

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

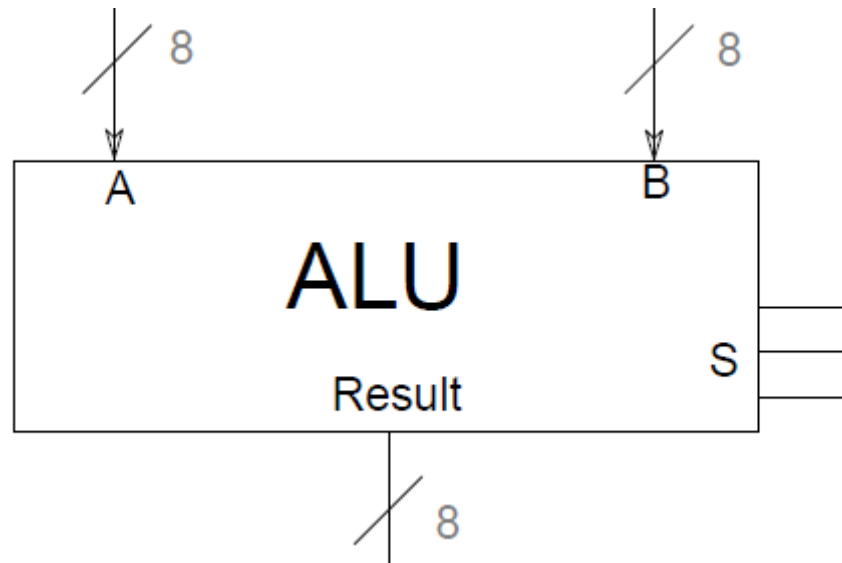


IIC2343 – Arquitectura de Computadores

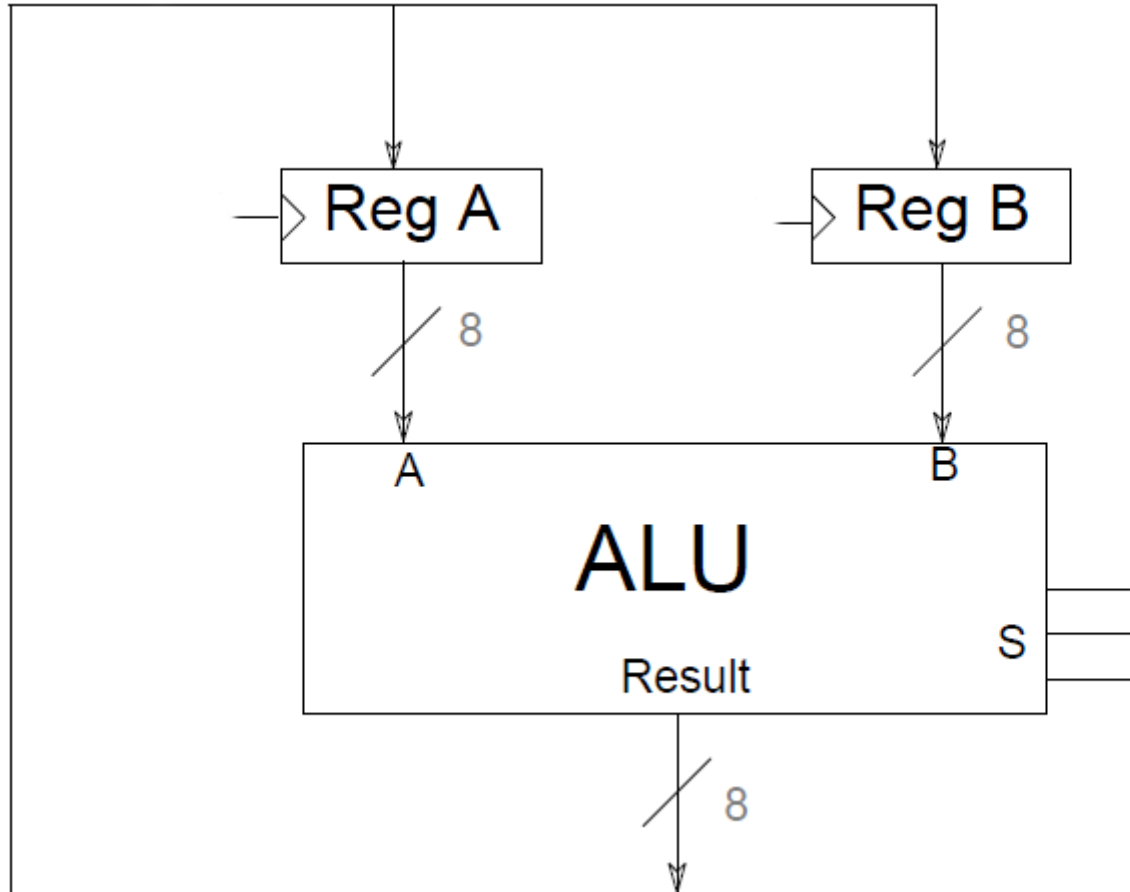
Programabilidad Parte 1 - Elementos Básicos

Profesor: Hans Löbel

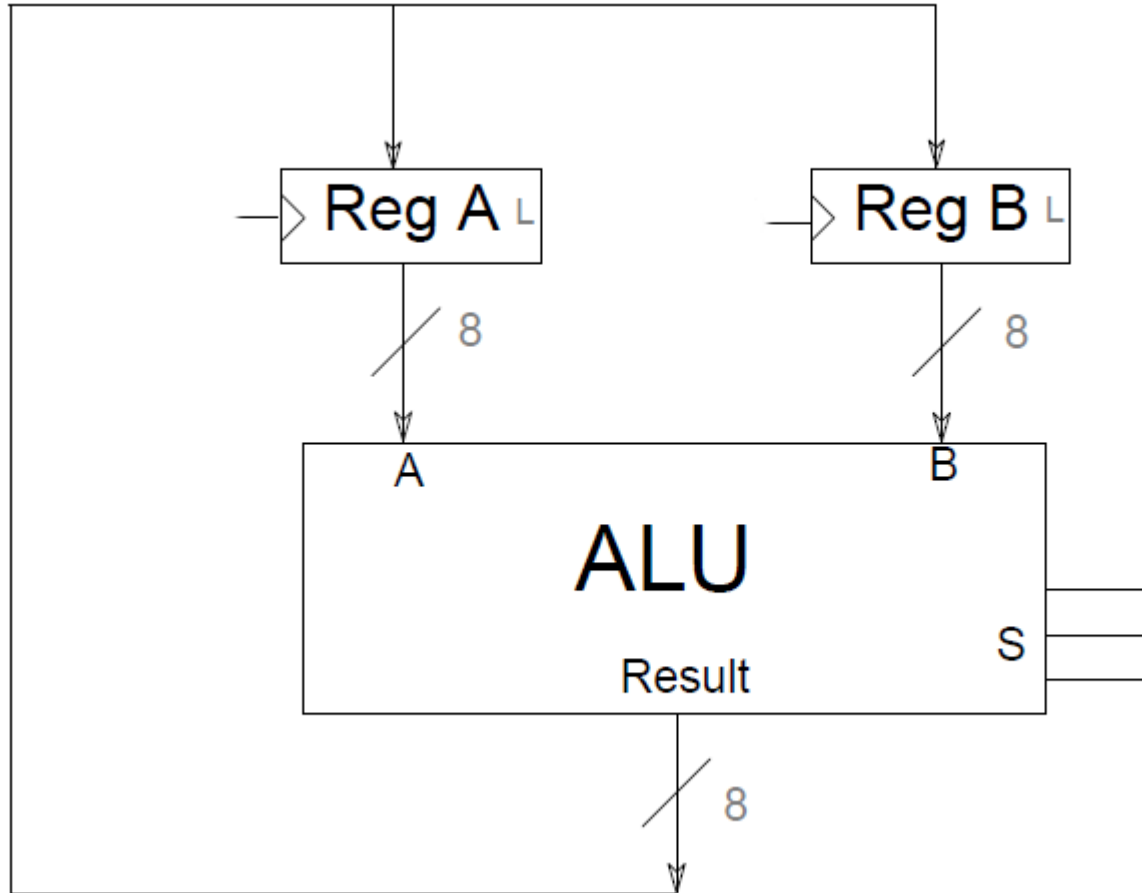
Ahora que tenemos **registros**, ¿cómo podemos mejorar la calculadora?



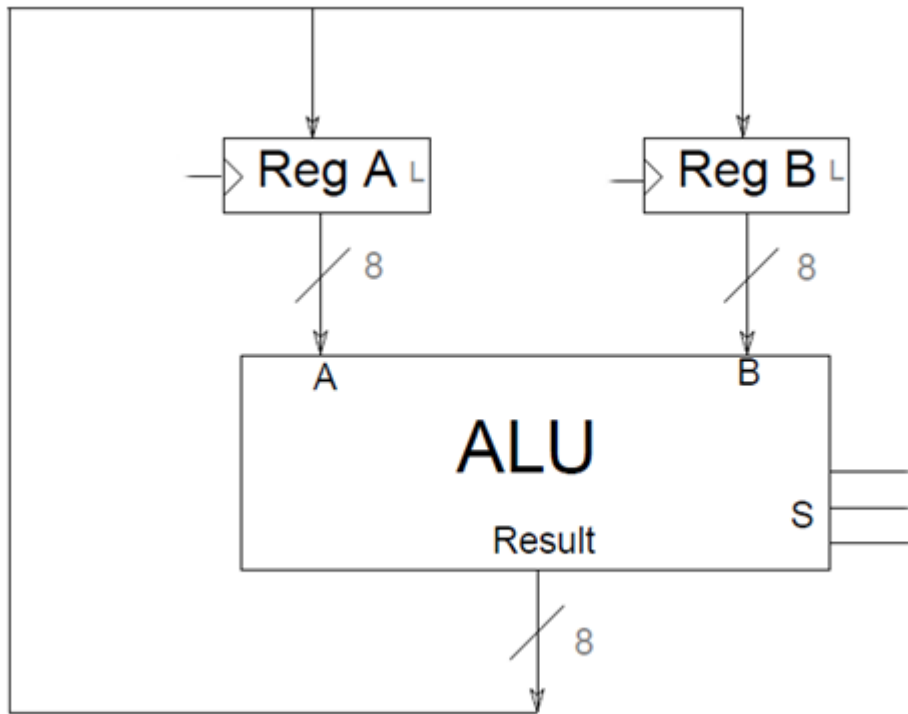
Unimos la salida de la ALU con la entrada de los registros



Con la señal **L** podemos controlar la escritura de los registros



Tenemos ahora un conjunto de **señales de control** para las distintas acciones



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 and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

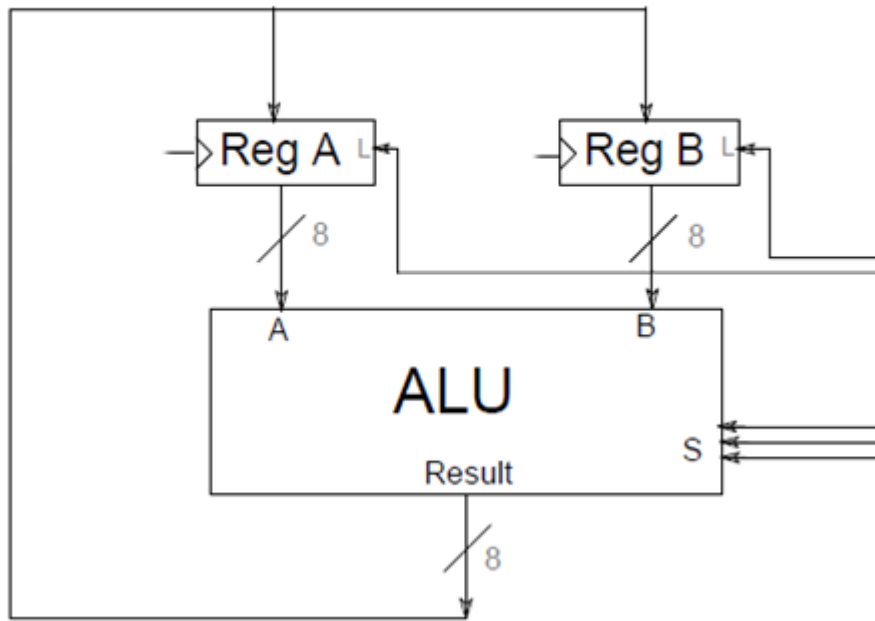
¿Qué hace esta secuencia de **señales de control**?

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

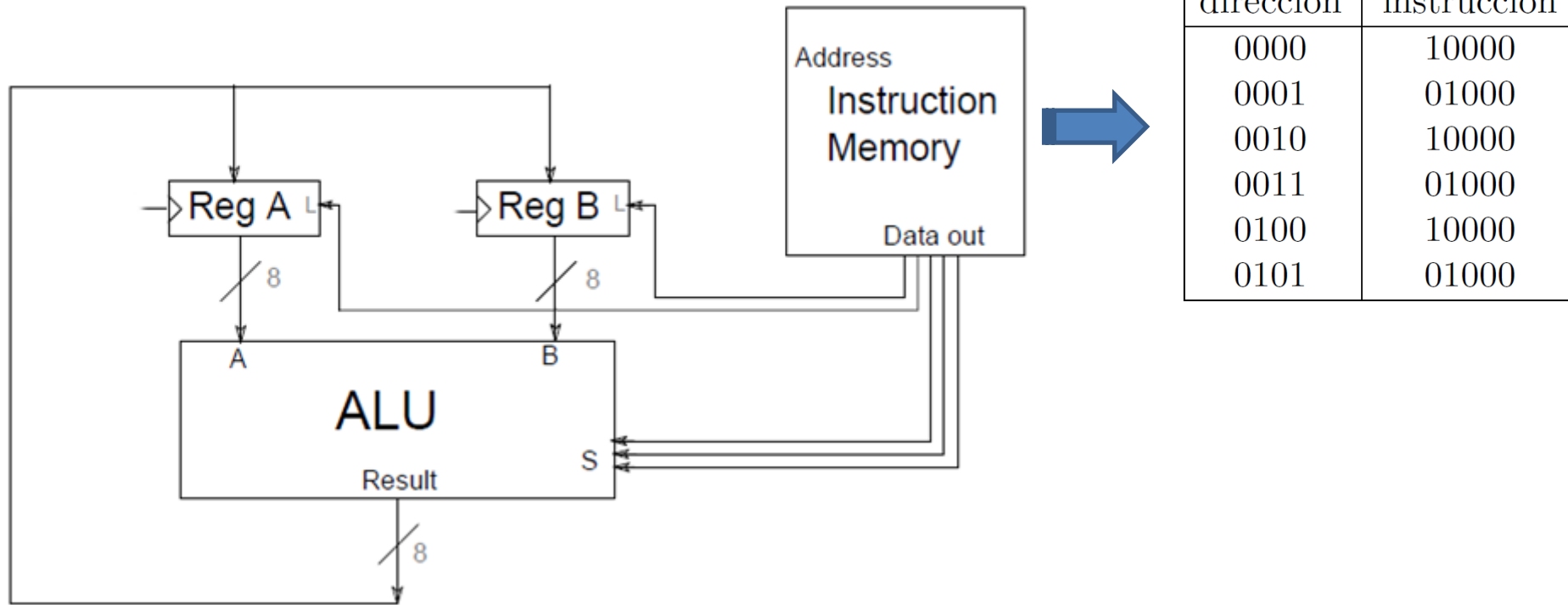
Cada **palabra de control** representa una **instrucción**,
y una secuencia de instrucciones es un **programa**

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

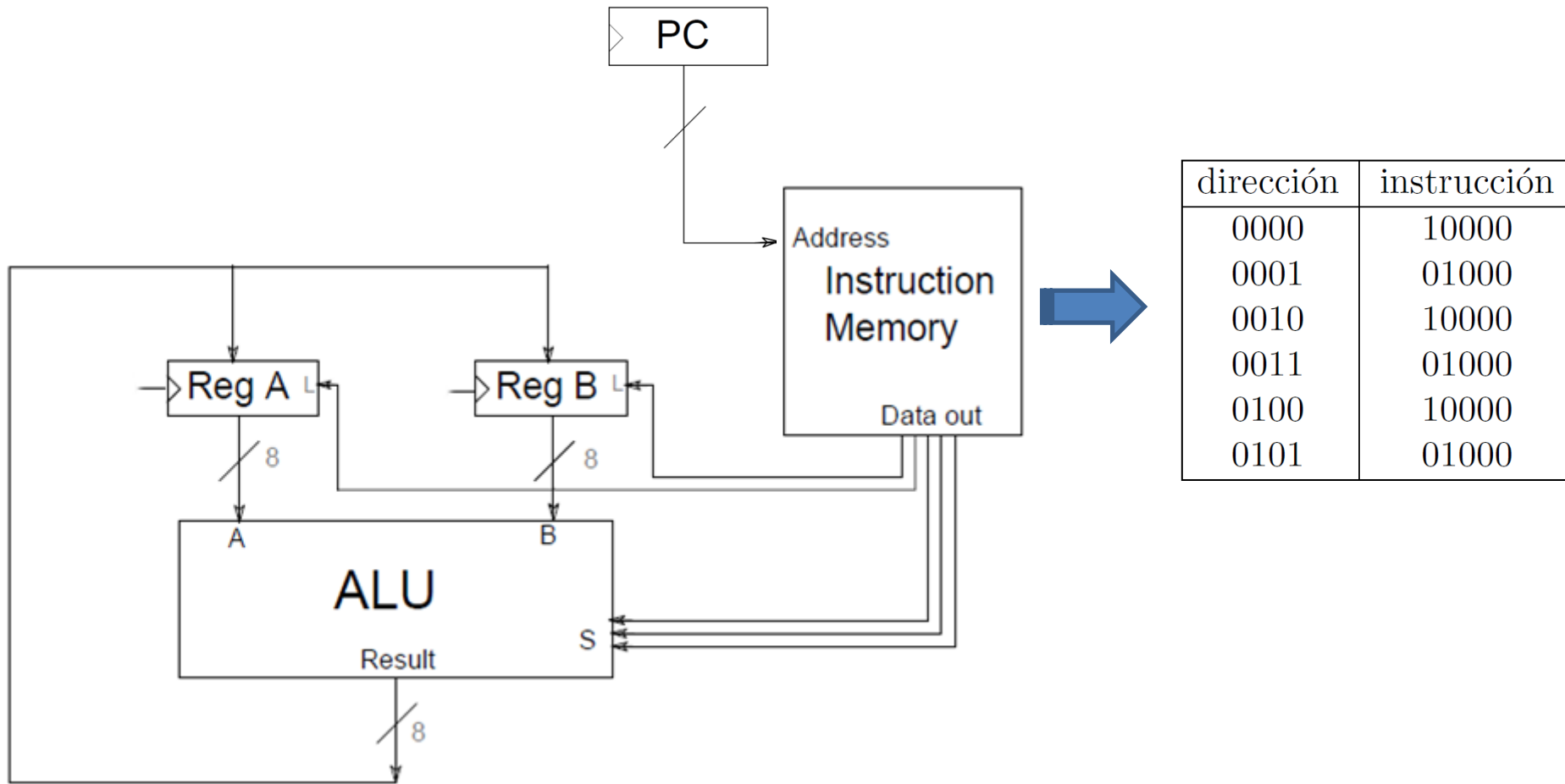
¿Cómo y donde podemos almacenar las instrucciones?



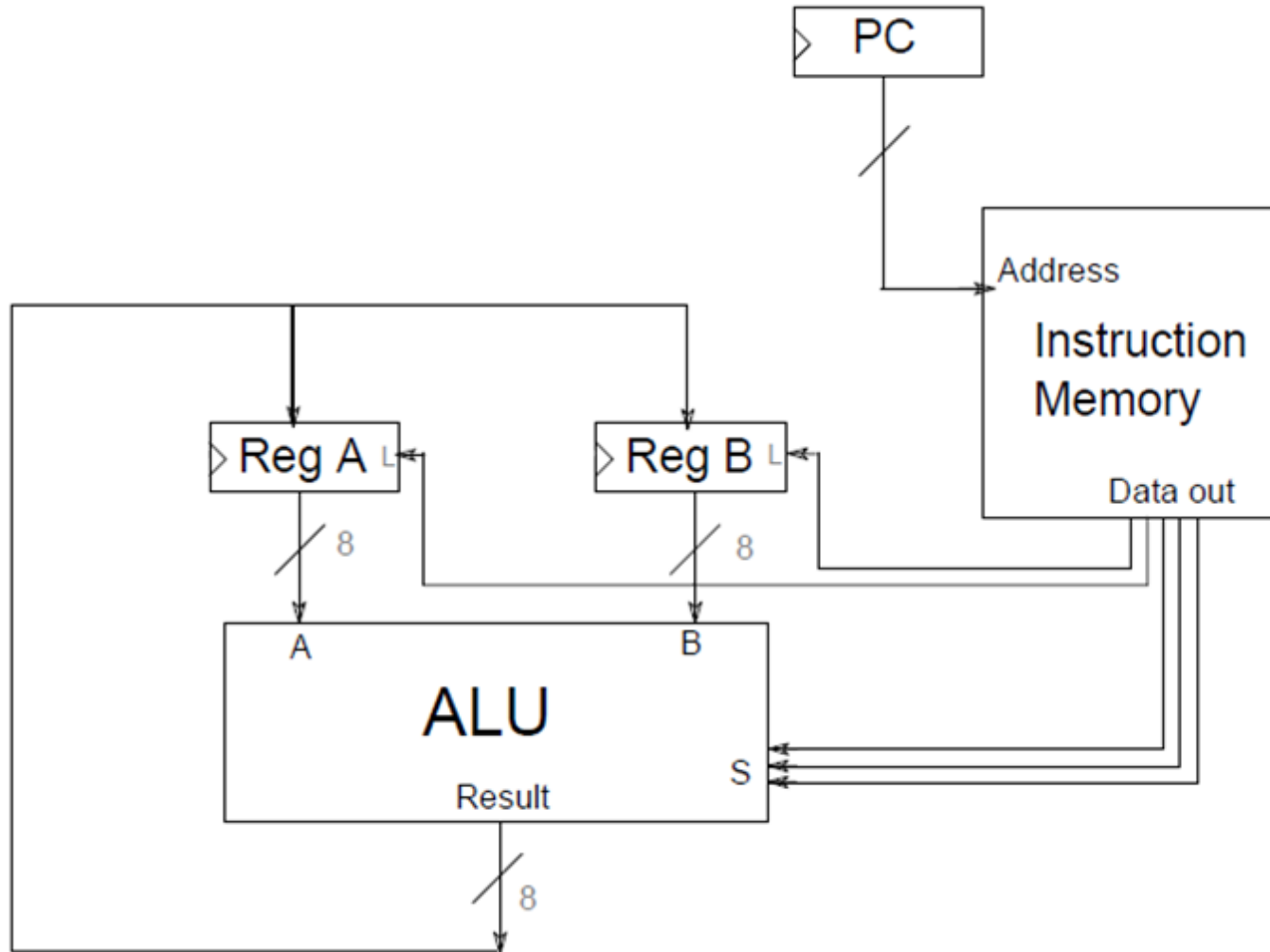
Memoria **ROM** nos permite almacenar la secuencia de instrucciones



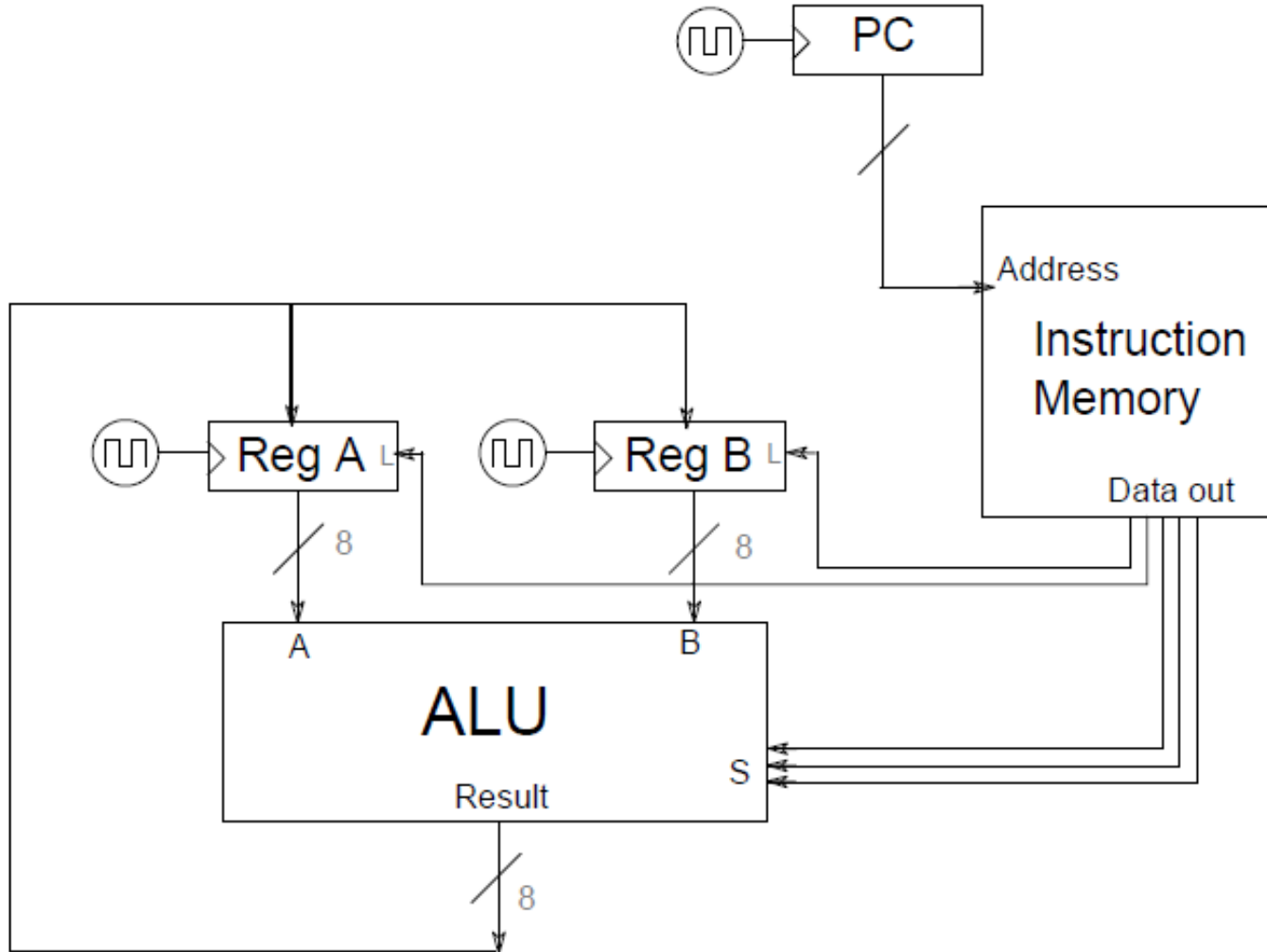
Necesitamos hacer **secuencial** la lectura de las instrucciones



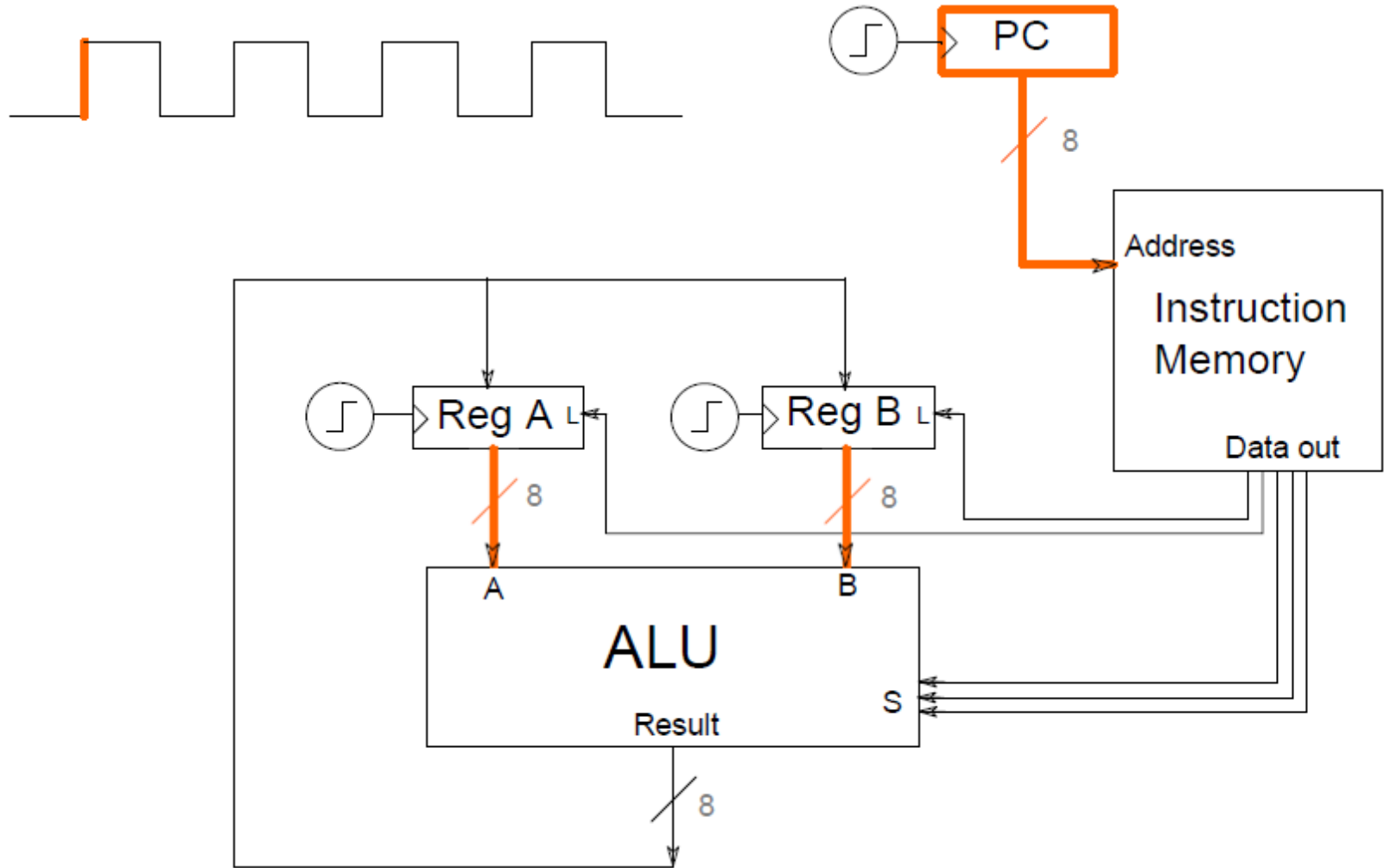
Falta un último paso para hacer que esta máquina sea automática



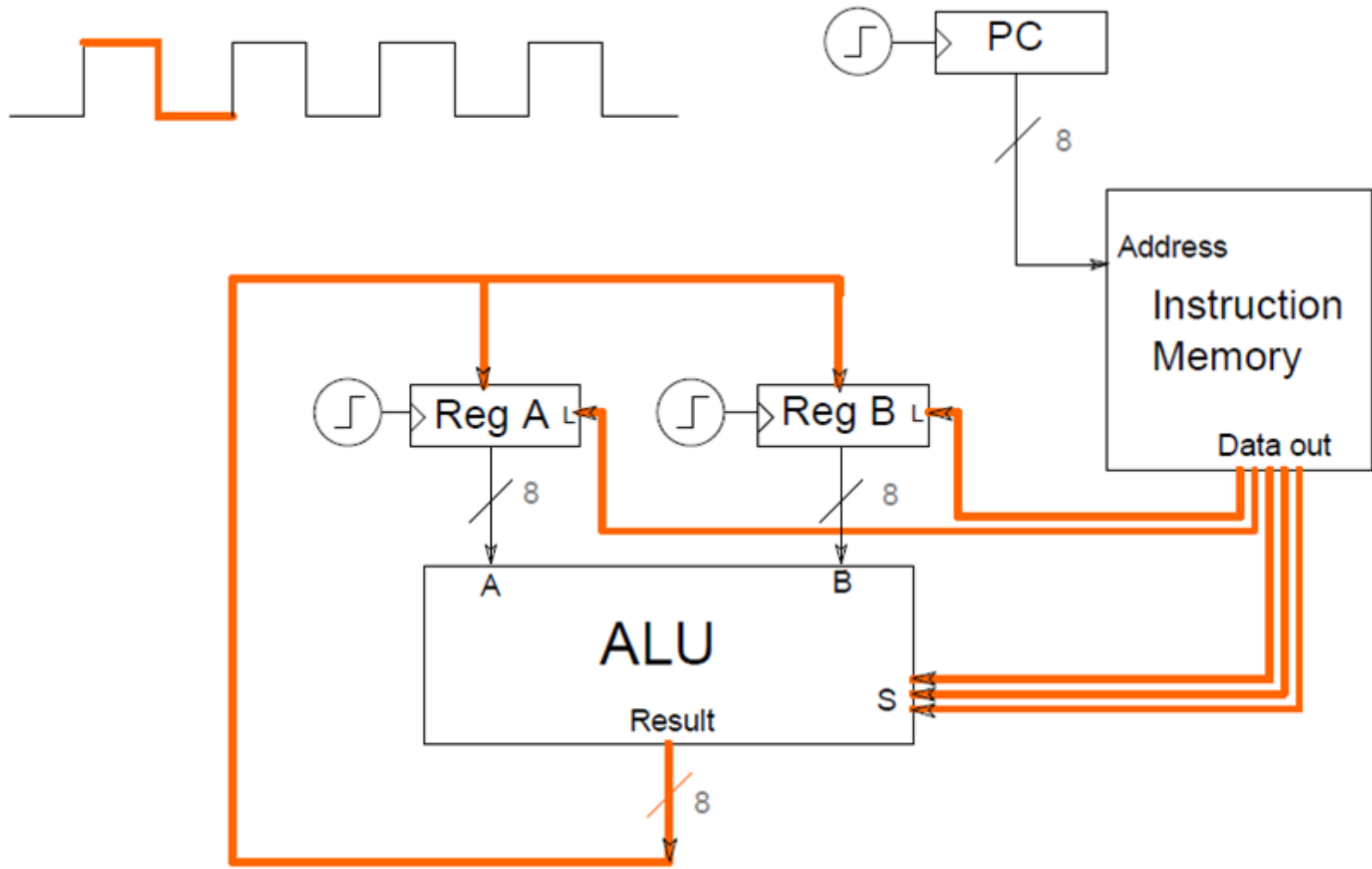
El último paso es sincronizar todos los elementos del computador con un **clock**



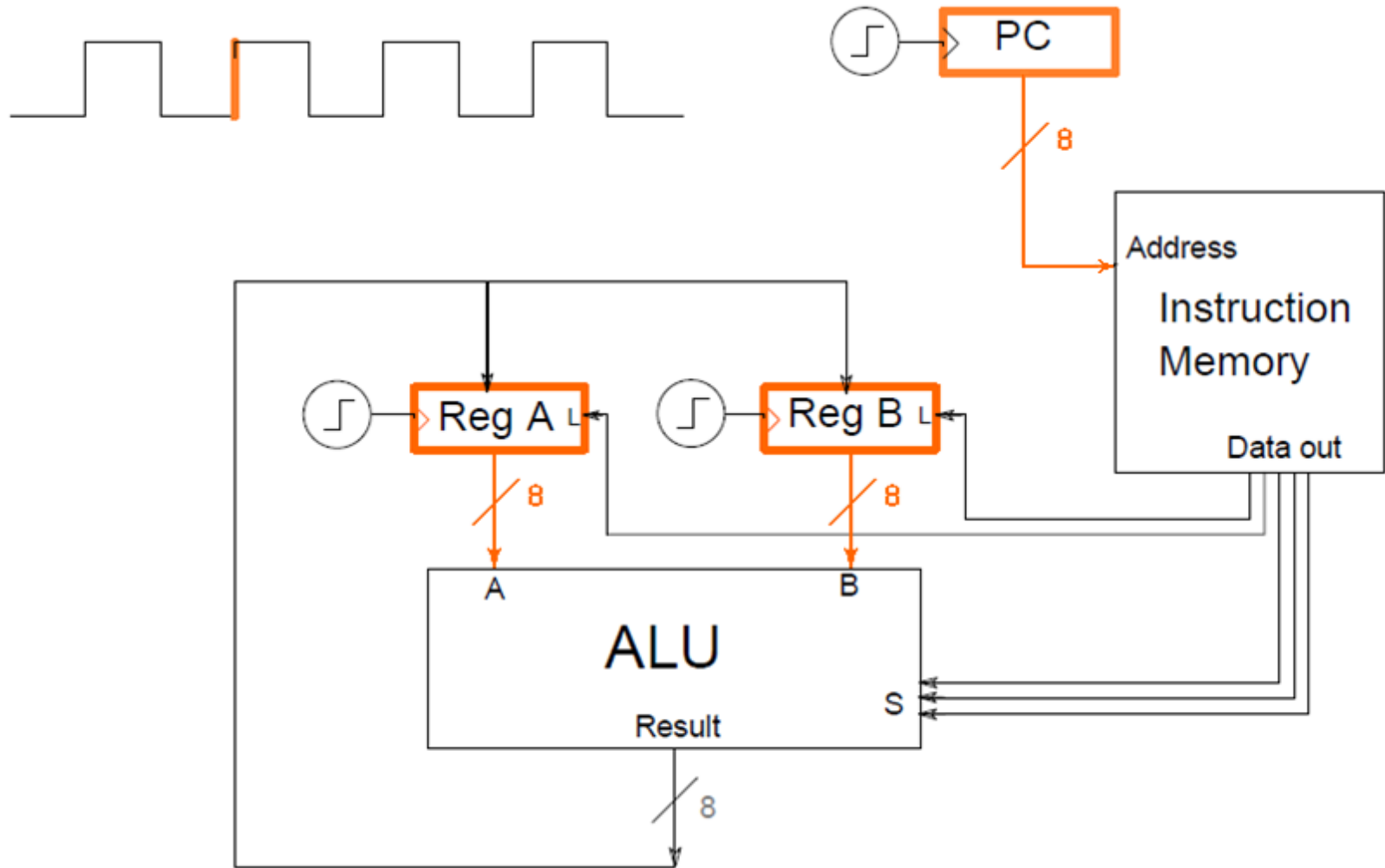
Revisemos cómo funciona ahora nuestra máquina programable



Revisemos cómo funciona ahora nuestra máquina programable



Revisemos cómo funciona ahora nuestra máquina programable



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

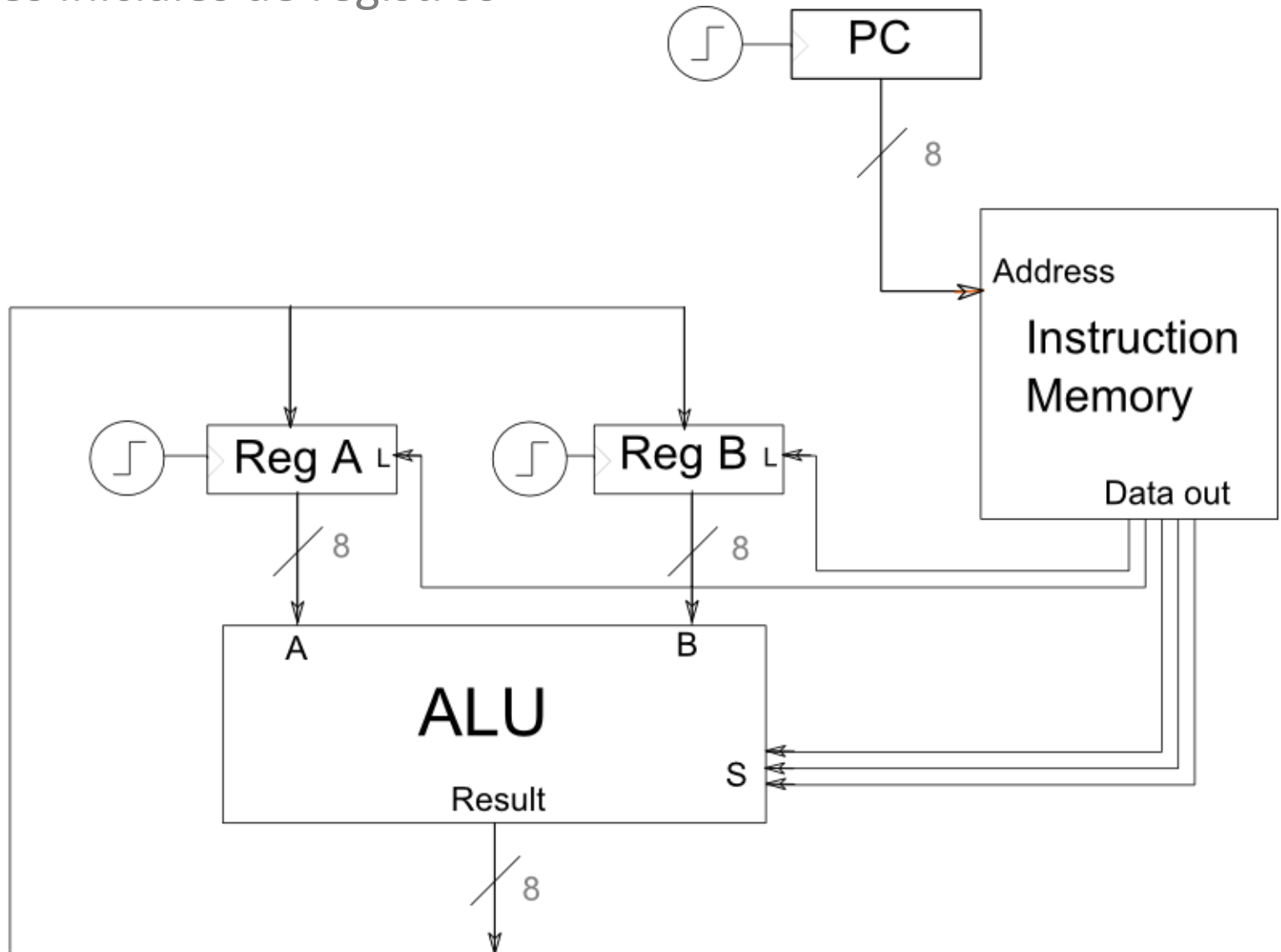


IIC2343 – Arquitectura de Computadores

Programabilidad Parte 1 - Elementos Básicos

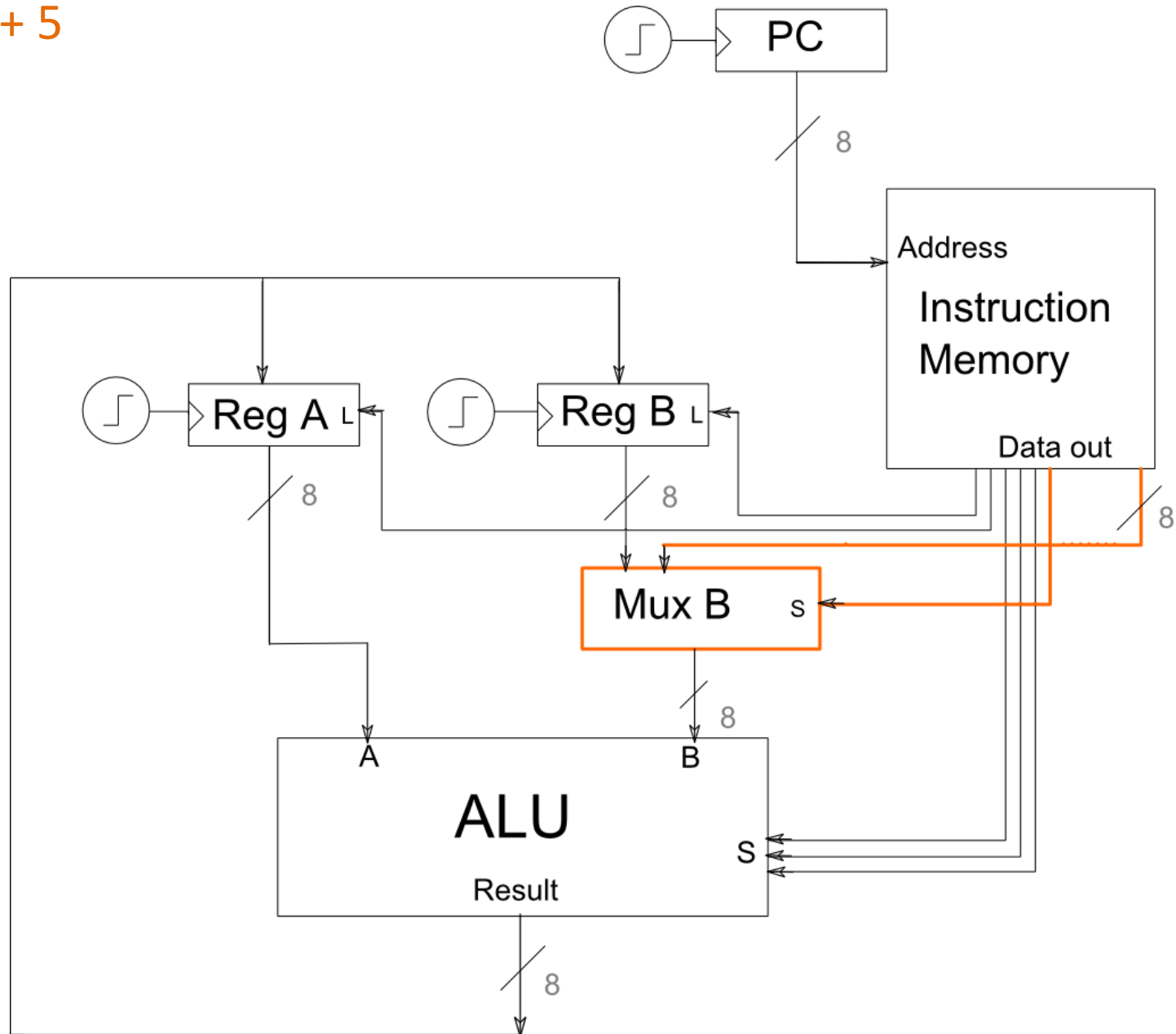
Profesor: Hans Löbel

Máquina programable aún está limitada
por valores iniciales de registros



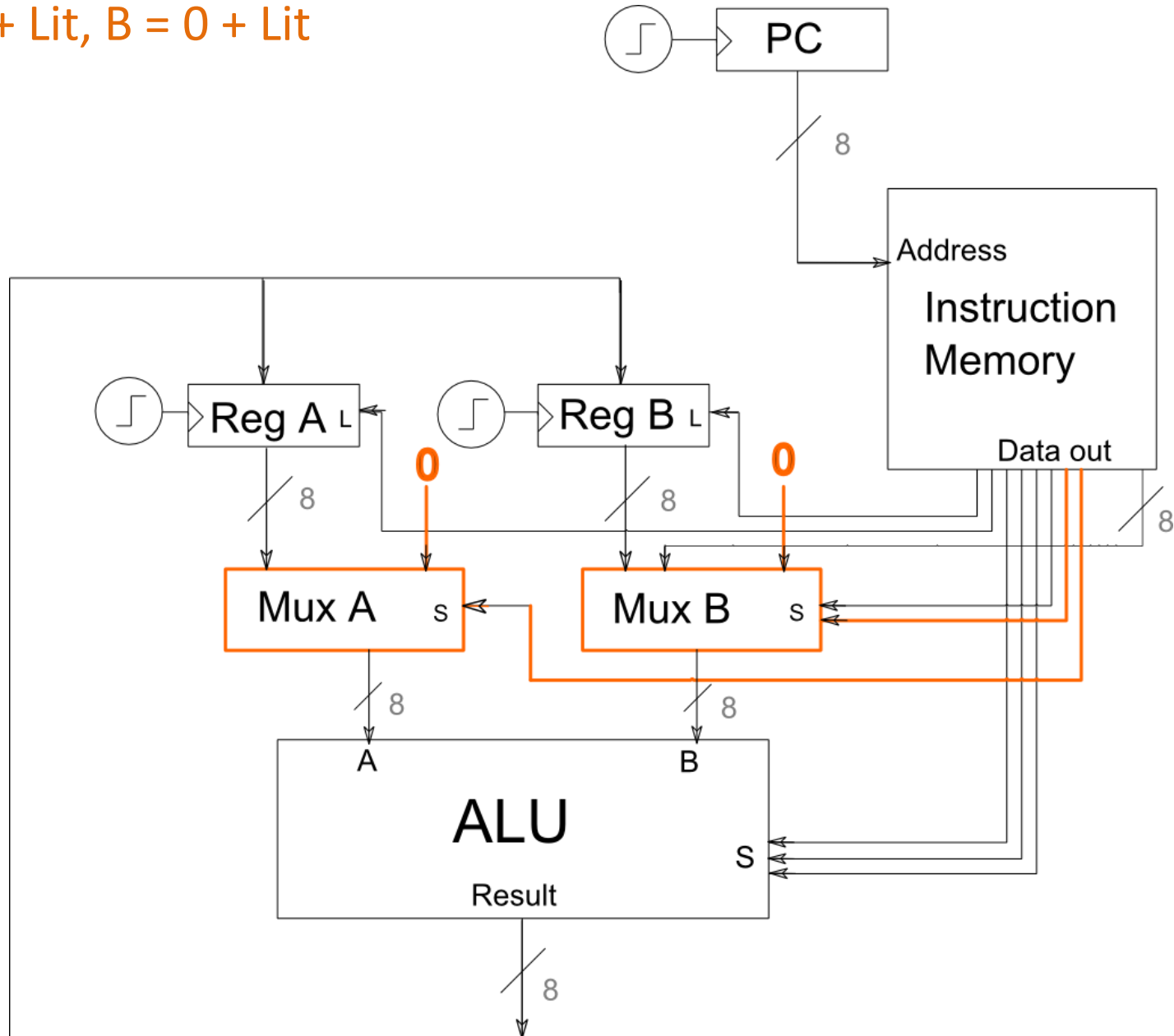
Literales aumentan poder expresivo

Ej.: $A = A + 5$



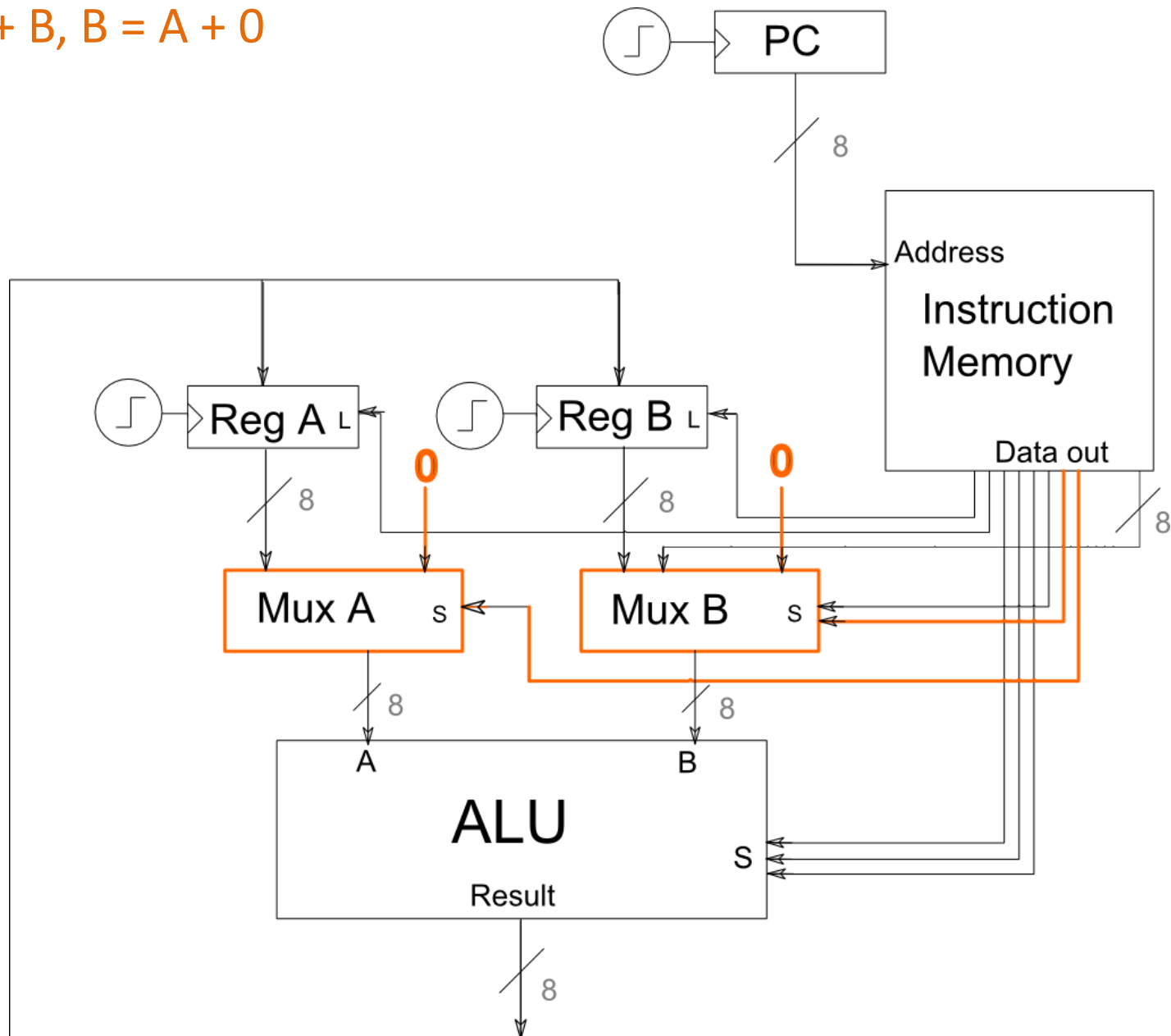
Ahora podemos inicializar registros

Ej.: $A = 0 + \text{Lit}$, $B = 0 + \text{Lit}$



Y también copiarlos

Ej.: $A = 0 + B$, $B = A + 0$



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	0	0	1	0	0	0	A=Lit
0	1	0	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
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=notA
0	1	0	0	0	1	0	0	B=notA
1	0	0	0	1	1	0	0	A=notLit
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	1	1	1	0	A=shift left Lit
1	0	0	0	0	1	1	1	A=shift right A
0	1	0	0	0	1	1	1	B=shift right A
1	0	0	0	1	1	1	1	A=shift right Lit

Tenemos palabras de control de 8 bits.

Pero tenemos sólo 28 instrucciones

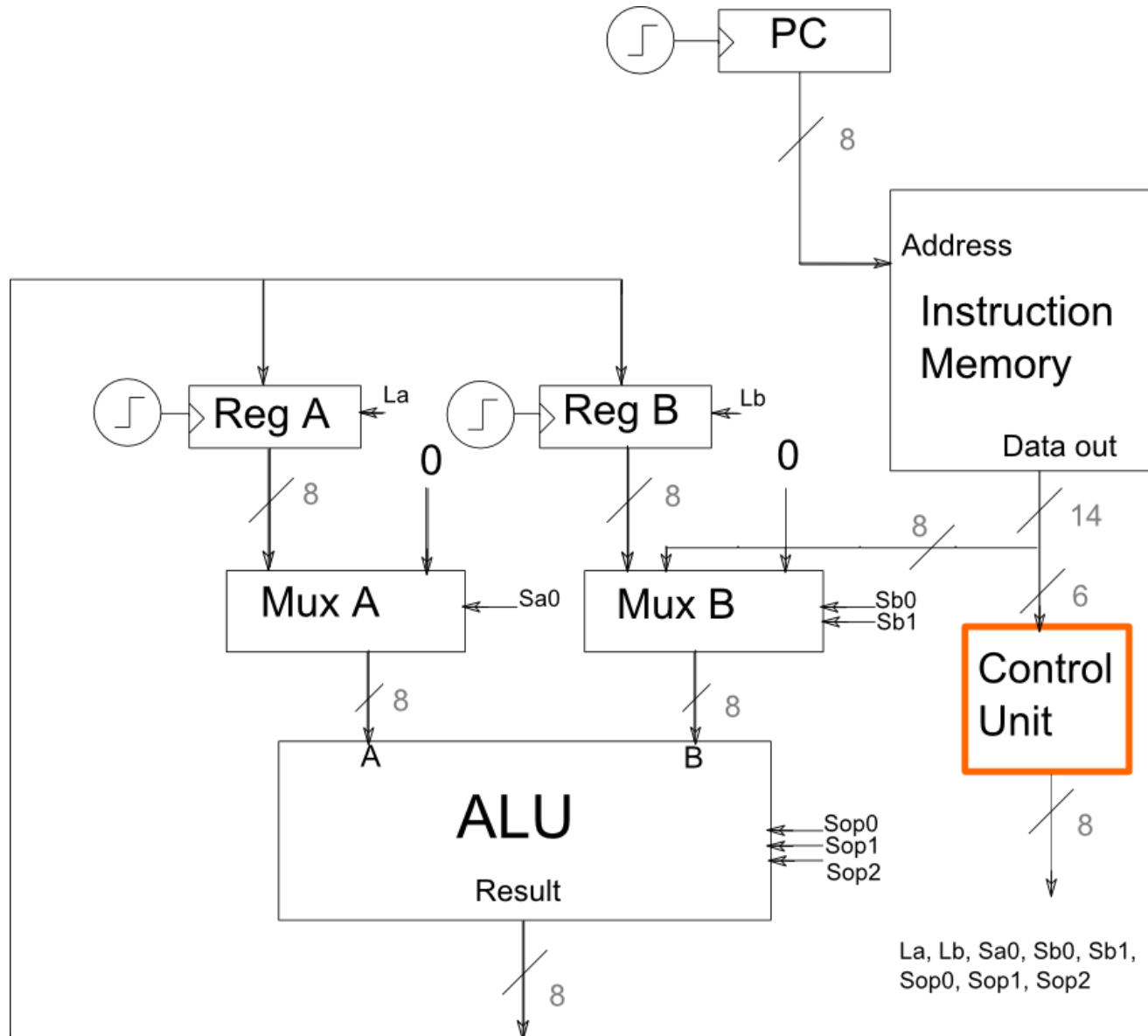
¿Qué podemos hacer para ahorrar espacio?

Solucionamos esto
usando **opcodes**.

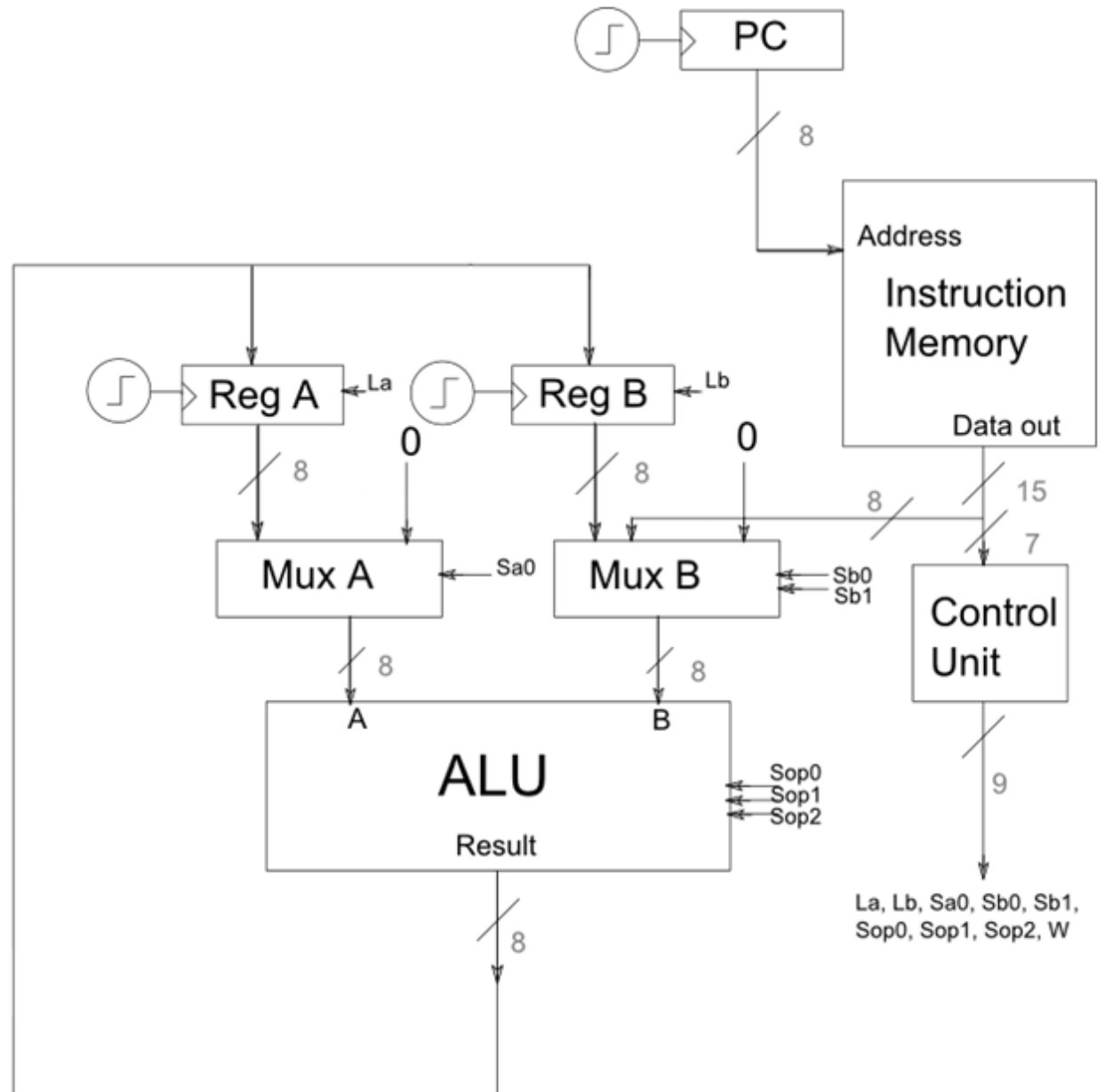
Cada uno se asocia a
una instrucción.

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	0	0	1	0	0	0	A=Lit
000011	0	1	0	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=notA
010001	0	1	0	0	0	1	0	0	B=notA
010010	1	0	0	0	1	1	0	0	A=notLit
010011	1	0	0	0	0	1	0	1	A=A xor B
010100	0	1	0	0	0	1	0	1	B=A xor B
010101	1	0	0	0	1	1	0	1	A=A xor Lit
010110	1	0	0	0	0	1	1	0	A=shift left A
010111	0	1	0	0	0	1	1	0	B=shift left A
011000	1	0	0	0	1	1	1	0	A=shift left Lit
011001	1	0	0	0	0	1	1	1	A=shift right A
011010	0	1	0	0	0	1	1	1	B=shift right A
011011	1	0	0	0	1	1	1	1	A=shift right Lit

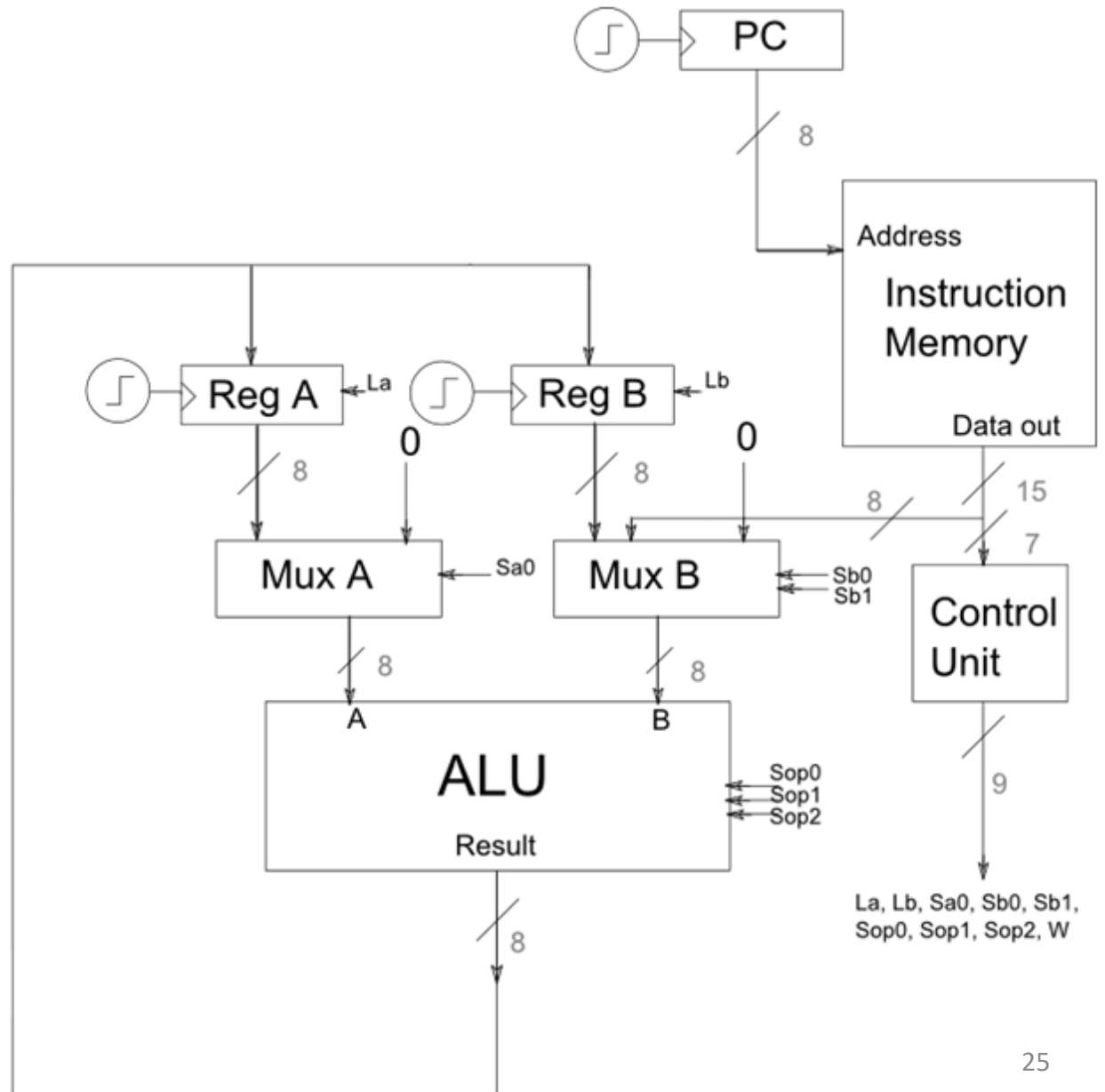
La **CU** traduce **opcodes** a señales de control y se implementa con una ROM



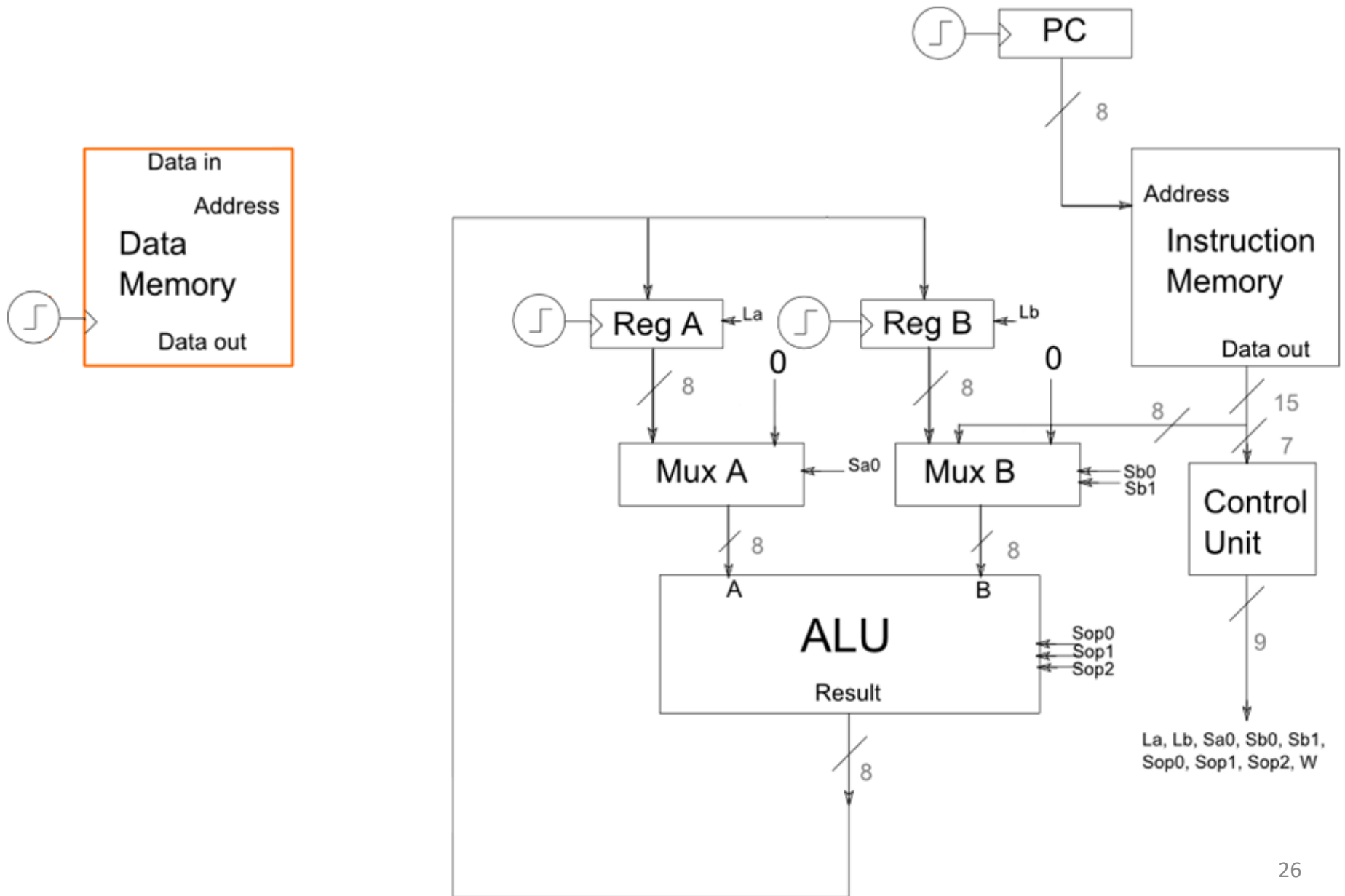
Aún necesitamos una manera para almacenar **mayor cantidad de datos**



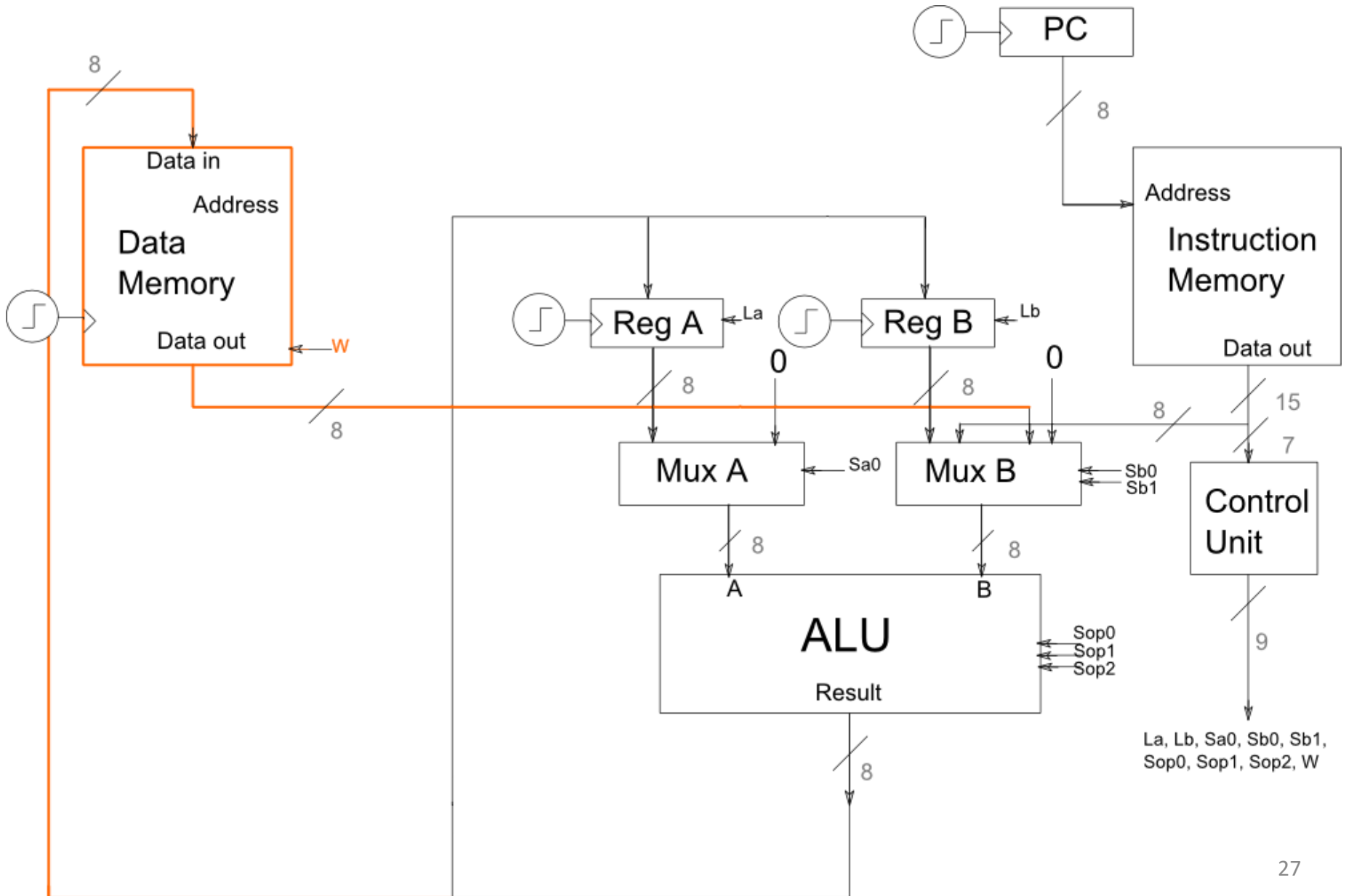
¿Donde podemos agregar una memoria?



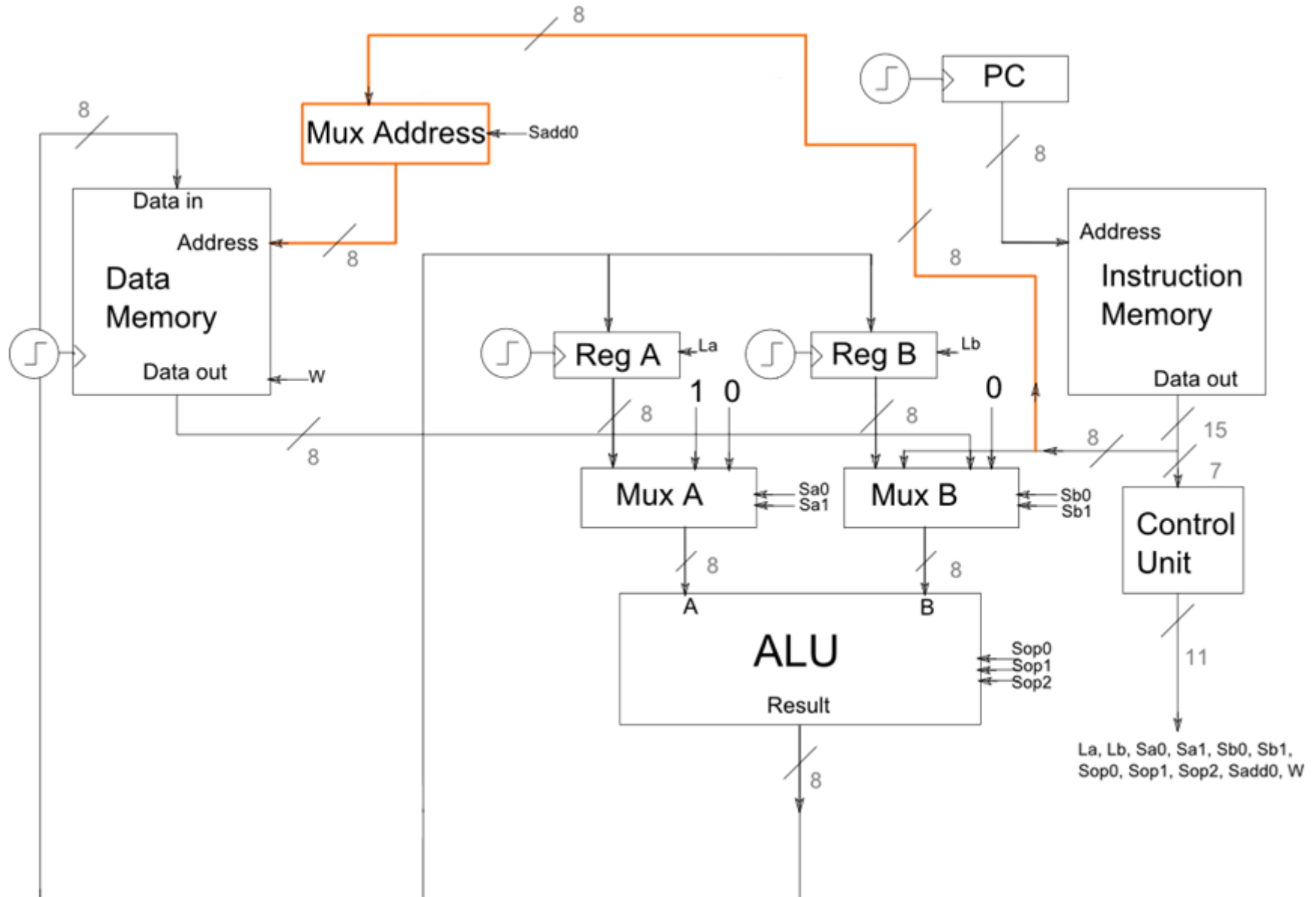
¿Donde podemos agregar una memoria?



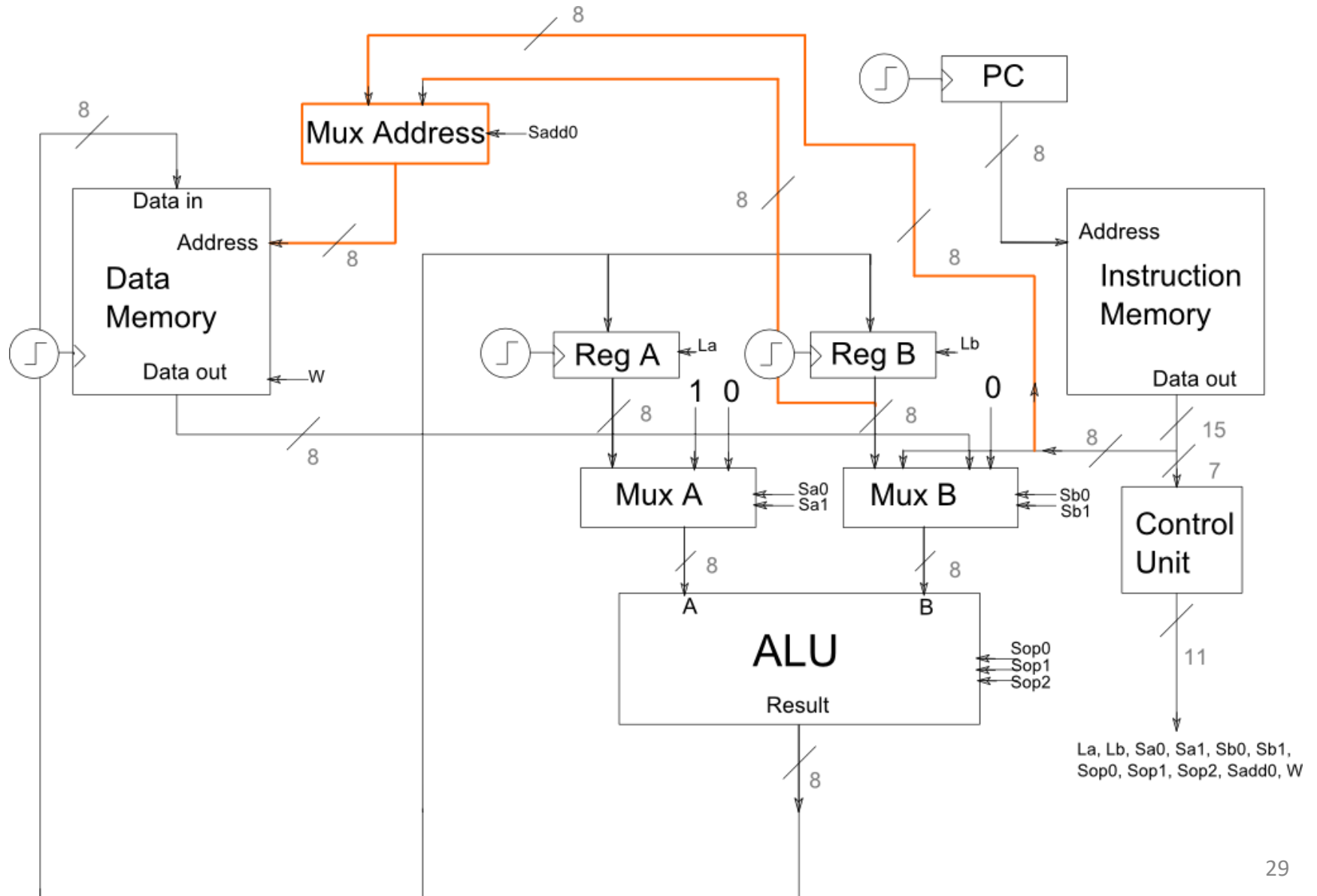
Ahora podemos almacenar variables



Podemos direccionar la memoria mediante un literal...



...o también mediante el registro B



¿Cómo podemos darle órdenes (**programar**) a la máquina programable?

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=notA
010001	0	1	0	0	0	1	0	0	B=notA
010010	1	0	0	0	1	1	0	0	A=notLit
010011	1	0	0	0	0	1	0	1	A=A xor B
010100	0	1	0	0	0	1	0	1	B=A xor B
010101	1	0	0	0	1	1	0	1	A=A xor Lit
010110	1	0	0	0	0	1	1	0	A=shift left A
010111	0	1	0	0	0	1	1	0	B=shift left A
011000	1	0	0	0	1	1	1	0	A=shift left Lit
011001	1	0	0	0	0	1	1	1	A=shift right A
011010	0	1	0	0	0	1	1	1	B=shift right A
011011	1	0	0	0	1	1	1	1	A=shift right Lit

Assembly nos permite trabajar de forma más natural y con menor **redundancia**

Instrucción	Operandos	Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
MOV	A,B	000000	1	0	1	0	0	0	0	0	A=B
	B,A	000001	0	1	0	1	1	0	0	0	B=A
	A,Lit	000010	1	0	0	0	1	0	0	0	A=Lit
	B,Lit	000011	0	1	0	0	1	0	0	0	B=Lit
ADD	A,B	000100	1	0	0	0	0	0	0	0	A=A+B
	B,A	000101	0	1	0	0	0	0	0	0	B=A+B
	A,Lit	000110	1	0	0	0	1	0	0	0	A=A+Lit
SUB	A,B	000111	1	0	0	0	0	0	0	1	A=A-B
	B,A	001000	0	1	0	0	0	0	0	1	B=A-B
	A,Lit	001001	1	0	0	0	1	0	0	1	A=A-Lit
AND	A,B	001010	1	0	0	0	0	0	1	0	A=A and B
	B,A	001011	0	1	0	0	0	0	1	0	B=A and B
	A,Lit	001100	1	0	0	0	1	0	1	0	A=A and Lit
OR	A,B	001101	1	0	0	0	0	0	1	1	A=A or B
	B,A	001110	0	1	0	0	0	0	1	1	B=A or B
	A,Lit	001111	1	0	0	0	1	0	1	1	A=A or Lit
NOT	A,A	010000	1	0	0	0	0	1	0	0	A=notA
	B,A	010001	0	1	0	0	0	1	0	0	B=notA
	A,Lit	010010	1	0	0	0	1	1	0	0	A=notLit
XOR	A,A	010011	1	0	0	0	0	1	0	1	A=A xor B
	B,A	010100	0	1	0	0	0	1	0	1	B=A xor B
	A,Lit	010101	1	0	0	0	1	1	0	1	A=A xor Lit
SHL	A,A	010110	1	0	0	0	0	1	1	0	A=shift left A
	B,A	010111	0	1	0	0	0	1	1	0	B=shift left A
	A,Lit	011000	1	0	0	0	1	1	1	0	A=shift left Lit
SHR	A,A	011001	1	0	0	0	0	1	1	1	A=shift right A
	B,A	011010	0	1	0	0	0	1	1	1	B=shift right A
	A,Lit	011011	1	0	0	0	1	1	1	1	A=shift right Lit

Ocupando el **assembly** recién descrito, escriba un programa que realice las siguientes operaciones:

- Cargar los valores 5 y 6 en los registros **A** y **B** respectivamente
- Guardar en **A** la suma de los valores almacenados en **A** y **B**
- Guardar en **A** la resta de los valores almacenados en **A** y **B**
- Setear en 0 el valor de **B**

Necesitamos dar soporte para **variables** y **direccionamiento** en el **assembly**

- Separamos el código en 2 segmentos: datos y código
- En el de datos definimos las variables indicando su contenido y un *label* para facilitar el acceso
- En el de código, va lógicamente el código

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

Necesitamos dar soporte para **variables** y **direccionamiento** en el **assembly**

- Para el direccionamiento, usaremos **()**
- **(label)** para el direccionamiento directo

MOV A, (var) significa guardar en **A** el contenido de la dirección de memoria de la variable **var**

- **(B)** para el indirecto

MOV A, (B) significa guardar en **A** el contenido de la dirección de memoria indicada en **B**

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

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación



IIC2343 – Arquitectura de Computadores

Programabilidad

Profesor: Hans Löbel