

Clase 06 - Saltos

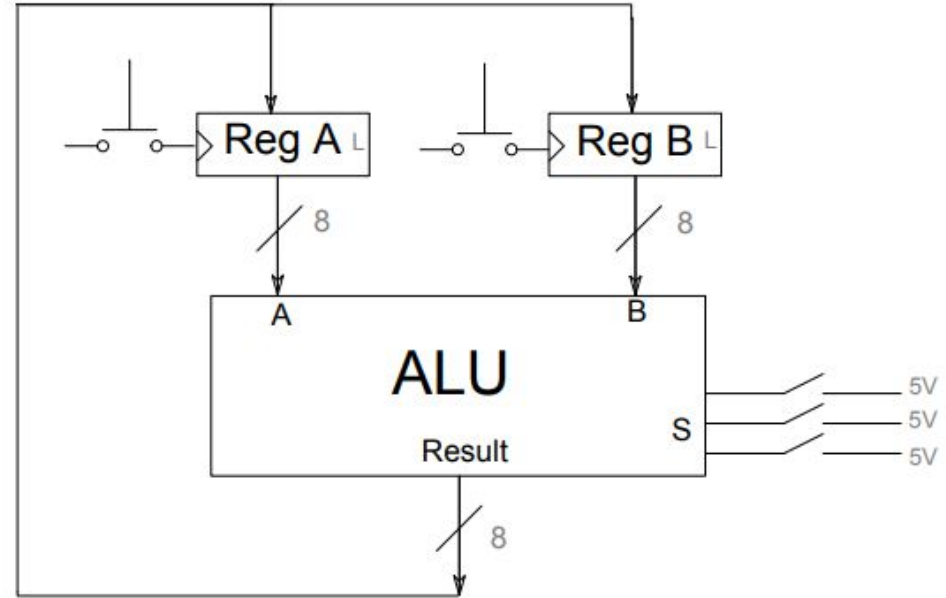


Profesor: **IIC2343 - Arquitectura de Computadores**
- Felipe Valenzuela González
Correo:
frvalenzuela@alumni.uc.cl

Resumen de la clase pasada

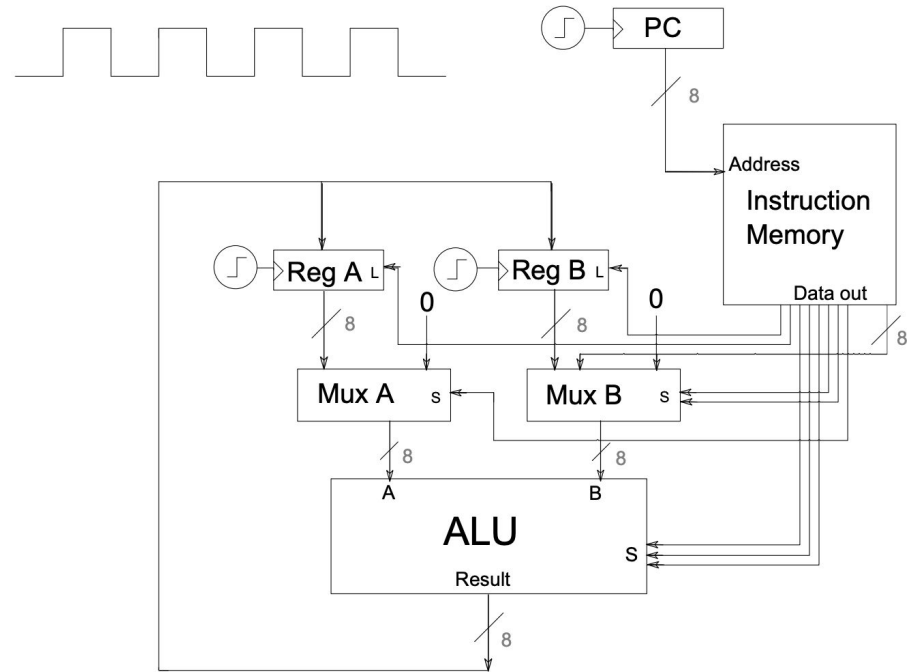
Programabilidad: Calculadora Avanzada

- 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



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



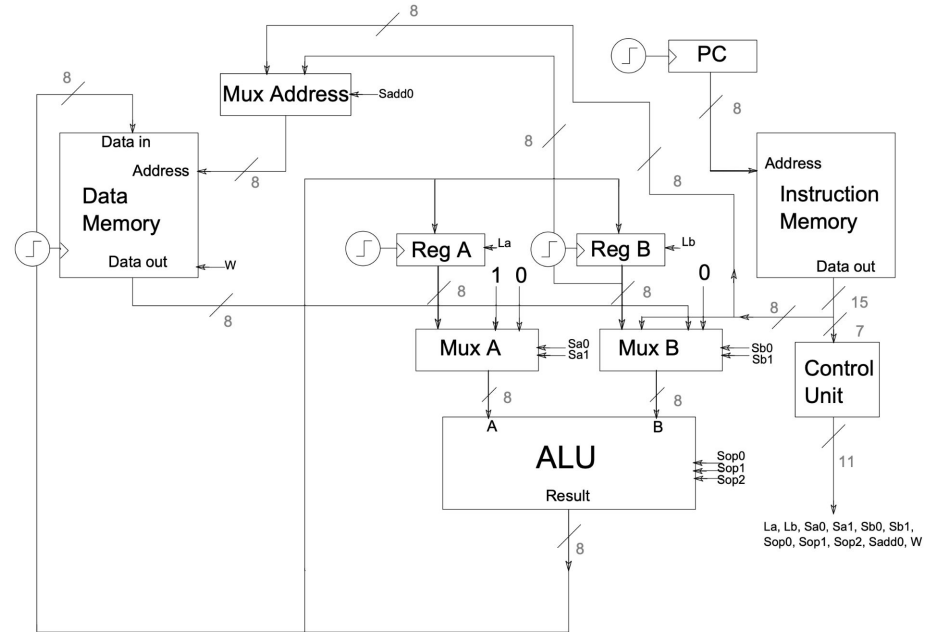
Computador básico: Opcodes

- 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: 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: 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?

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!**

¿Dudas?

Programabilidad: Assembly Ins. Parte 1

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	1	0	1	0	0	0	A=Lit
	B,Lit	000011	0	1	1	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

Programabilidad: Assembly Ins. Parte 2

Instrucción	Operandos	Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
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=not A
	B,A	010001	0	1	0	0	0	1	0	0	B=not A
XOR	A,B	010010	1	0	0	0	0	1	0	1	A=A xor B
	B,A	010011	0	1	0	0	0	1	0	1	B=A xor B
	A,Lit	010100	1	0	0	0	1	1	0	1	A=A xor Lit
SHL	A,A	010101	1	0	0	0	0	1	1	0	A=shift left A
	B,A	010110	0	1	0	0	0	1	1	0	B=shift left A
SHR	A,A	010111	1	0	0	0	0	1	1	1	A=shift right A
	B,A	011000	0	1	0	0	0	1	1	1	B=shift right A

Programabilidad: Assembly Ins. Parte 3

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		-

Programabilidad: Assembly Ins. Parte 4

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

¿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)
- **¿Qué limitante tiene este programa?**

Dirección	Instrucción	Operandos	A	B
0x00	MOV	A,0	0	?
0x01	MOV	B,1	0	1
0x02	ADD	A,B	1	1
0x03	ADD	B,A	1	2
0x04	ADD	A,B	3	2
0x05	ADD	B,A	3	5
0x06	ADD	A,B	8	5
0x07	ADD	B,A	8	13

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

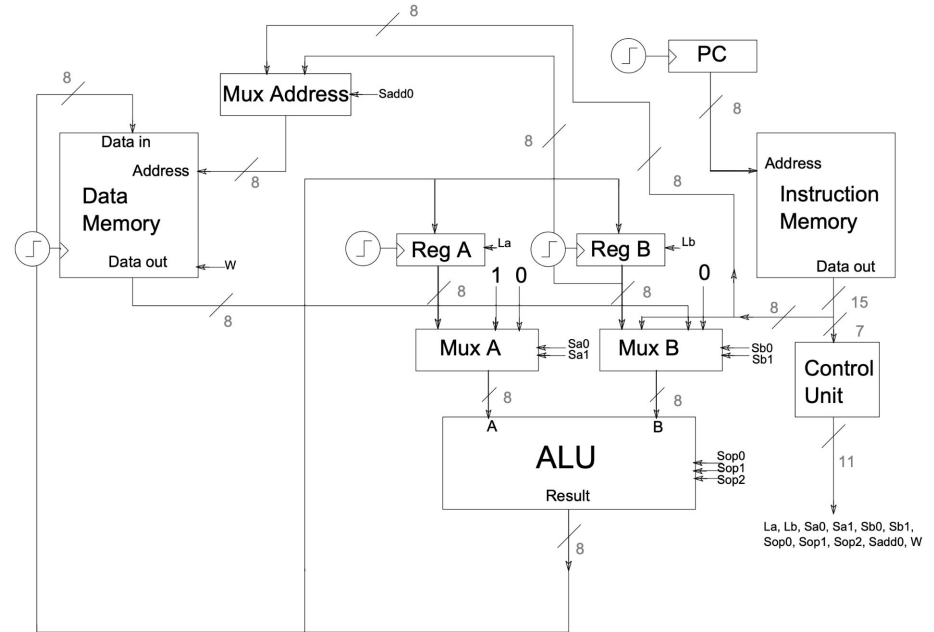
Salto Incondicional: Label

- 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

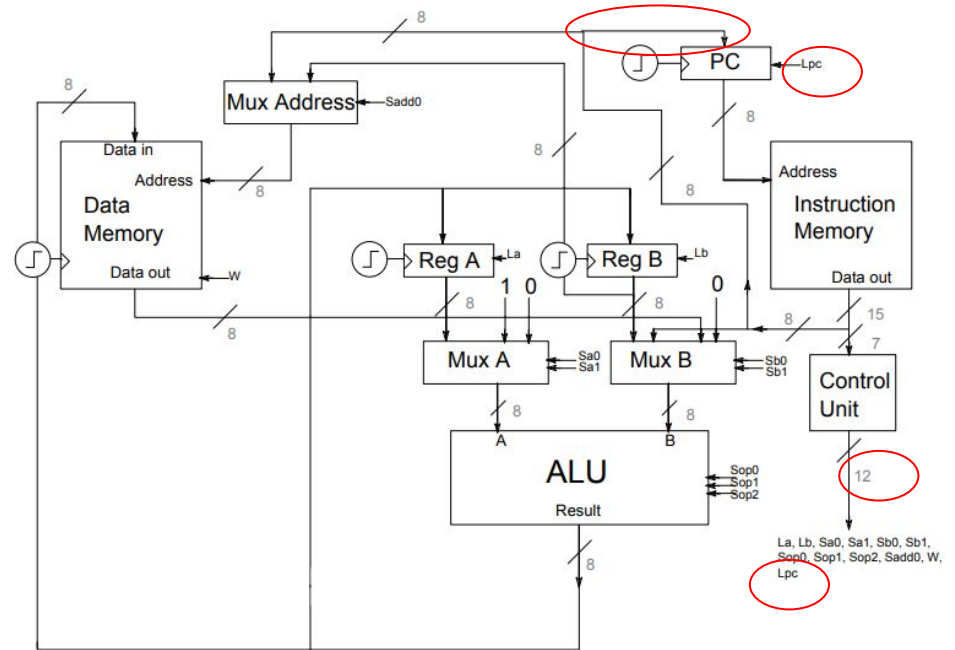
Salto Incondicional: Implementación

- ¿Qué modificación debemos hacer al computador presentado la clase pasada?



Salto Incondicional: Implementación

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



¿Dudas?

Salto condicional: Comparaciones

- El salto incondicional **no es suficiente** para entregar la capacidad de **decisión** al computador
- Es necesario agregar la opción de saltar **condicionalmente**

```
if a == b:  
    print("something")  
else:  
    print("nothing")
```

Salto condicional: Comparaciones aritméticas

- En el curso trabajaremos con las siguientes tipos de comparaciones:

a == b,

a != b,

a > b,

a < b,

a ≥ b,

a ≤ b

Salto condicional: Comparaciones aritméticas

- Por propiedades de las desigualdades podemos escribirlas en base a **resta**.
- Agregaremos la instrucción **CMP**, que restará **sin alterar valor de registros**
- Haremos uso de flags de la ALU (**recordar laboratorio de la actividad 02**)

$$a == b, \Rightarrow a - b == 0,$$

$$a != b, \Rightarrow a - b != 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b, \Rightarrow a - b \leq 0$$

Salto condicional: Jump Equal (JEQ)

- Cuando el resultado de la última operación fue cero
- La flag Z es igual a uno (Z=1)

$$a == b, \Rightarrow a - b == 0,$$

$$a \neq b, \Rightarrow a - b \neq 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b, \Rightarrow a - b \leq 0$$

Salto condicional: Jump Not Equal (JNE)

- Cuando el resultado de la última operación fue distinto de cero
- La flag Z es igual a cero (Z=0)

$$a == b, \Rightarrow a - b == 0,$$

$$a \neq b, \Rightarrow a - b \neq 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b, \Rightarrow a - b \leq 0$$

Salto condicional: Jump Greater Than (JGT)

- Cuando el resultado de la última operación **no** fue cero **ni** negativo
- La flag Z es igual a cero (Z=0)
- La flag N es igual a cero (N=0)

$$a == b, \Rightarrow a - b == 0,$$

$$a != b, \Rightarrow a - b != 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b, \Rightarrow a - b \leq 0$$

Salto condicional: Jump Less Than (JLT)

- Cuando el resultado de la última operación fue negativa
- La flag N es igual a uno (N=1)

$$a == b, \Rightarrow a - b == 0,$$

$$a != b, \Rightarrow a - b != 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b, \Rightarrow a - b \leq 0$$

Salto condicional: Jump Greater or Equal Than (JGE)

- Cuando el resultado de la última operación **no** fue negativa
- La flag N es igual a cero (N=0)

$$a == b, \Rightarrow a - b == 0,$$

$$a != b, \Rightarrow a - b != 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b \Rightarrow a - b \leq 0$$

Salto condicional: Jump Less or Equal Than (JLE)

- Cuando el resultado de la última operación fue cero y negativa
- La flag Z es igual a uno (Z=1)
- La flag N es igual a uno (N=1)

$$a == b, \Rightarrow a - b == 0,$$

$$a != b, \Rightarrow a - b != 0,$$

$$a > b, \Rightarrow a - b > 0,$$

$$a < b, \Rightarrow a - b < 0,$$

$$a \geq b, \Rightarrow a - b \geq 0,$$

$$a \leq b \Rightarrow a - b \leq 0$$

¿Dudas?

Salto condicional: Excepciones

- Al programar nos interesa saber cuando ocurre algo inesperado
- Dichos casos en muchos lenguajes lo llamamos como excepciones

```
try :  
    a - b  
except:  
    print("bad")
```


Salto condicional: Excepciones

- En el curso trabajaremos con las siguientes tipos de saltos por excepción:
- **Salto por Carry:** Se activará cuando se tenga la señal de carry activa (C=1)
- **Salto por Overflow:** Se activará cuando se tenga la señal de overflow activa (V=1)

Instrucción	Operandos	Operación	Condiciones
JCR	Dir	PC = Dir	C=1
JOV	Dir	PC = Dir	V=1

Salto condicional: Salto por Carry (ejemplo)

- El bit de carry puede ser simplemente la salida «*carry out*» del sumador restador
- Este flag fue visto también en el laboratorio de la actividad 02

Dirección	Label	Instrucción	Operandos
0x00	start:	MOV	A,0
0x01		MOV	B,1
0x02	acum:	ADD	A,B
0x03		JC	exc
0x04		JMP	acum
0x05	exc:	JMP	start

Salto condicional: Salto por Overflow

- El bit de overflow puede ser por distintos motivos
- Se definirá por los dados en las operaciones de suma y de resta en la tabla

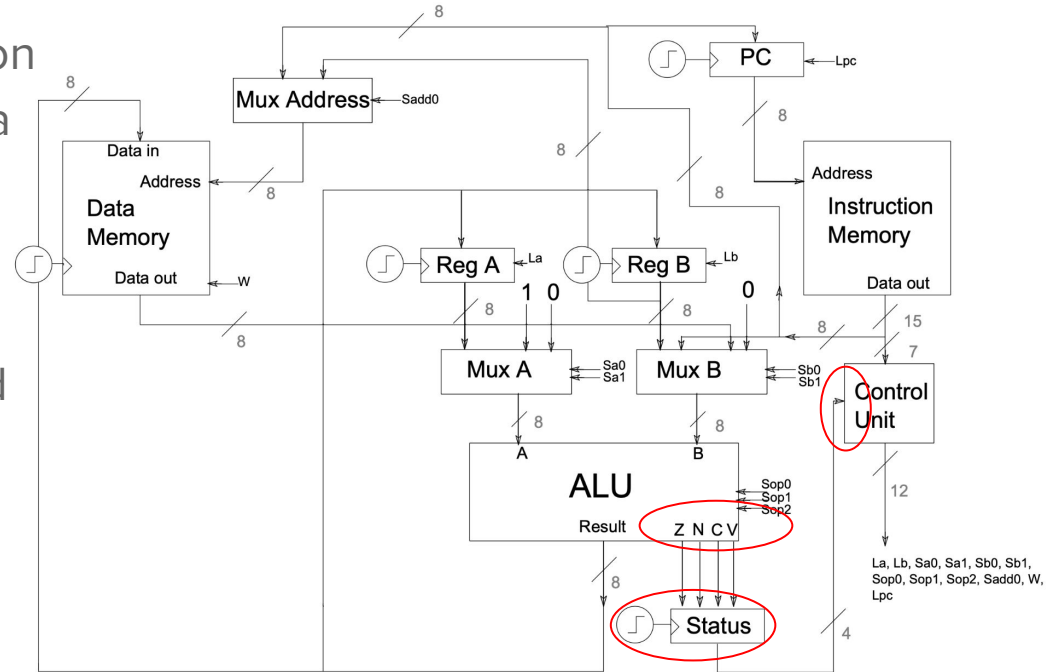
Operación	A	B	Resultado	Ejemplo (1 byte)
$A + B$	≥ 0	≥ 0	< 0	$127 + 4 = -125$
$A + B$	< 0	< 0	≥ 0	$-127 + -4 = 125$
$A - B$	≥ 0	< 0	< 0	$127 - -4 = -125$
$A - B$	< 0	≥ 0	≥ 0	$-127 - 4 = 125$

Salto condicional: Salto por Overflow (ejm)

Dirección	Label	Instrucción	Operandos
0x00	start:	MOV	A,0
0x01		MOV	B,1
0x02	acum:	ADD	A,B
0x03		JOV	exc
0x04		JMP	acum
0x05	exc:	JMP	start

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



¿Dudas?

Salto Resumen

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CMP	A,B A,Lit	A-B 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

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 ✓
- Ahora podemos volver a revisar un que es un programa...

Introducción: ¿Qué es un programa?

```
def encontrar_maximo(arreglo):  
    largo_maximo = len(arreglo)  
  
    maximo = arreglo[0]  
    i = 1  
  
    while i < largo_maximo:  
        if arreglo[i] > maximo:  
            maximo = arreglo[i]  
        i += 1  
  
    return maximo
```

Introducción: ¿Qué es un programa?

```
def encontrar_maximo(arreglo):
```

```
    largo_maximo = len(arreglo)
```

```
    maximo = arreglo[0]
```

```
    i = 1
```

```
    while i < largo_maximo:
```

```
        if arreglo[i] > maximo:
```

```
            maximo = arreglo[i]
```

```
        i += 1
```

```
    return maximo
```



**NOS
FALTA!!**



**A new foe
has appeared!**







SUBROUTINAS

CHALLENGER APPROACHING

Bibliografía

- Apuntes históricos. Hans Löbel, Alejandro Echeverría

06 - Saltos y Subrutinas

⋮	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	✓	⋮
⋮	 06 - Saltos y Subrutinas.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 this directory, listing 15 commits with their names, messages, and dates.

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

Clase 06 - Saltos

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