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

IIC2343 – Arquitectura de Computadores

Saltos y Subrutinas

Profesor: Hans Löbel

```java
public static void promedio()
{
    int[] arreglo = new int[]{6,4,2,3,5};
    int n = 5;
    int i = 0;
    float promedio = 0;

    while(i < n)
    {
        promedio += arreglo[i];
        i++;
    }
    promedio /= n;
    System.out.println(promedio);
}
```

¿Cómo podríamos implementar un while?

```
while (i > 0)
{
        BLA BLA BLA
        i--;
}
```

```
while:
        CMP A,0
        JLE end
        BLA BLA BLA
        SUB A,1
        JMP while
end:
```

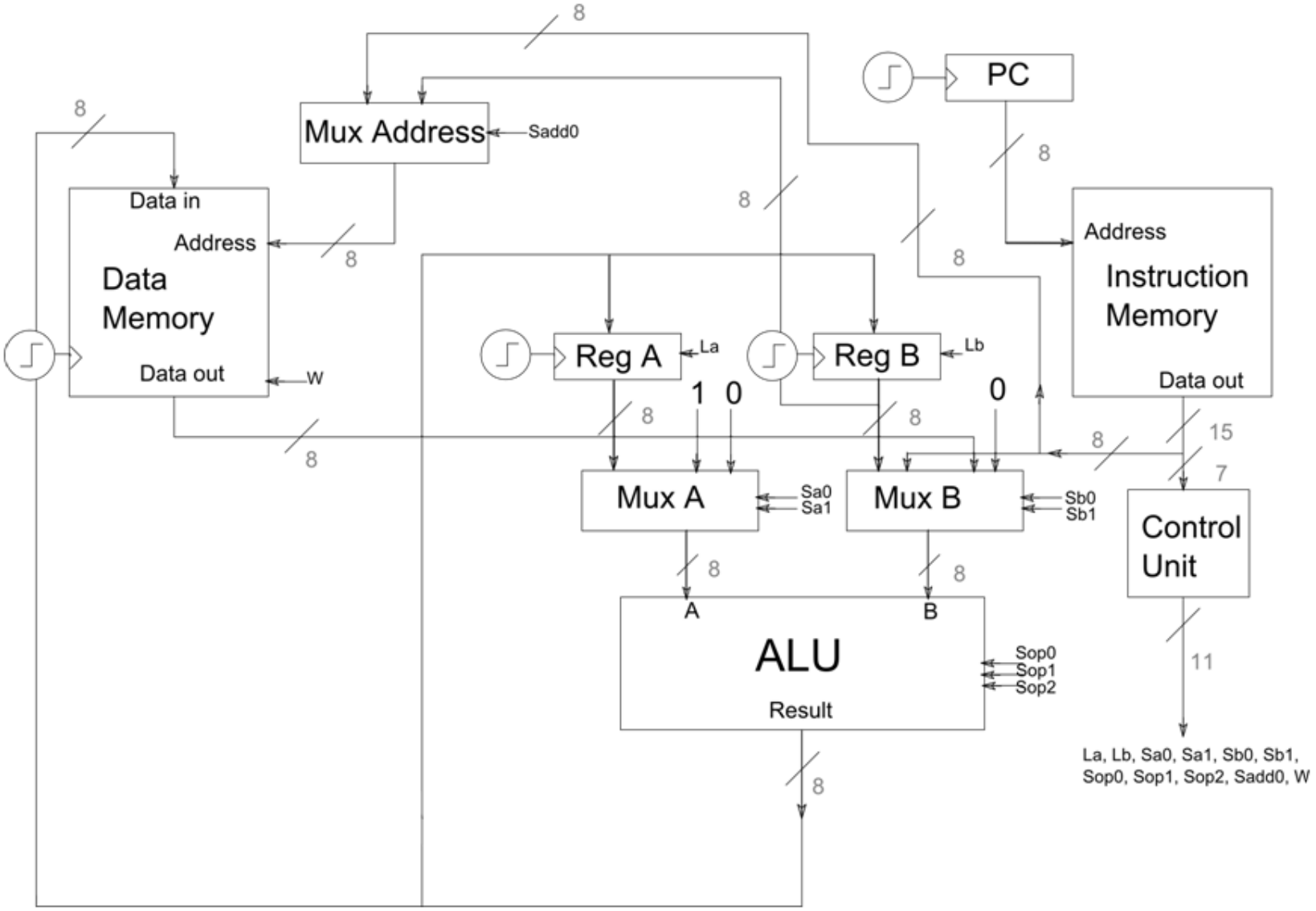
¿Cómo podríamos implementar un if?

```
if (x == 0)
{
        bla bla
}
else
{
        ble ble
}
```

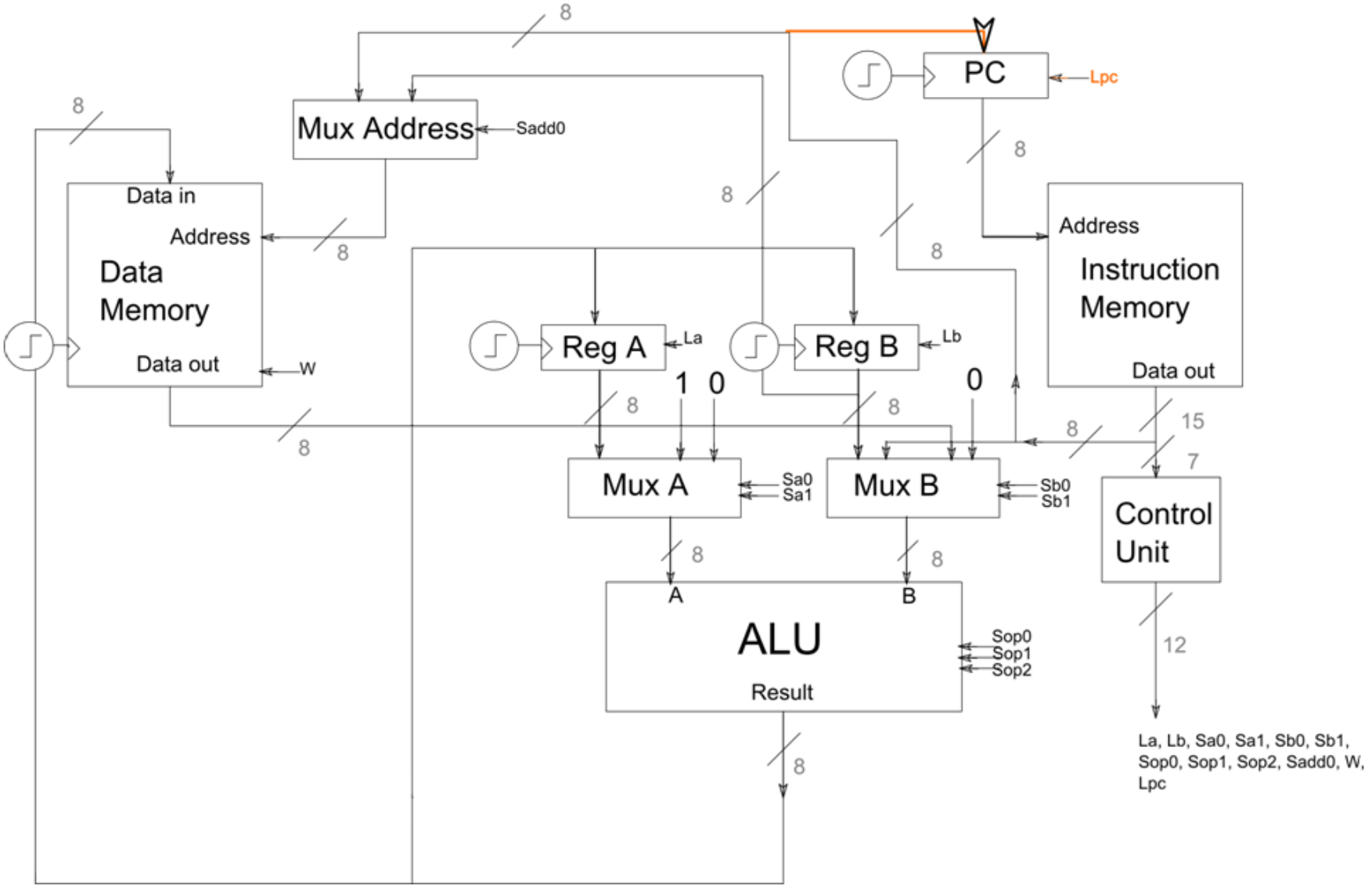
```
CMP A,0
JNE else
        BLA BLA BLA
        JMP end
else:
        BLA BLA BLA
end:
```

Necesitamos saltos incondicionales y condicionales basados en una comparación

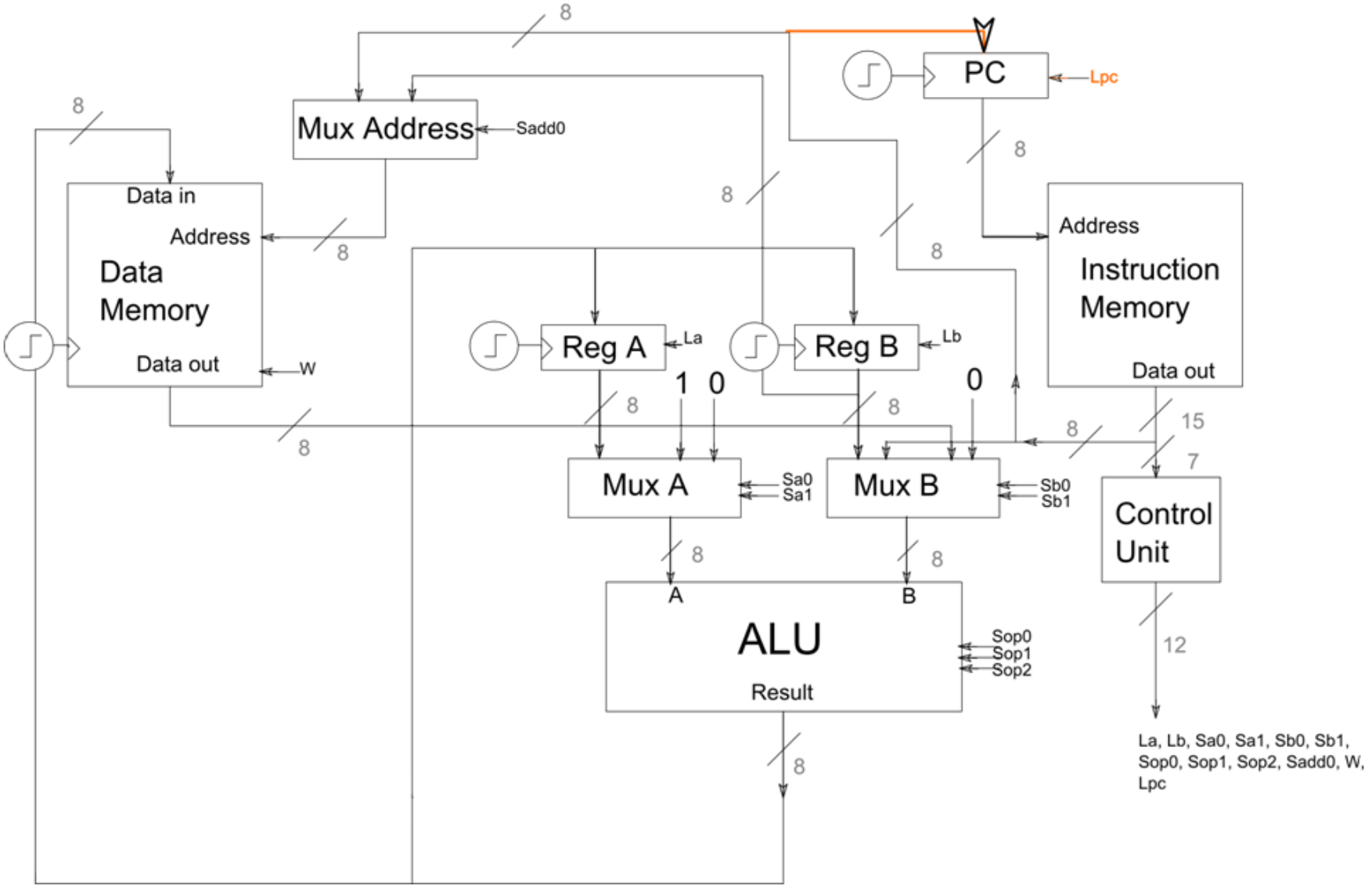
¿Qué debemos agregar para tener soporte en HW para saltos incondicionales?



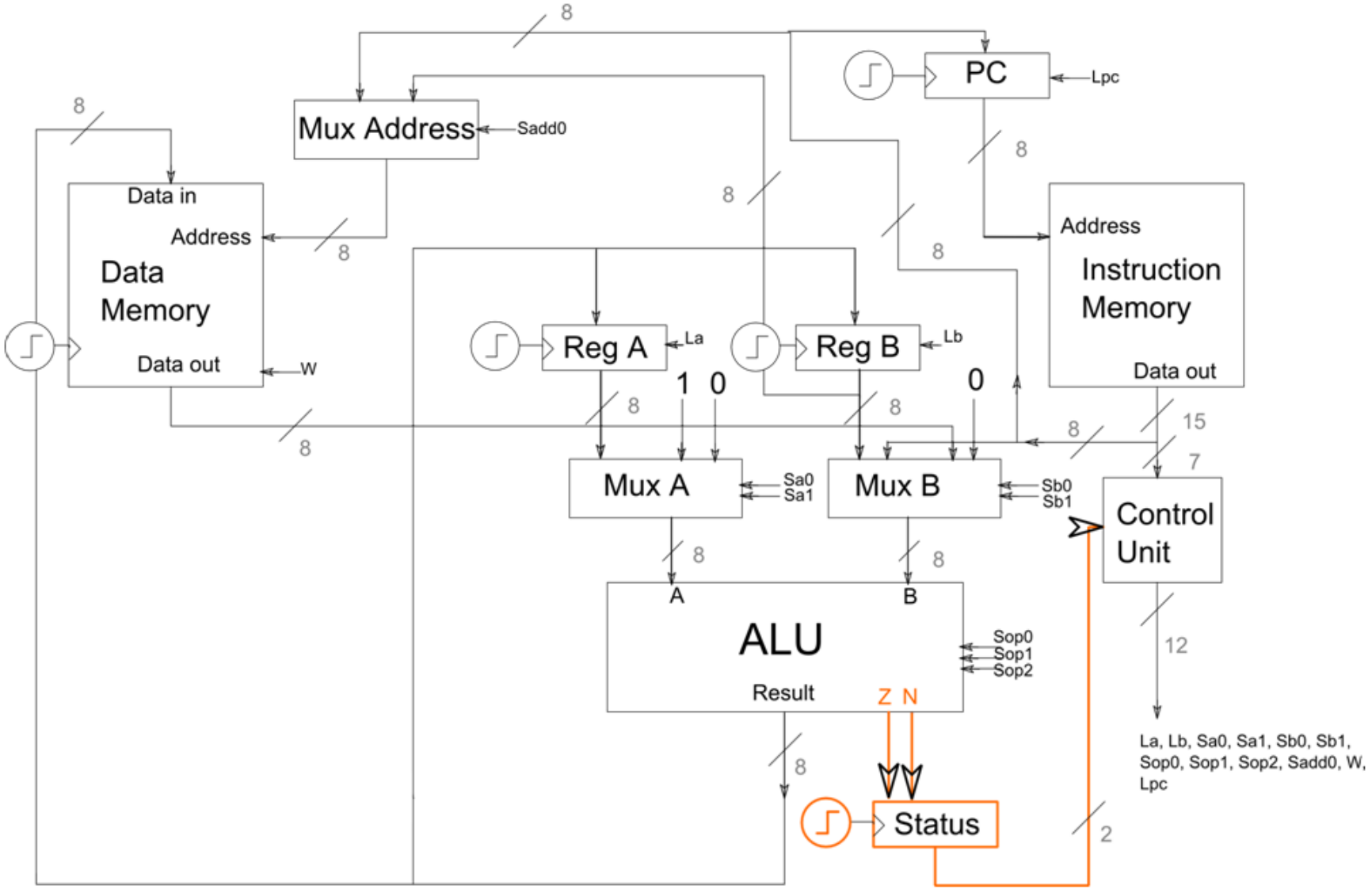
¿Qué debemos agregar para tener soporte en HW para saltos incondicionales?



¿Y para saltos condicionales?

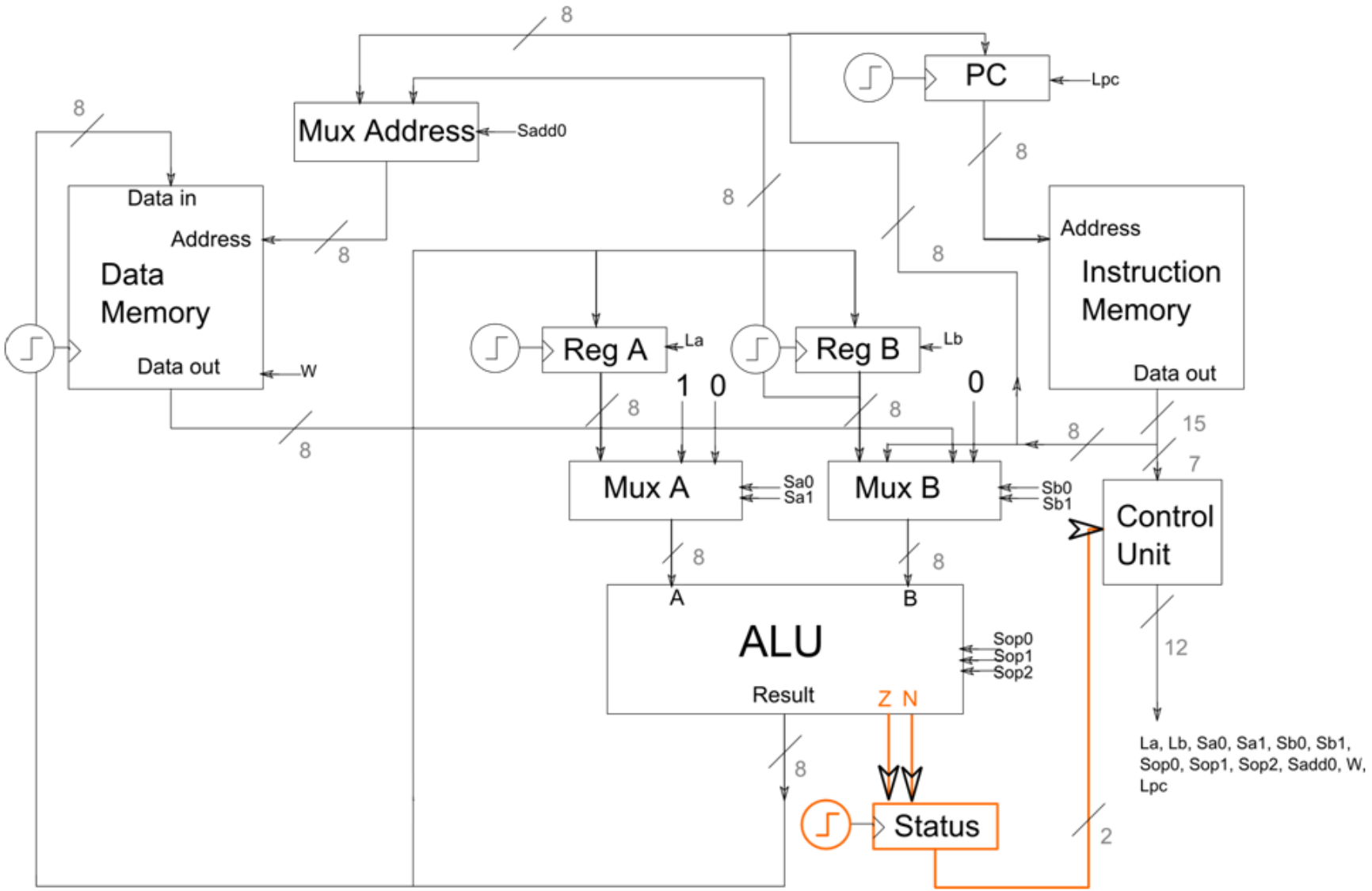


Agregamos el registro Status, que permite saber el estado de la ALU después de la última operación

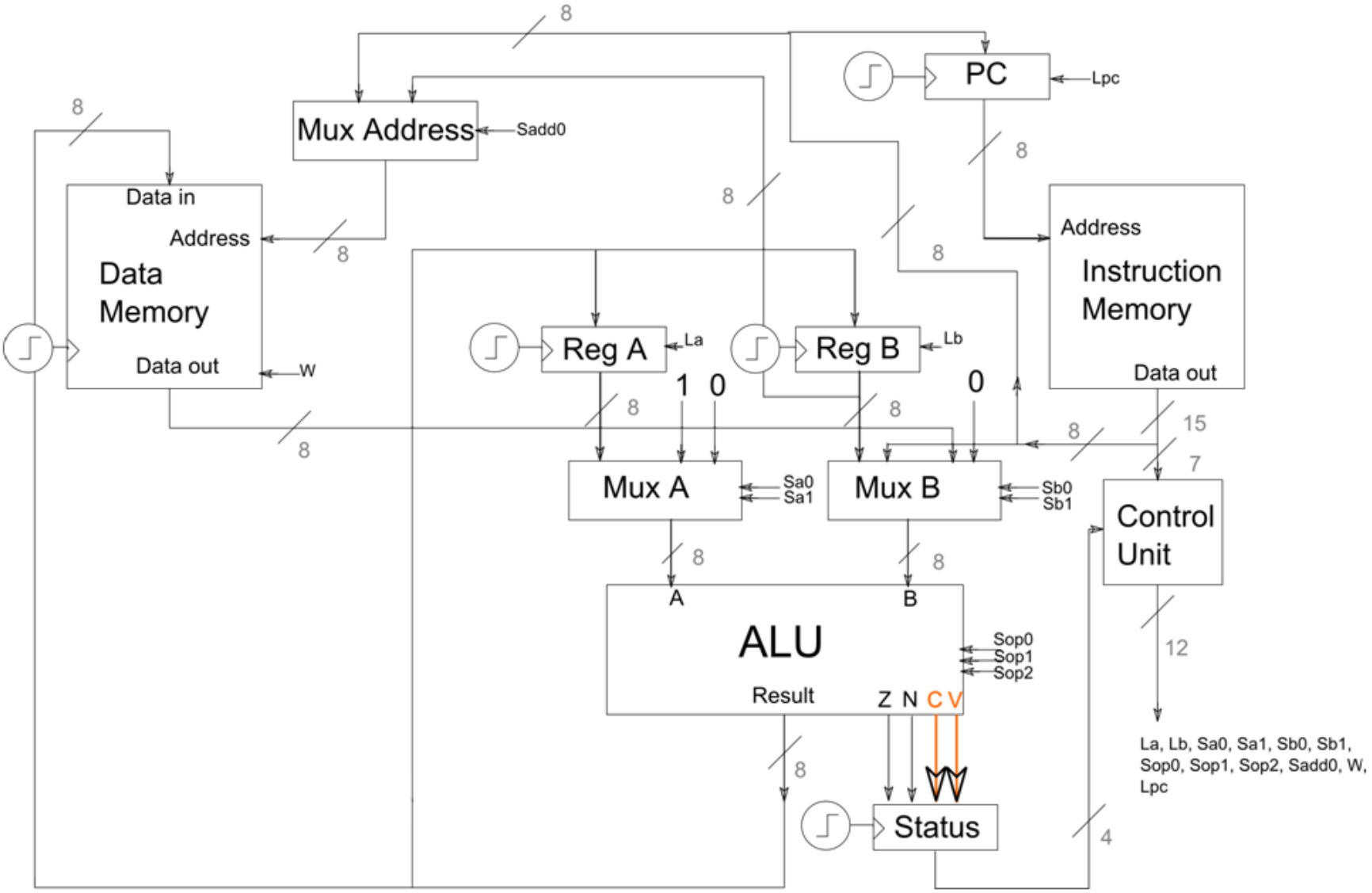


Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CMP	A,B	A-B		
	A,Lit	A-Lit		CMP A,0
JEQ	Dir	PC = Dir	Z=1	JEQ label
JNE	Dir	PC = Dir	Z=0	JNE label
JGT	Dir	PC = Dir	N=0 y Z=0	JGT label
JLT	Dir	PC = Dir	N=1	JLT label
JGE	Dir	PC = Dir	N=0	JGE label
JLE	Dir	PC = Dir	Z=1 o N=1	JLE label

¿Existen más situaciones que nos gustaría controlar?



Agregamos bits para carry y overflow al registro Status



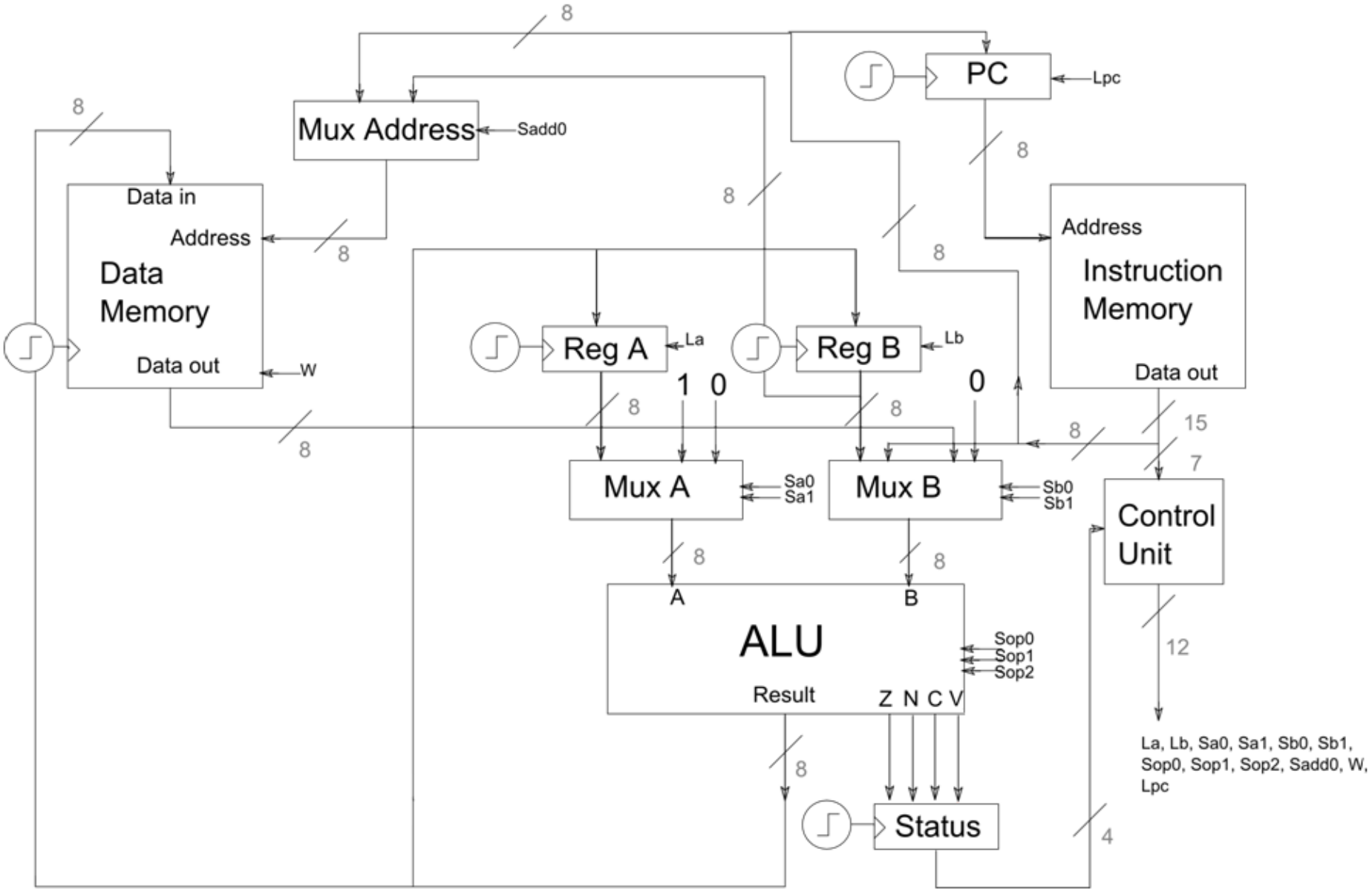
Operación	A	B	Resultado	Ejemplo (1 byte)
$A + B$	≥ 0	≥ 0	< 0	$127 + 4 = -125$
$A + B$	< 0	< 0	≥ 0	$-127 + -4 = 125$
$A - B$	≥ 0	< 0	< 0	$127 - -4 = -125$
$A - B$	< 0	≥ 0	≥ 0	$-127 - 4 = 125$

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
JCR	Dir	PC = Dir	C=1	JCR label
JOV	Dir	PC = Dir	V=1	JOV label

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

- ¿Cómo implementamos el producto punto entre vectores con las instrucciones actuales?
- Un primer paso lógico es implementar la multiplicación
- Pero lamentablemente tenemos que repetir el código de la multiplicación por cada dimensión de los vectores.
- Tenemos el poder expresivo, pero nos falta poder de modularización para poder reutilizar código.

Computador actual

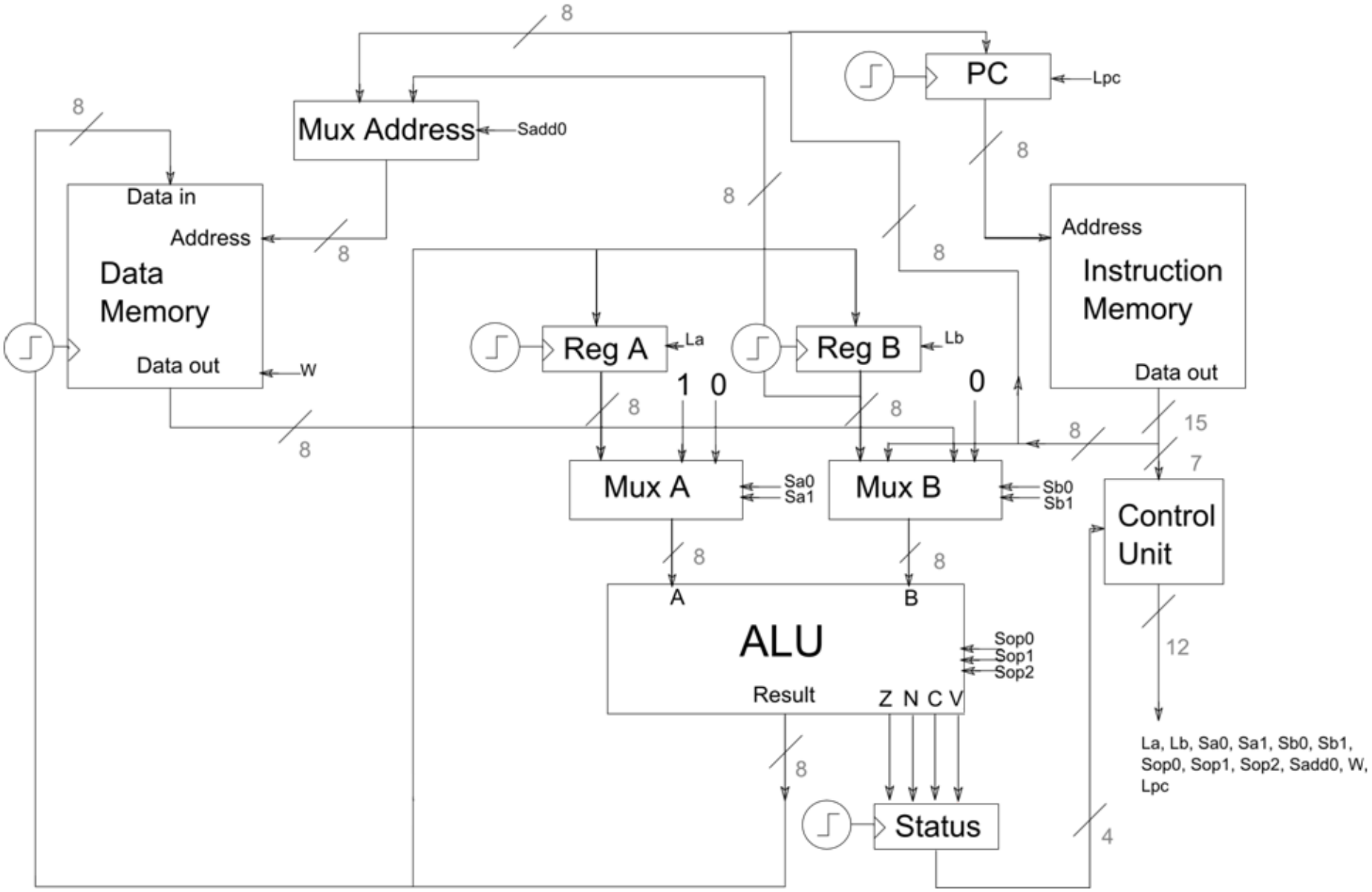


Agreguemos soporte para subrutinas primero desde el punto de vista del assembly

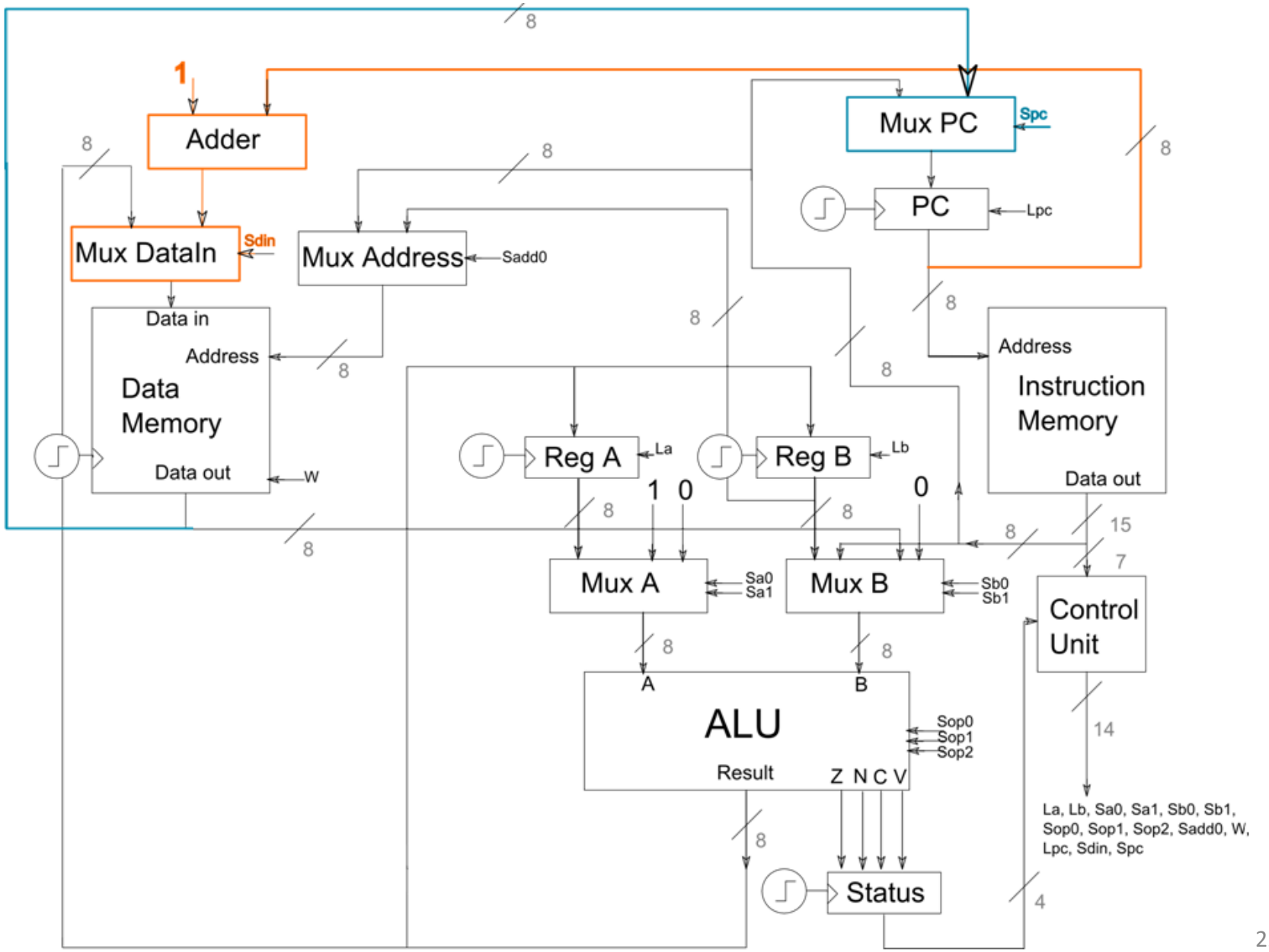
¿Qué elementos son necesarios para implementar subrutinas?

1. Parámetros de entrada
2. Valor de retorno
3. Llamada a la subrutina (salto y retorno)

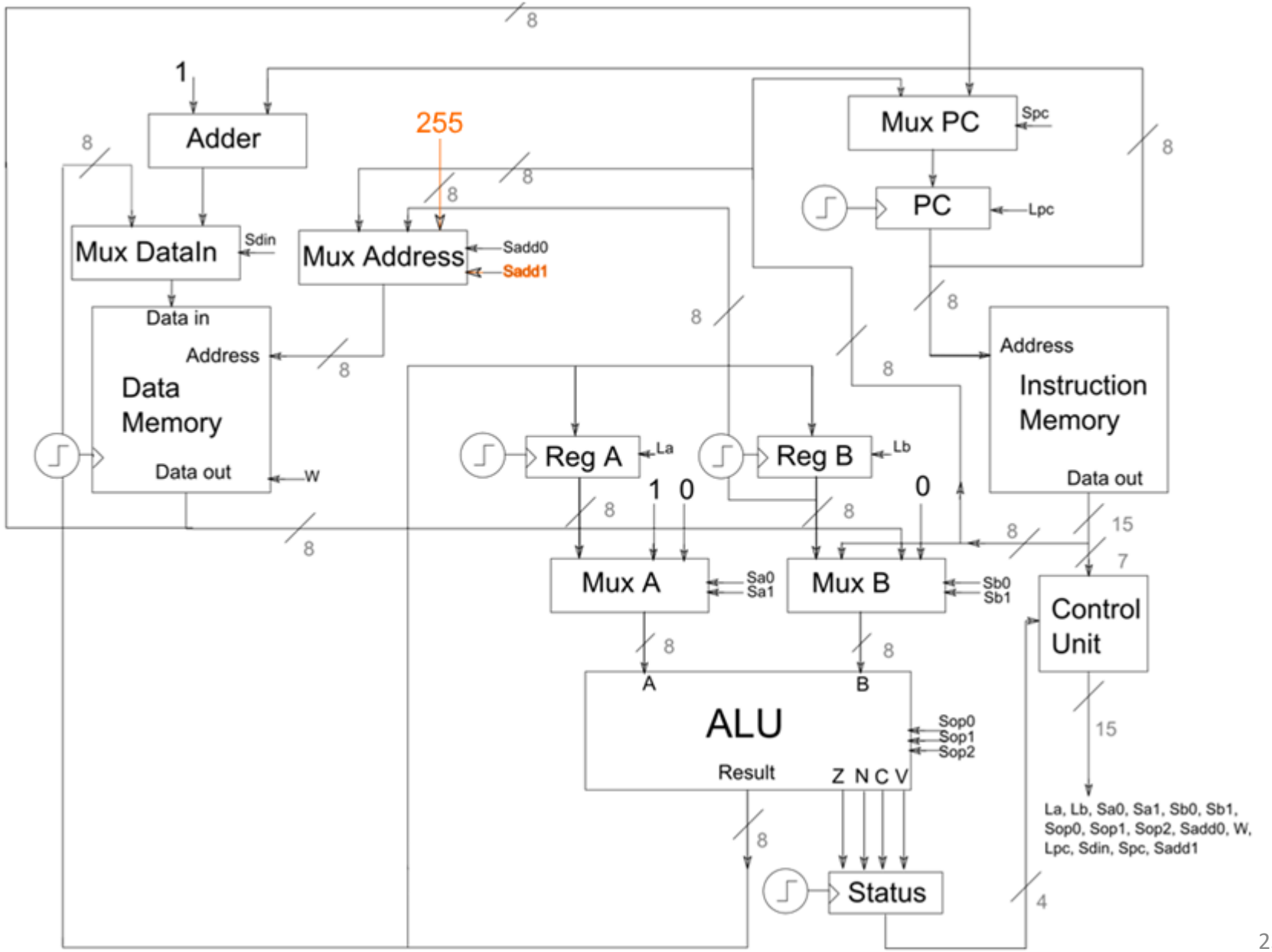
¿Tenemos soporte en HW para almacenar PC?



Agregamos conexión entre PC y memoria



Literal fijo (255) indica donde se almacena PC+1



Agregamos dos nuevas instrucciones al assembly del computador

1. CALL dir: almacena PC+1 en Mem[255] y salta a la dirección dir de la memoria de instrucciones

2. RET: extrae el valor de Mem[255] y lo guarda en PC, lo que conduce a un salto a la dirección siguiente al llamado de la subrutina. Debe ejecutarse siempre al final de esta.

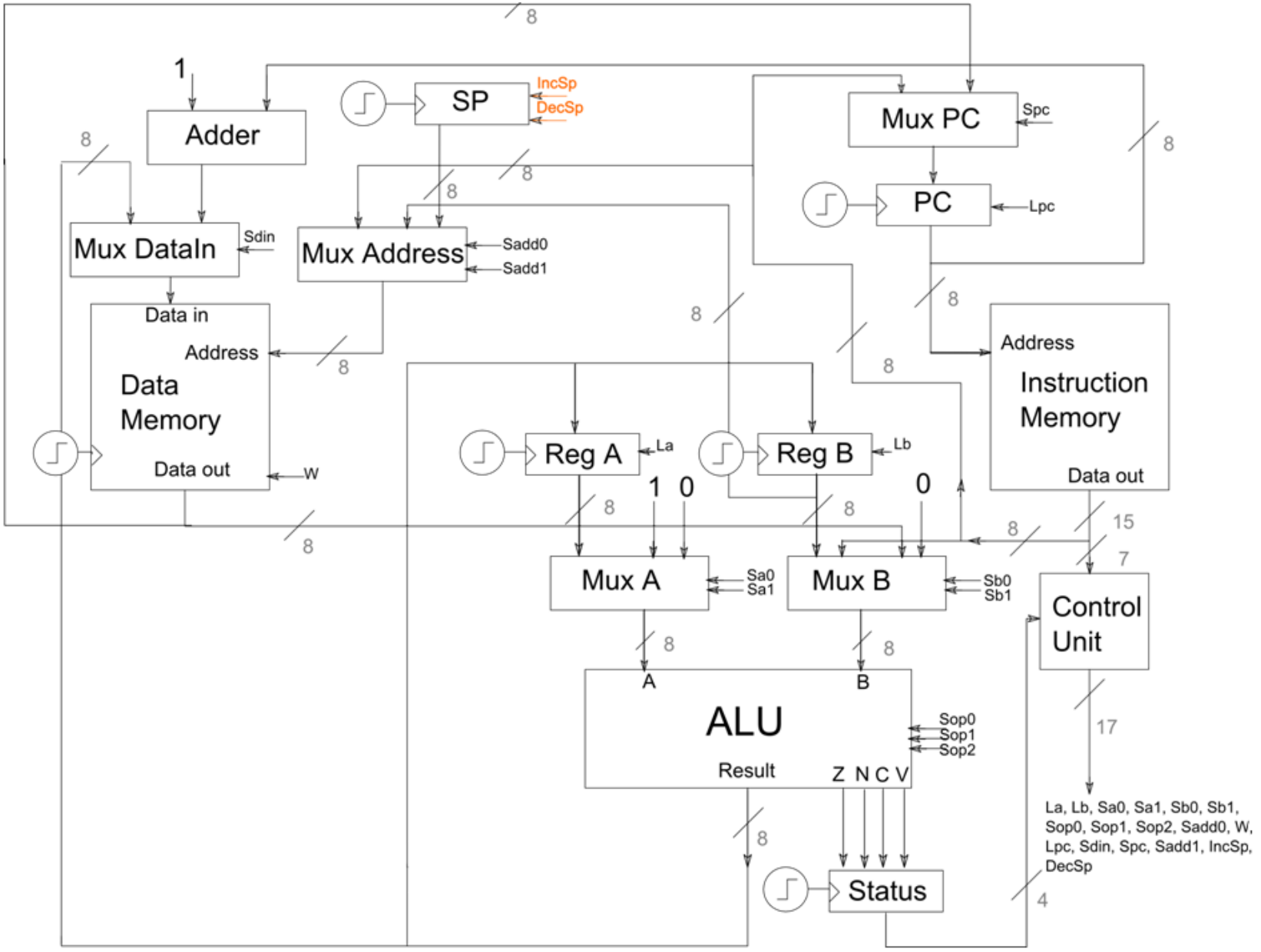
¿Qué pasa en este caso?

```
        JMP main
func1:  MOV A, (var1)
        MOV B, (var2)
        ADD A, B
        MOV (var1), A
        CALL func2
        RET

func2:  MOV A, var1
        MOV B, var2
        ADD A, B
        RET

main:
        MOV A, 5
        MOV B, 2
        MOV (var1), A
        MOV (var2), B
        CALL func1
        …
        …
```

Agregamos incremento y decremento para SP



Resumamos que pasa al llamar y retornar de una subrutina

Al llamar a una subrutina, debemos:

1. Guardar PC+1 en la posición actual de SP
2. Decrementar en 1 SP
3. Guardar la dirección de la subrutina en PC

¿Cuánto ciclos del clock necesitamos para ejecutar estas 3 acciones?

Respuesta: 1

Resumamos que pasa al llamar y retornar de una subrutina

Al retornar de una subrutina, debemos:

1. Incrementar en 1 SP
2. Guardar el valor de memoria apuntado por SP incrementado, en PC

¿Cuánto ciclos del clock necesitamos para ejecutar estas 2 acciones?

Respuesta: 2

¿Qué pasa en este caso con A y B?

```
main:   …
        …
        MOV A, 5
        MOV B, 3
        CALL func
        ADD A, B
        …
        …

func:   INC B
        ADD A, B
        RET
```

Stack de uso general soluciona estos problemas

- Agregamos las instrucciones PUSH y POP
- PUSH Reg almacena en Mem[SP] el valor almacenado en el registro Reg y luego decrementa SP
- POP Reg primero incrementa SP y luego escribe en Reg el valor almacenado actualmente en Mem[SP]

Ahora no tenemos problemas

```
main:   …
        …
        MOV A, 5
        MOV B, 3
        PUSH A
        PUSH B
        CALL func
        POP B
        POP A
        ADD A, B
        …
        …

func:   INC B
        ADD A, B
        RET
```

En POP se invierte el orden de PUSH

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CALL	Dir	Mem[SP] = PC + 1, SP- -, PC = Dir		CALL func
RET		SP++		-
		PC = Mem[SP]		-
PUSH	A	Mem[SP] = A, SP- -		-
PUSH	B	Mem[SP] = B, SP- -		-
POP	A	SP++		-
		A = Mem[SP]		-
POP	B	SP++		-
		B = Mem[SP]		-

Instrucción	Operandos	Opcode	Condition	Lpc	La	Lb	Sa0,1	Sb0,1	Sop0,1,2	Sadd0,1	Sdin0	Spc0	W	IncSp	DecSp
XOR	A,B	0101000		0	1	0	A	B	XOR	-	-	-	0	0	0
	B,A	0101001		0	0	1	A	B	XOR	-	-	-	0	0	0
	A,Lit	0101010		0	1	0	A	LIT	XOR	-	-	-	0	0	0
	A,(Dir)	0101011		0	1	0	A	DOUT	XOR	LIT	-	-	0	0	0
	A,(B)	0101100		0	1	0	A	DOUT	XOR	B	-	-	0	0	0
	(Dir)	0101101		0	0	0	A	B	XOR	LIT	ALU	-	1	0	0
SHL	A,A	0101110		0	1	0	A	-	SHL	-	-	-	0	0	0
	B,A	0101111		0	0	1	A	-	SHL	-	-	-	0	0	0
	(Dir)	0110011		0	0	0	A	B	SHL	LIT	ALU	-	1	0	0
SHR	A,A	0110100		0	1	0	A	-	SHR	-	-	-	0	0	0
	B,A	0110101		0	0	1	A	-	SHR	-	-	-	0	0	0
	(Dir)	0111001		0	0	0	A	B	SHR	LIT	ALU	-	1	0	0
INC	B	0111010		0	0	1	ONE	B	ADD	-	-	-	0	0	0
CMP	A,B	0111011		0	0	0	A	B	SUB	-	-	-	0	0	0
	A,Lit	0111100		0	0	0	A	LIT	SUB	-	-	-	0	0	0
JMP	Dir	0111101		1	0	0	-	-	-	-	-	LIT	0	0	0
JEQ	Dir	0111110	Z=1	1	0	0	-	-	-	-	-	LIT	0	0	0
JNE	Dir	0111111	Z=0	1	0	0	-	-	-	-	-	LIT	0	0	0
JGT	Dir	1000000	N=0 y Z=0	1	0	0	-	-	-	-	-	LIT	0	0	0
JLT	Dir	1000001	N=1	1	0	0	-	-	-	-	-	LIT	0	0	0
JGE	Dir	1000010	N=0	1	0	0	-	-	-	-	-	LIT	0	0	0
JLE	Dir	1000011	N=1 o Z=1	1	0	0	-	-	-	-	-	LIT	0	0	0
JCR	Dir	1000100	C=1	1	0	0	-	-	-	-	-	LIT	0	0	0
JOV	Dir	1000101	V=1	1	0	0	-	-	-	-	-	LIT	0	0	0
CALL	Dir	1000101		1	0	0	-	-	-	SP	PC	LIT	1	0	1
RET		1000110		0	0	0	-	-	-	-	-	-	0	1	0
		1000111		1	0	0	-	-	-	SP	-	DOUT	0	0	0
PUSH	A	1001000		0	0	0	A	ZERO	ADD	SP	ALU	-	1	0	1
PUSH	B	1001001		0	0	0	ZERO	B	ADD	SP	ALU	-	1	0	1
POP	A	1001010		0	1	0	-	-	-	-	-	-	0	1	0
		1001011		0	1	0	ZERO	DOUT	ADD	SP	ALU	-	0	0	0
POP	B	1001100		0	0	1	-	-	-	-	-	-	0	1	0
		1001101		0	0	1	ZERO	DOUT	ADD	SP	ALU	-	0	0	0

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

IIC2343 – Arquitectura de Computadores

Saltos y Subrutinas

Profesor: Hans Löbel