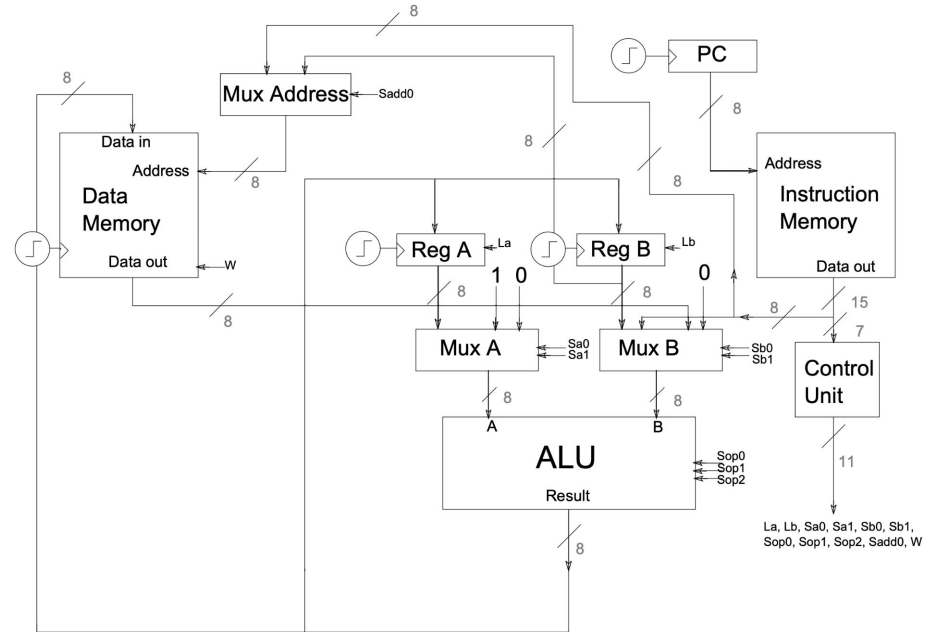
Computador básico: Memoria de datos

- Agregamos una Memoria de Datos, comunmente referida como **RAM**
- Permitirá el manejo de variables temporales
- Se sincroniza con el mismo reloj que los registros



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



Computador básico: Ins. de Direccionamiento

- La notación será con paréntesis para indicar la dirección de memoria **(Dir)** o **(B)**
- Se puede incrementar el valor de B más fácilmente

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
MOV	A,(Dir)	A=Mem[Dir]		MOV A,(var1)
	B,(Dir)	B=Mem[Dir]		MOV B,(var2)
	(Dir),A	Mem[Dir]=A		MOV (var1),A
	(Dir),B	Mem[Dir]=B		MOV (var2),B
	A,(B)	A=Mem[B]		-
	B,(B)	B=Mem[B]		-
	(B),A	Mem[B]=A		-
ADD	A,(Dir)	A=A+Mem[Dir]		ADD A,(var1)
	A,(B)	A=A+Mem[B]		-
SUB	(Dir)	Mem[Dir]=A+B		ADD (var1)
	A,(Dir)	A=A-Mem[Dir]		SUB A,(var1)
	A,(B)	A=A-Mem[B]		-
AND	(Dir)	Mem[Dir]=A-B		SUB (var1)
	A,(Dir)	A=A and Mem[Dir]		AND A,(var1)
	A,(B)	A=A and Mem[B]		-
OR	(Dir)	Mem[Dir]=A and B		-
	A,(Dir)	A=A or Mem[Dir]		OR A,(var1)
	A,(B)	A=A or Mem[B]		-
NOT	(Dir)	Mem[Dir]=A or B		OR (var1)
	(Dir)	Mem[Dir]=not A		NOT (var1)
XOR	A,(Dir)	A=A xor Mem[Dir]		XOR A,(var1)
	A,(B)	A=A xor Mem[B]		-
	(Dir)	Mem[Dir]=A xor B		XOR (var1)
SHL	(Dir)	Mem[Dir]=shift left A		SHL (var1)
SHR	(Dir)	Mem[Dir]=shift right A		SHR(var1)
INC	B	B=B+1		-

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

¿Dudas?

Bibliografía

- Apuntes históricos. Hans Löbel, Alejandro Echeverría

05 - Programabilidad

⋮	Apuntes históricos	✓	⋮
⋮	🔗 01 - Representaciones Numéricas Parte 1 - Números Enteros.pdf	✓	⋮
⋮	🔗 02 - Representaciones Numéricas Parte 2 - Números Racionales.pdf	✓	⋮
⋮	🔗 03 - Operaciones Aritmeticas y Logicas.pdf	✓	⋮
⋮	🔗 04 - Almacenamiento de datos.pdf	✓	⋮
⋮	🔗 05 - Programabilidad.pdf	✓	⋮

Bibliografía

- <https://github.com/IIC2343/Syllabus-antiores/tree/main/Enunciados>

The screenshot shows the GitHub interface for the repository `IIC2343 / Syllabus-antiores`. The left sidebar displays the file tree with the `Enunciados` directory selected. The main area shows the commit history for `Syllabus-antiores / Enunciados`, listing 15 commits by user `Geermy`. The commits include updates to previous exams, assessments, and the deletion of a specific file.

Name	Last commit message	Last commit date
...		
Enunciados 2019-1	feat: add more solutions to previous exams; add previous auxiliary cl...	2 years ago
Enunciados 2019-2	feat: add more solutions to previous exams; add previous auxiliary cl...	2 years ago
Enunciados 2020-1	feat: add more solutions to previous exams; add previous auxiliary cl...	2 years ago
Enunciados 2021-1	add enunciados antiguos	2 years ago
Enunciados 2021-2	feat: add assessments for past semester	2 years ago
Enunciados 2022-1	feat: add assessments for past semester	2 years ago
Enunciados 2022-2	feat: add more solutions to previous exams; add previous auxiliary cl...	2 years ago
Enunciados 2023-1	feat: add assessments for past semester	2 years ago
Enunciados 2023-2	add missing test solutions for last semester	last year
Enunciados 2024-1	remove unused readme	10 months ago
Enunciados 2024-2	Delete Enunciados/Enunciados 2024-2/a	4 months ago
Compilado_IIC2343.pdf	Agregando compilado de Felipe	5 years ago

Introducción del curso:

- 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 ? ? ?
- Para poder tener esta última parte requerimos de comenzar el aprender sobre **¡saltos!**

Clase 05 - Programabilidad



Profesor: **IIC2343 - Arquitectura de Computadores**
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl