

Ayudantía 4

Tomás Contreras
Susana Figueroa
Andrés Gonzalez
Jorge Schenke
Sebastián Ramos
Rocío Hernández
Joaquin Viñuela

13/09/2021

Jumps

Assembly

Sub-Rutinas

[Recordemos]

Así quedamos con nuestro computador basico.

Hasta ahora podemos ejecutar instrucciones básicas, pero queremos agregarle funcionalidades que son infaltables para cualquier programa moderno.

Jumps

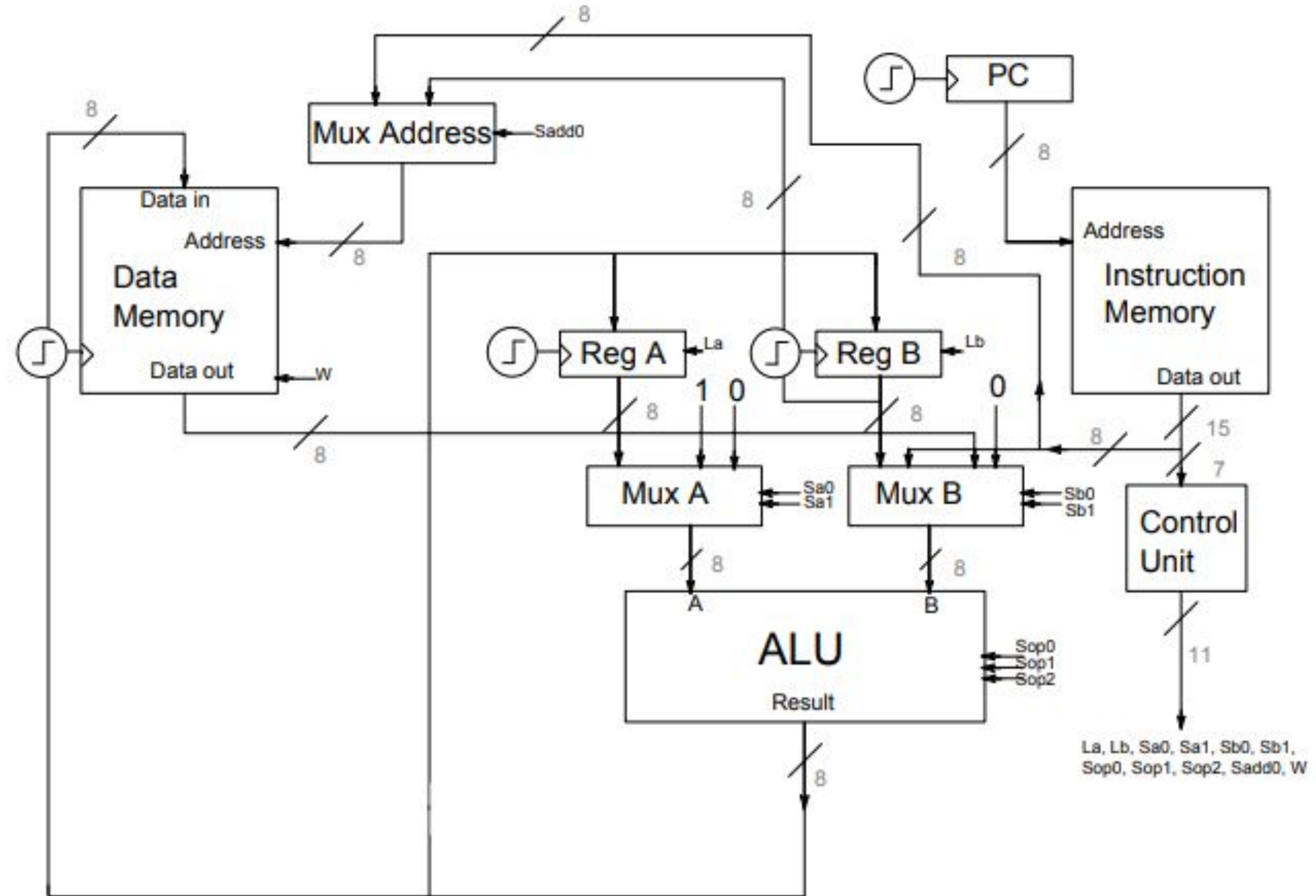


Figure 15: Modos de direccionamiento del computador básico

Computador con salto incondicional

Se agrega:

- Señal Lpc para PC

Jumps

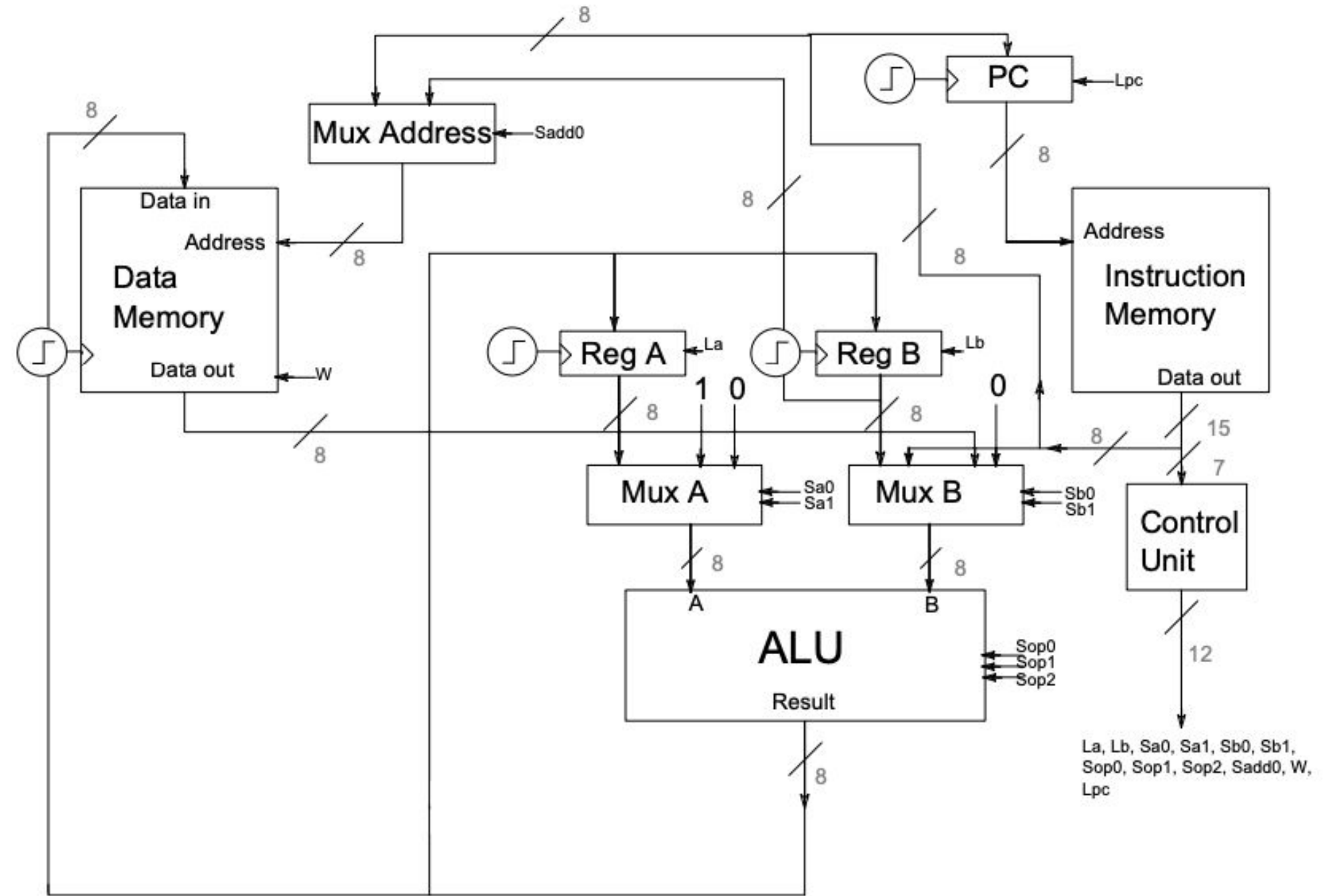


Figure 1: Computador con salto incondicional.

Salto condicional

Se agrega:

- Registro Status

Señales de conexión de la ALU con Status:

- N: Negativo?
- Z: Cero?
- C: Carry?
- V: overflow?

Jumps

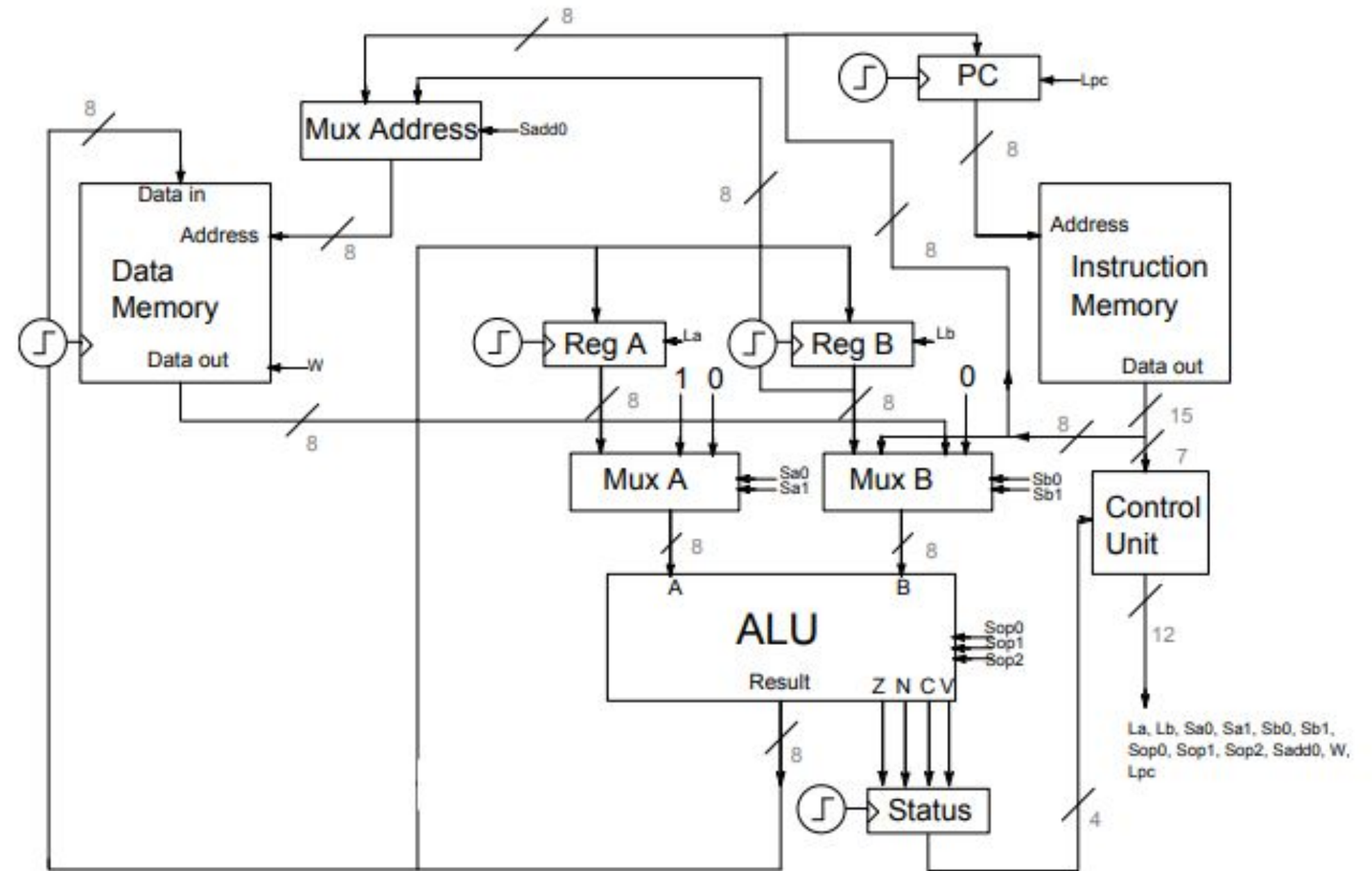


Figure 2: Computador con salto condicional.

Assembly

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]		-
	(Dir)	Mem[Dir]=A+B		ADD (var1)
SUB	A,(Dir)	A=A-Mem[Dir]		SUB A,var1
	A,(B)	A=A-Mem[B]		-
	(Dir)	Mem[Dir]=A-B		SUB (var1)
AND	A,(Dir)	A=A and Mem[Dir]		AND A,(var1)
	A,(B)	A=A and Mem[B]		-
	(Dir)	Mem[Dir]=A and B		-
OR	A,(Dir)	A=A or Mem[Dir]		OR A,(var1)
	A,(B)	A=A or Mem[B]		-
	(Dir)	Mem[Dir]=A or B		OR (var1)
NOT	A,(Dir)	A=not Mem[Dir]		NOT A,(var1)
	A,(B)	A=not Mem[B]		-
	(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	A,(Dir)	A=shift left Mem[Dir]		SHL A,(var1)
	A,(B)	A=shiflt left Mem[B]		-
	(Dir)	Mem[Dir]=shift left A		SHL (var1)
SHR	A,(Dir)	A=shift right Mem[Dir]		SHR A,(var1)
	A,(B)	A=shiflt right Mem[B]		-
	(Dir)	Mem[Dir]=shift right A		SHR(var1)
INC	B	B=B+1		-

Assembly

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
CMP	A,B A,(B) A,Lit A,(Dir)	A-B A-Mem[B] A-Lit A-Mem[Dir]		CMP A,0 CMP A,(label)
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

Assembly

```
1  DATA:
2      var1: 1
3      var2: 2
4
5  CODE:
6      MOV A, (var1)
7      MOV B, var2
8      ADD A, 2
9      ADD B, 3
```

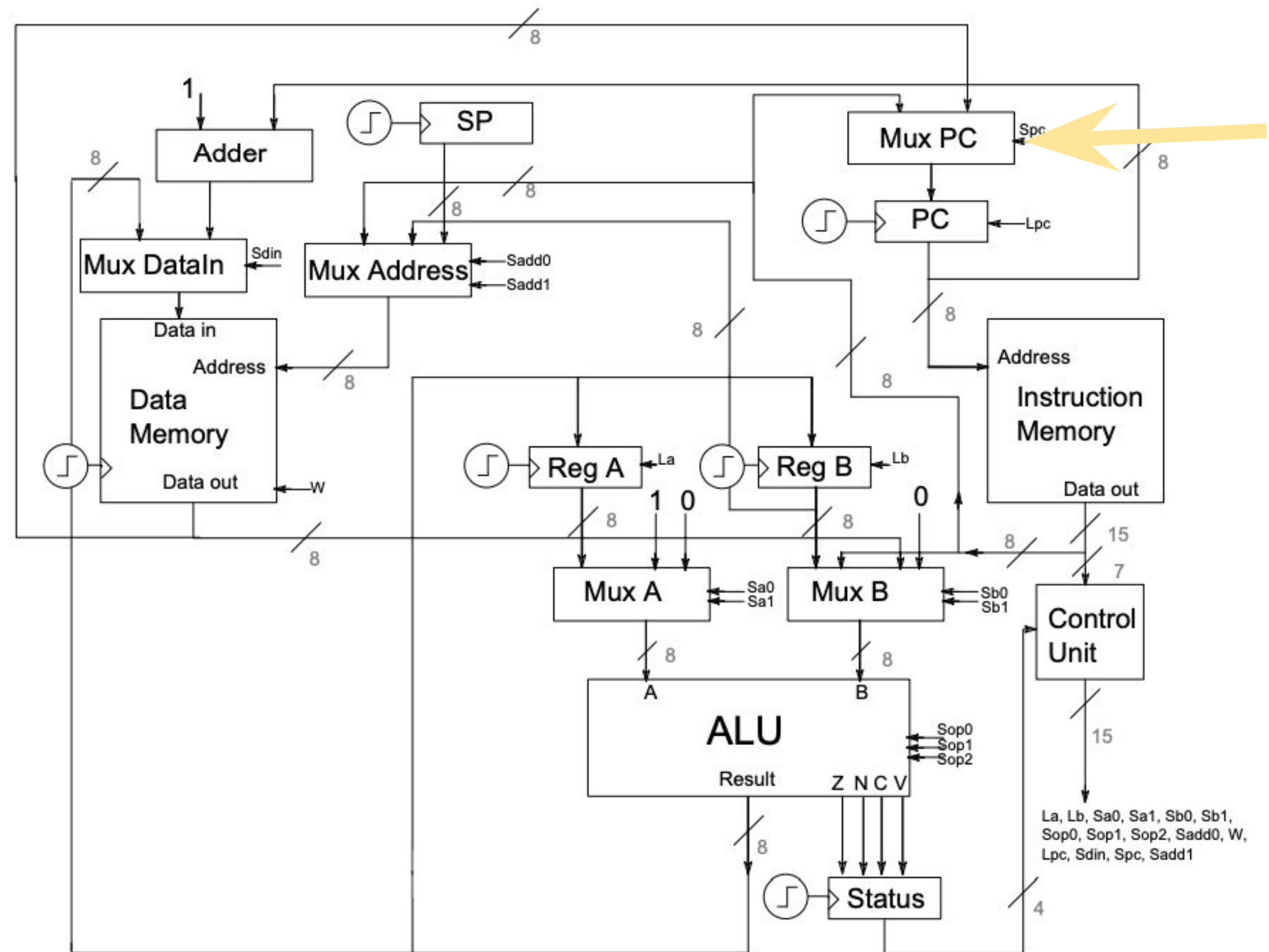

Subrutinas

```
1  DATA:
2      var1: 1
3      var2: 2
4
5  CODE:
6      MOV A, (var1)
7      MOV B, var2
8      ADD A, 2
9      ADD B, 3
10     CALL FUNC1
11     ...
12
13  FUNC1
14     SUB A, 1
15     MOV (B), A
16     RET
```

Subrutinas

Se agrega:

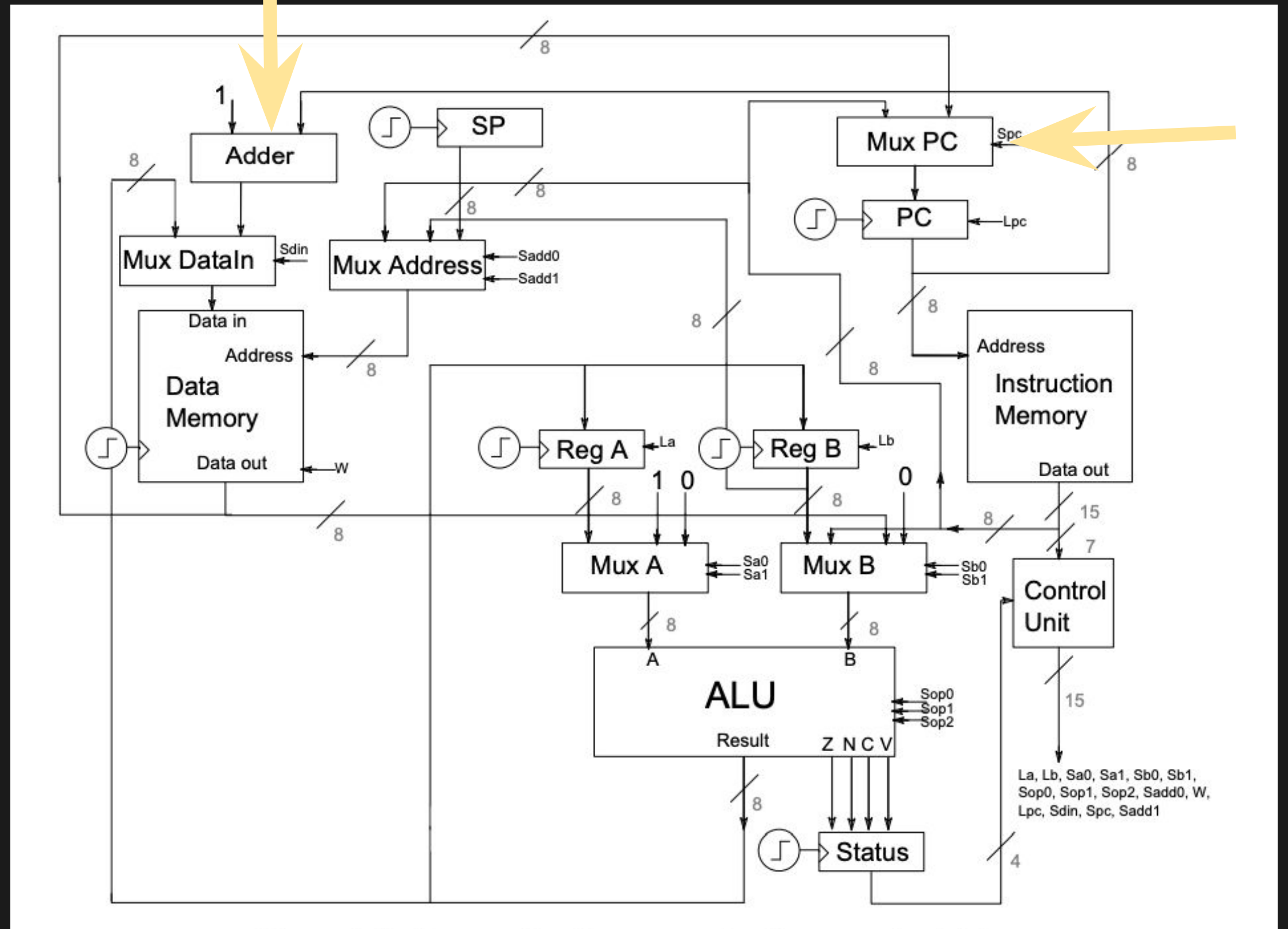
- Mux PC



Subrutinas

Se agrega:

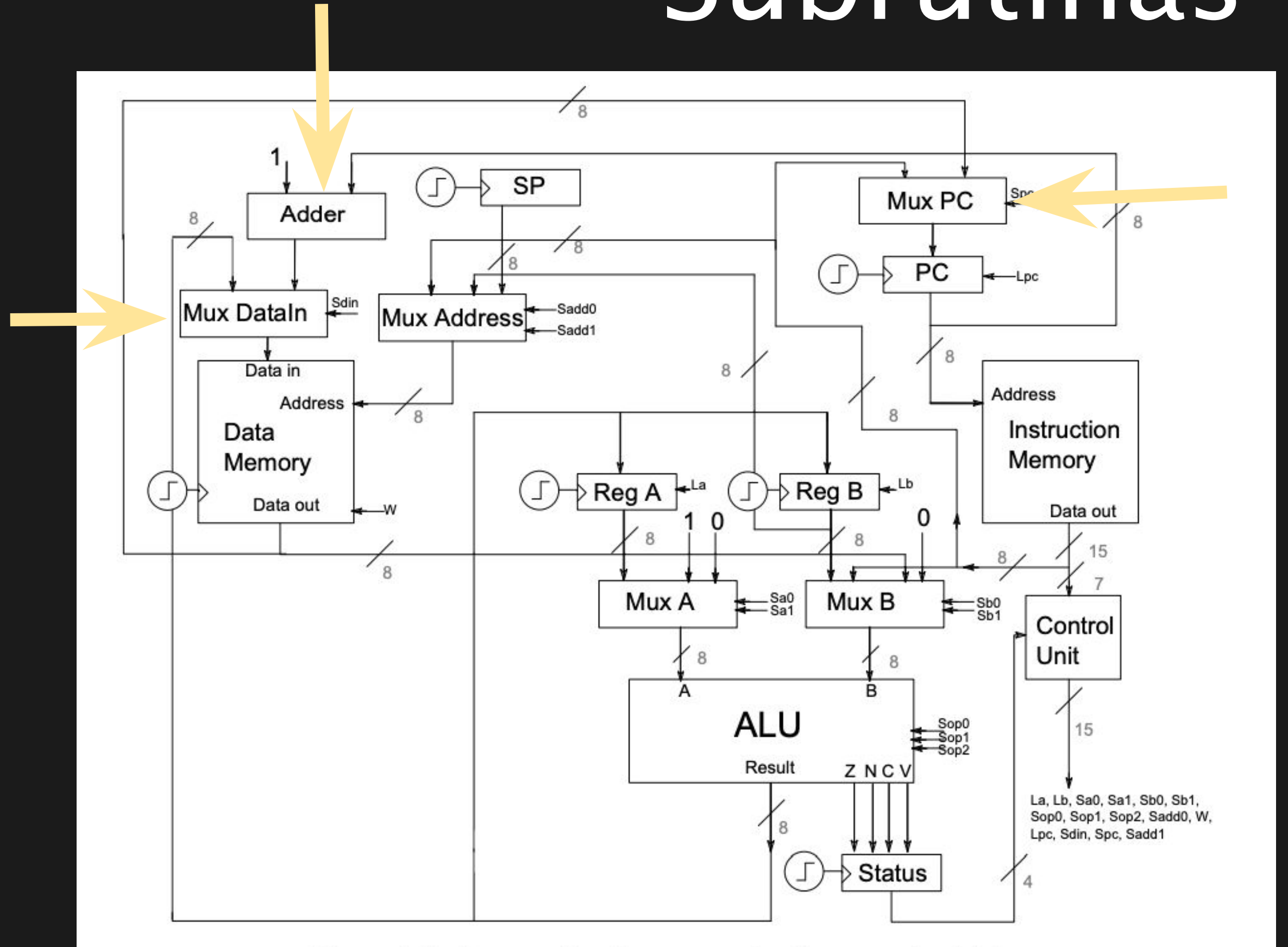
- Mux PC
- Adder



Subrutinas

Se agrega:

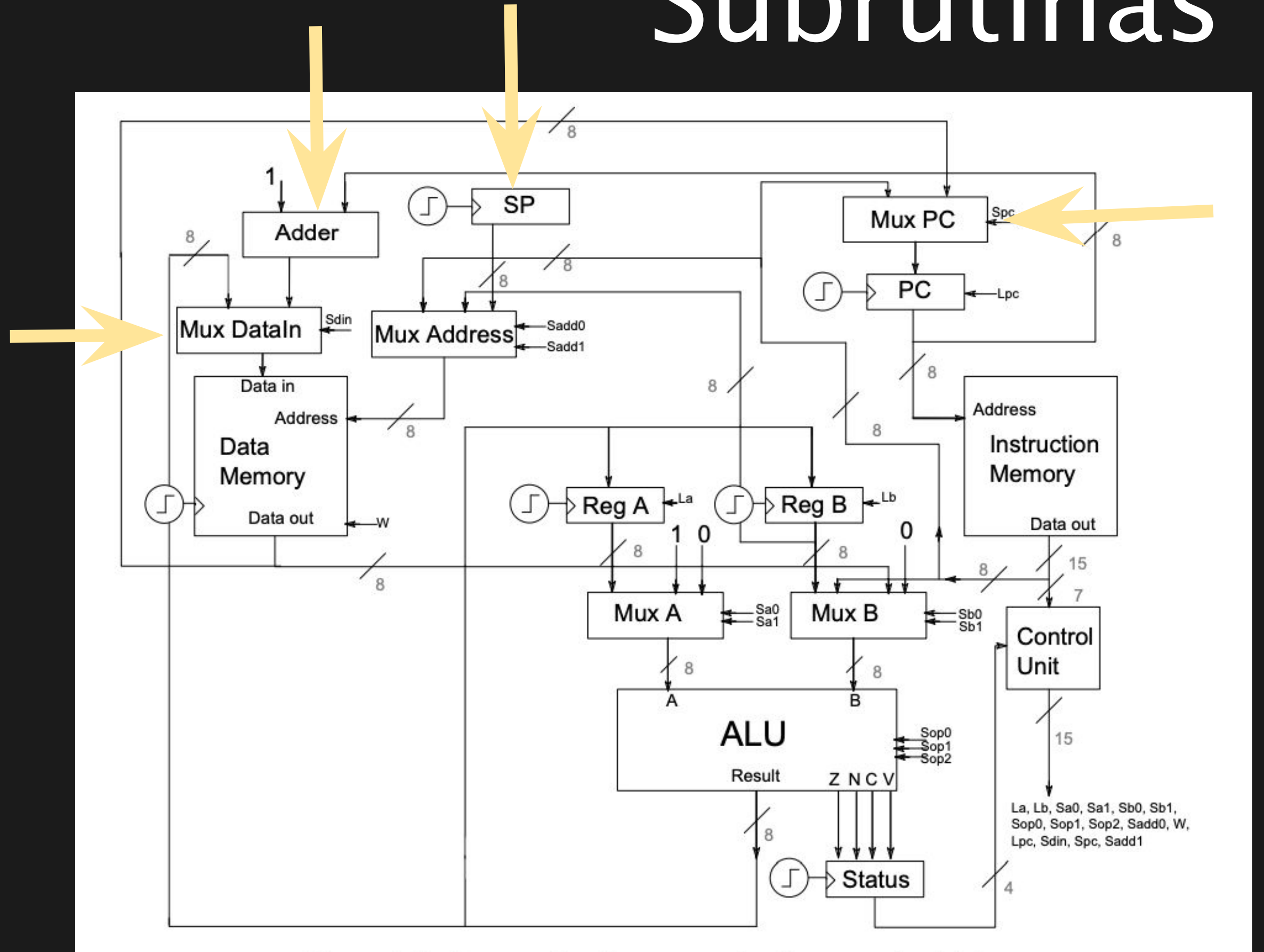
- Mux PC
- Adder
- Mux DataIn



Subrutinas

Se agrega:

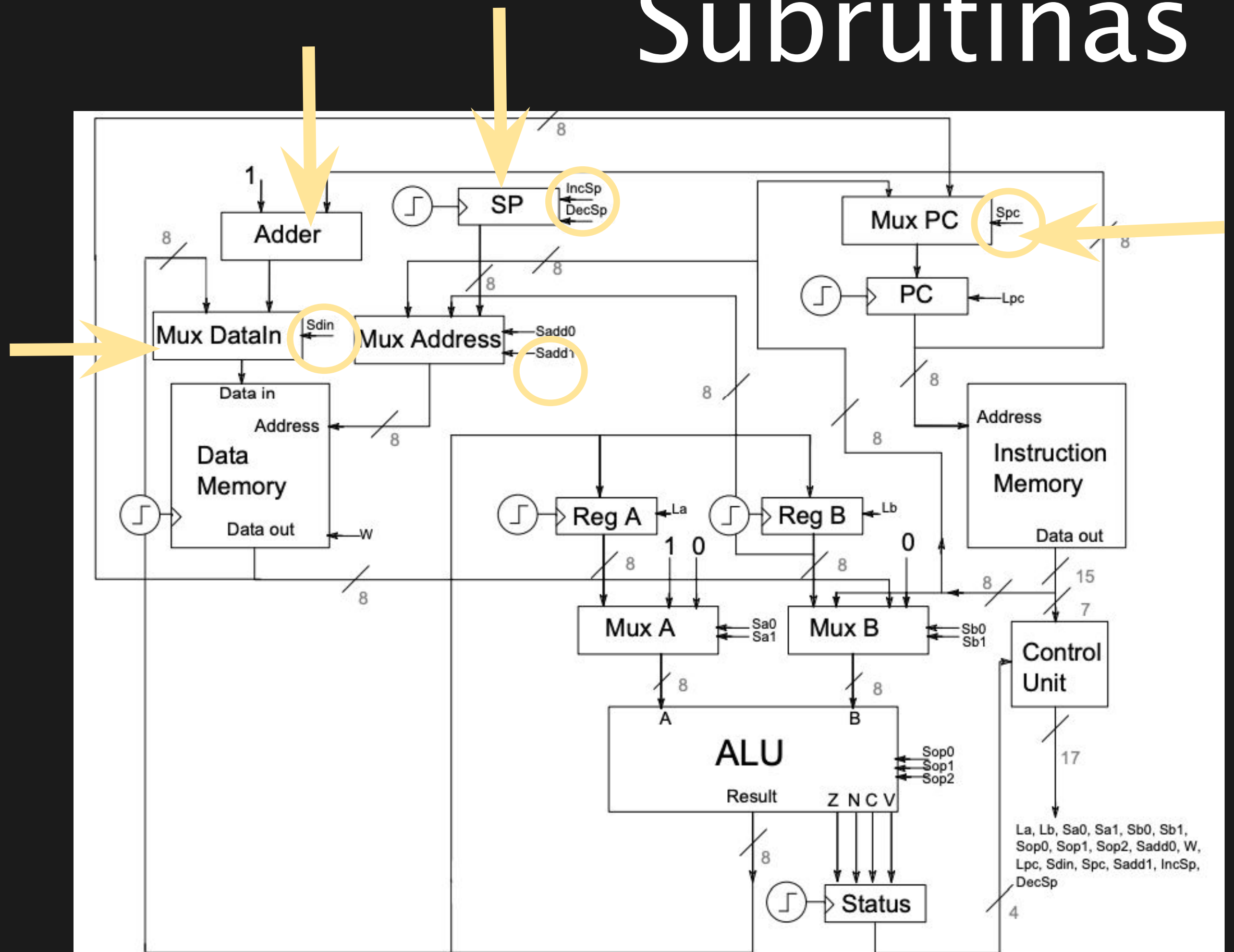
- Mux PC
- Adder
- Mux DataIn
- SP
-



Subrutinas

Se agrega:

- Mux PC
- Adder
- Mux DataIn
- SP
- Señales de control:
 - Spc
 - Sdin
 - Sadd1
 - IncSp
 - DecSp



Subrutinas

DATA:

var1: 1

var2: 2

CODE:

MOV A, (var1)

MOV B, var2

ADD A, 2

ADD B, 3

CALL FUNC1

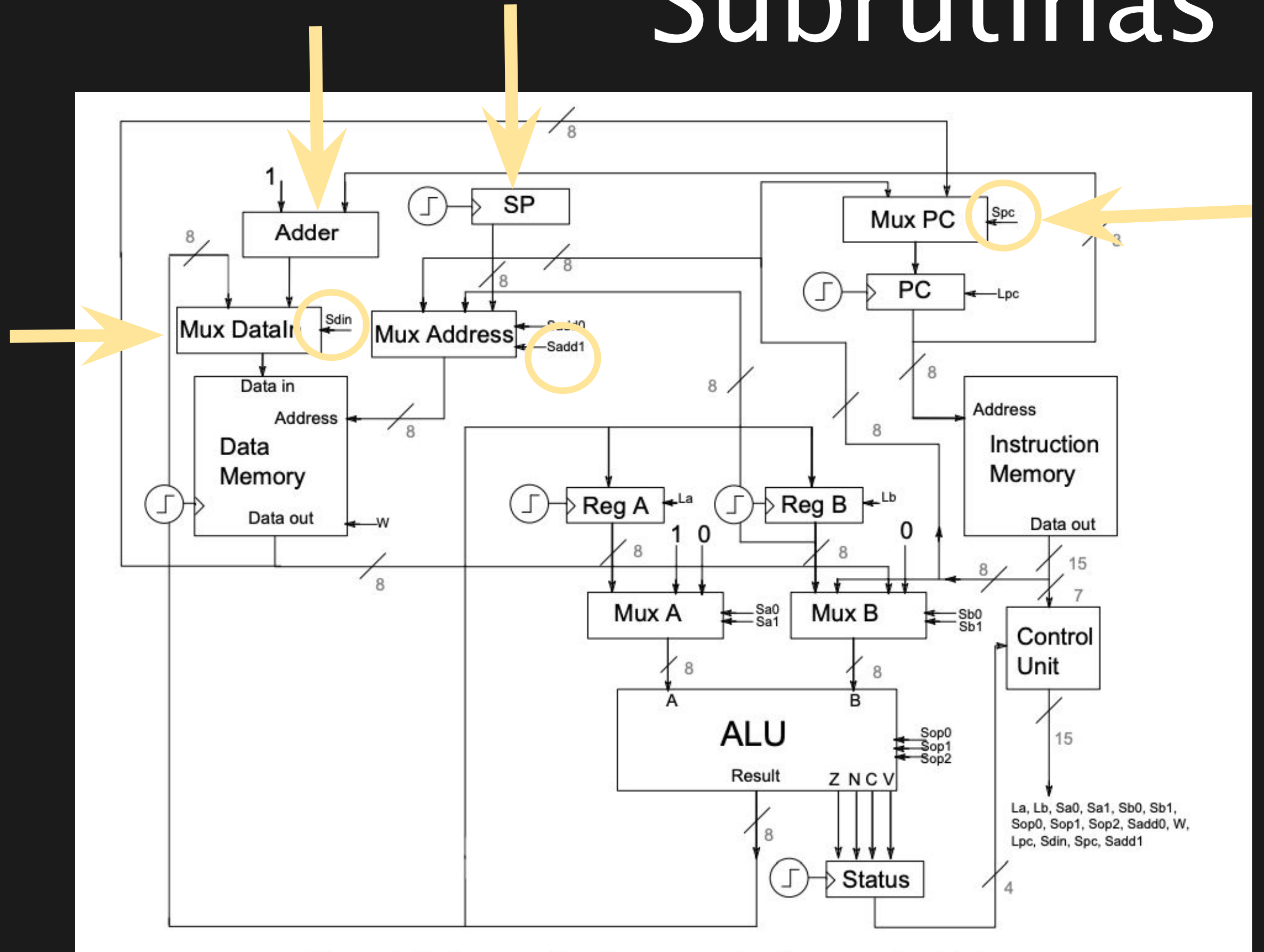
...

FUNC1

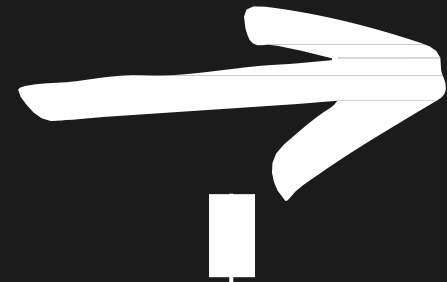
SUB A, 1

MOV (B), A

RET



Python

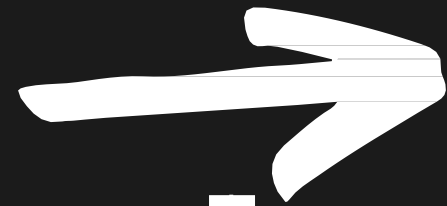


Assembly

```
n='0001011010'  
c=0  
while len(n)>0:  
    if n[0]==1:  
        c+=1  
    if len(n)>1:  
        n=n[1:]  
    else:  
        n=''
```

Python

```
n='0001011010'  
c=0  
while len(n)>0:  
    if n[0]==1:  
        c+=1  
    if len(n)>1:  
        n=n[1:]  
    else:  
        n=''
```



Assembly

Data:

n: 0001011010

CODE:

MOV A, (n)

MOV C, 0

Python

```
n='0001011010'
c=0
while len(n)>0:
    if n[0]==1:
        c+=1
    if len(n)>1:
        n=n[1:]
    else:
        n=''
```



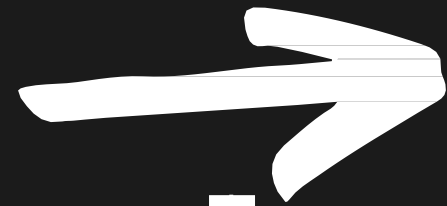
Assembly

```
Data:
    n: 0001011010
CODE:
MOV A, (n)
MOV C, 0
while:

CMP A, 0
JGT while
```

Python

```
n='0001011010'  
c=0  
while len(n)>0:  
    if n[0]==1:  
        c+=1  
    if len(n)>1:  
        n=n[1:]  
    else:  
        n=''
```

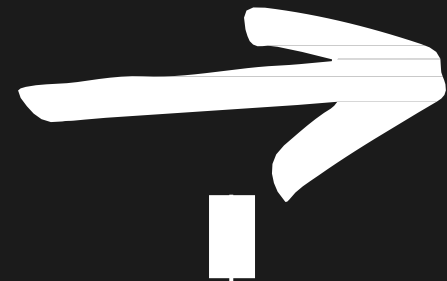


Assembly

```
Data:  
    n: 0001011010  
CODE:  
MOV A, (n)  
MOV C, 0  
while:  
    CMP A[0], 1  
    JNE fin_if  
  
    fin_if:  
    CMP A, 0  
    JGT while
```

Python

```
n='0001011010'
c=0
while len(n)>0:
    if n[0]==1:
        c+=1
    if len(n)>1:
        n=n[1:]
    else:
        n=''
```



Assembly

```
Data:
    n: 0001011010
CODE:
MOV A, (n)
MOV C, 0
while
    CMP A[0], 1
    JNE fin_if
    ADD C, 1
    fin_if
    SHL A
    CMP A, 0
    JGT while
```

Ej 2

Modifique la arquitectura del computador básico tal que se permitan hacer las siguientes funcionalidades:

- a) Agregar un tercer registro que cumpla la función de acumulador de resultados, además de las funcionalidades clásicas de un registro y que pueda usarse para direccionamiento indirecto.
- b) Agregar las instrucciones ADD3 reg1, reg2, reg3 y SUB3 reg1, reg2, reg3, que toman los valores en los registros reg1, reg2 y reg3, los suma/resta y los almacena en reg1

- a) Agregar un tercer registro que cumpla la función de acumulador de resultados, además de las funcionalidades clásicas de un registro y que pueda usarse para direccionamiento indirecto.

Ej 2

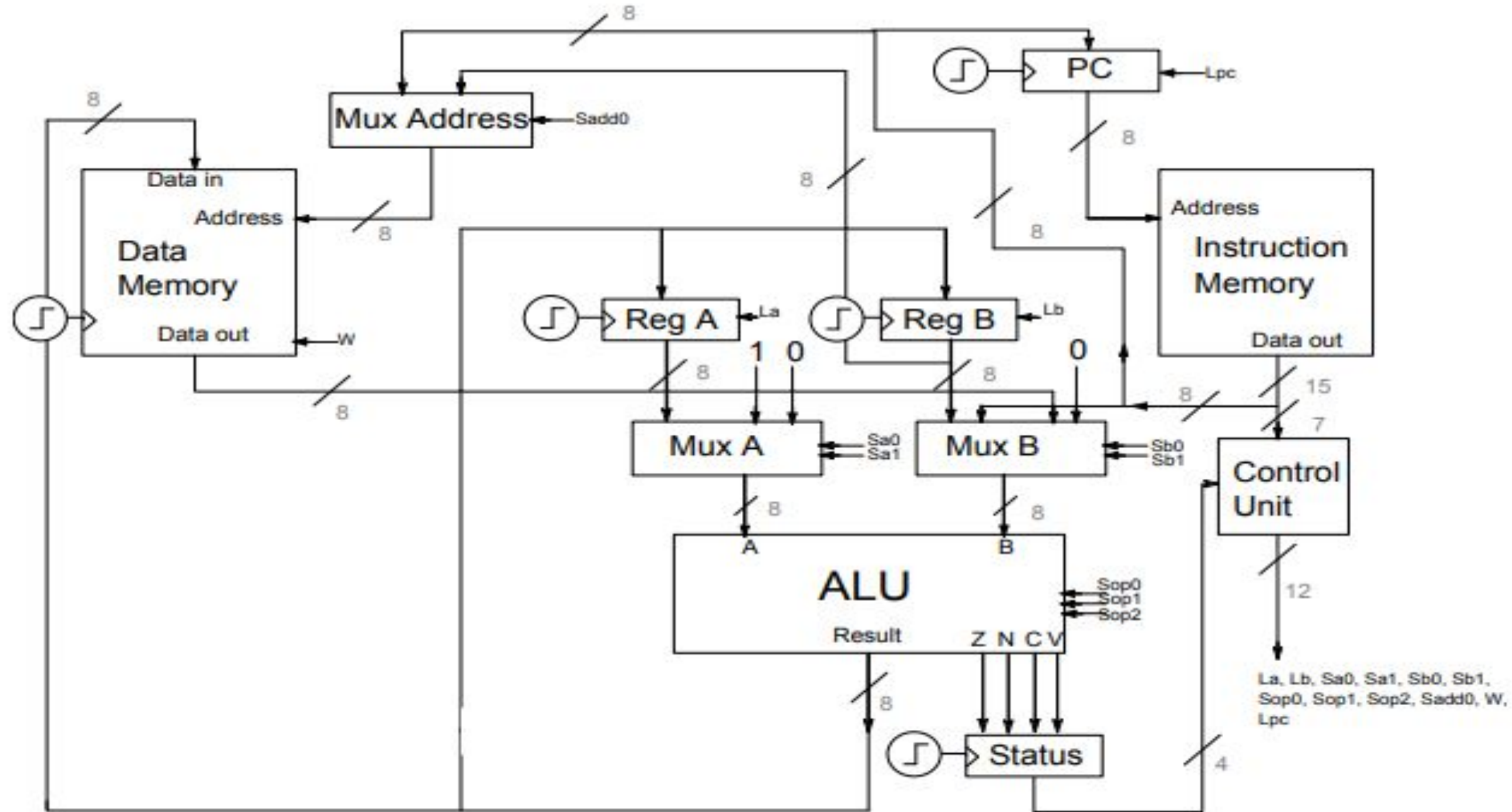


Figure 2: Computador con salto condicional.

b) Agregar las instrucciones ADD3 RegA, RegB, RegC y SUB3 RegA, RegB, RegC, que toman los valores en los registros RegA, RegB y RegC, los suma/resta y los almacena en RegA

Ej 2

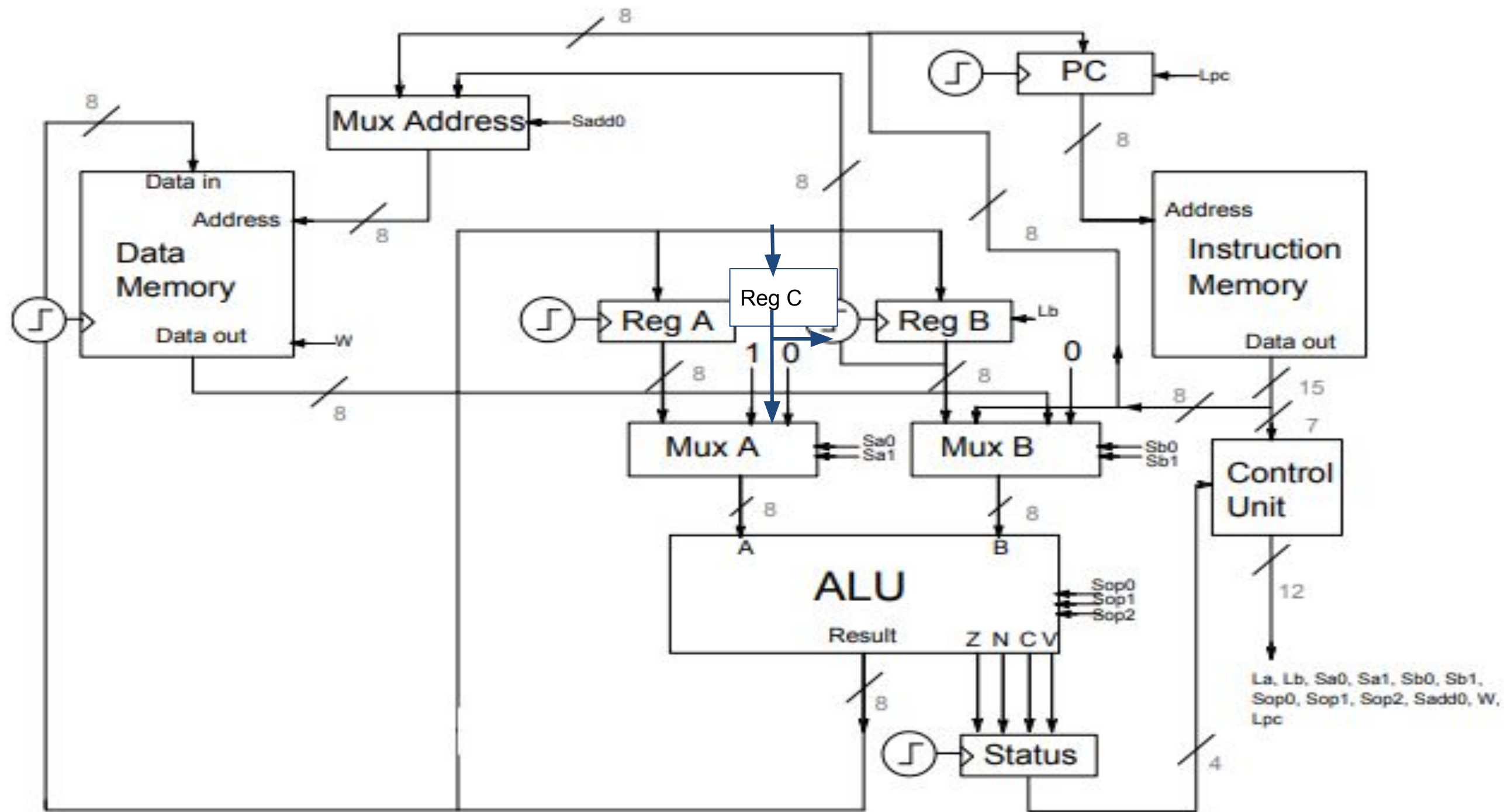


Figure 2: Computador con salto condicional.


Ej 3 (2013-2)

Considerando el siguiente código:

```
MOV A,0  
MOV B,2  
label1:  
CMP A,B  
JGE end  
ADD A,1  
JMP label1  
end:
```

- a) Determine la secuencia efectiva de instrucciones.
- b) Realice las modificaciones necesarias al computador básico de manera que se pueda implementar la instrucción **NUM_INSTRUCTIONS** (var1) la cual almacena en la variable var1 la cantidad de instrucciones que efectivamente se ejecutaron hasta que se llamó a esta instrucción (no incluyendo a esta instrucción).


Ej 3



```
MOV A,0  
MOV B,2  
label1:  
CMP A,B  
JGE end  
ADD A,1  
JMP label1  
end:
```

Ej 3

```
MOV A,0
MOV B,2
label1:
CMP A,B
JGE end
ADD A,1
JMP label1
end:
```



```
MOV A, 0 // Instruccion 0
MOV B, 2 // Instruccion 1
CMP A,B // Instruccion 2
JGE end // Instruccion 3
ADD A ,1 // Instruccion 4
JMP label1 // Instruccion 5
CMP A,B // Instruccion 2
JGE end // Instruccion 3
ADD A ,1 // Instruccion 4
JMP label1 // Instruccion 5
CMP A,B // Instruccion 2
JGE end // Instruccion 3
```

b) Realice las modificaciones necesarias al computador básico de manera que se pueda implementar la instrucción **NUM_INSTRUCTIONS** (var1) la cual almacena en la variable var1 la cantidad de instrucciones que efectivamente se ejecutaron hasta que se llamó a esta instrucción (no incluyendo a esta instrucción).

Ej 3

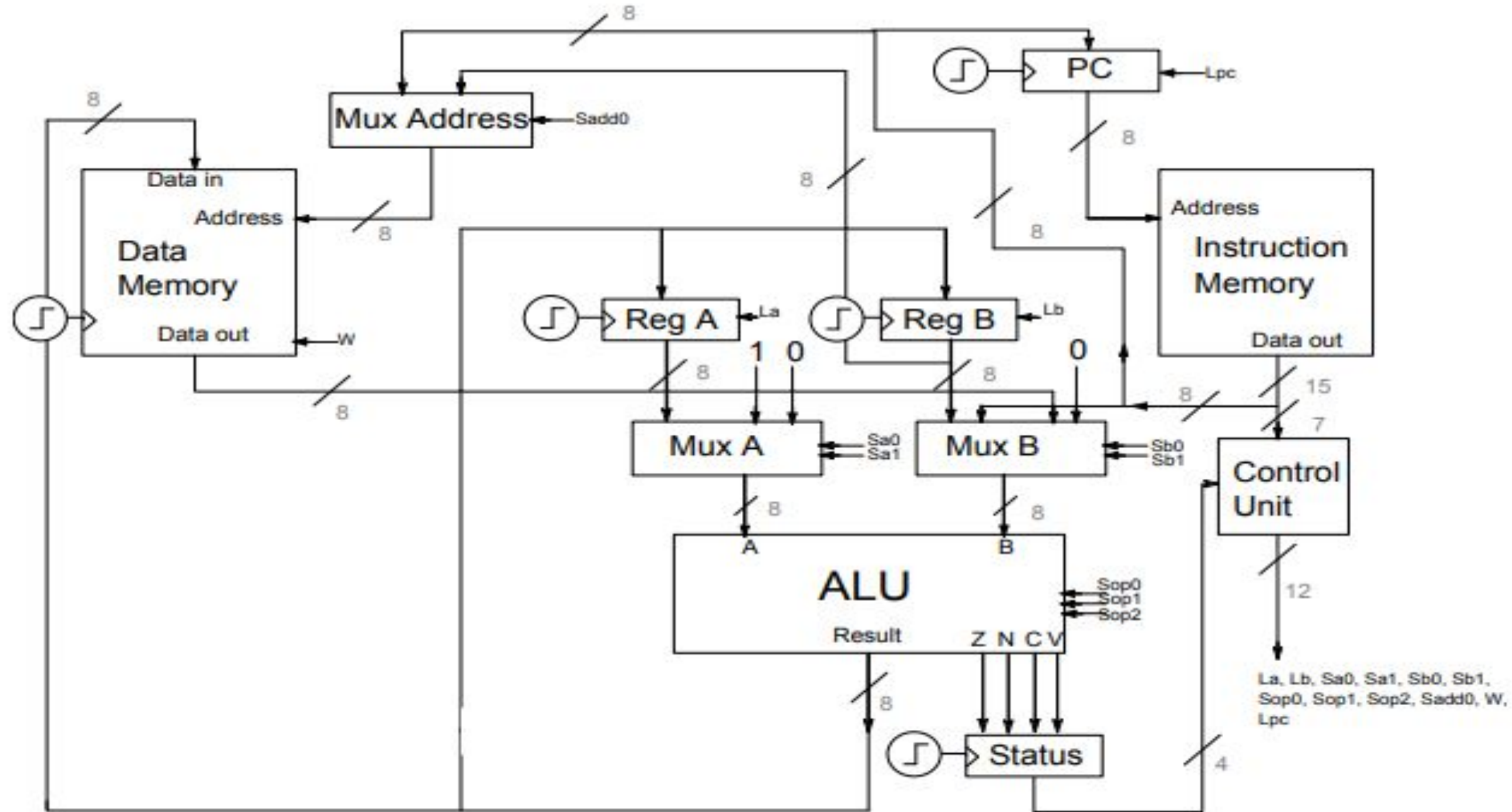


Figure 2: Computador con salto condicional.

b) Realice las modificaciones necesarias al computador básico de manera que se pueda implementar la instrucción **NUM_INSTRUCTIONS** (var1) la cual almacena en la variable var1 la cantidad de instrucciones que efectivamente se ejecutaron hasta que se llamó a esta instrucción (no incluyendo a esta instrucción).

Ej 3

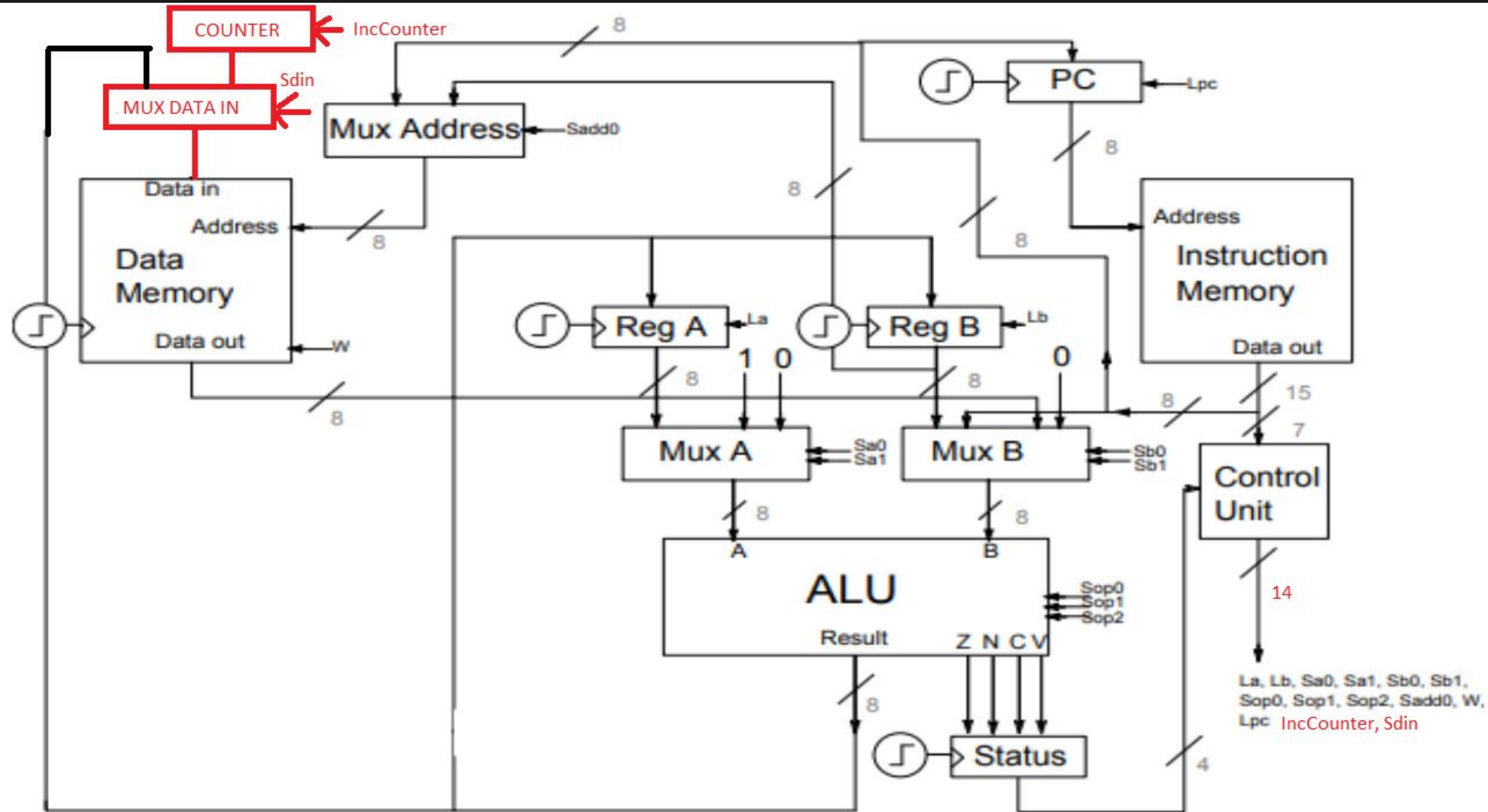


Figure 2: Computador con salto condicional.

¿DUDAS?