

---

# Estructura de Computadores

## *El computador simple*

---

Autores: David Guerrero, Isabel Gómez, Alberto Molina

Usted es libre de copiar, distribuir y comunicar públicamente la obra y de hacer obras derivadas siempre que se cite la fuente y se respeten las condiciones de la licencia Attribution-Share alike de Creative Commons.

Texto completo de la licencia: <http://creativecommons.org/licenses/by-nc-sa/3.0/es/>

---

---

# Guión

- ▶ **El punto de partida: La calculadora**
- ▶ **Automatización en la ejecución**
- ▶ **Almacenamiento de los datos**
- ▶ **Diversificación de instrucciones**
- ▶ **Una posible implementación**
- ▶ **Ejemplos de uso**

---

# Diversificación de instrucciones

- ▶ El computador simple 2 presenta muchas deficiencias: imposibilidad de realizar saltos en la ejecución del programa, ausencia de variables de estado que informen del resultado de las operaciones, imposibilidad de usar inmediatos falta de comunicación con el exterior...
- ▶ Se propone una arquitectura pensada para solventar estas deficiencias: el CS2010.
- ▶ Se ha procurado que la arquitectura del CS2010 sea similar a la del microcontrolador que veremos en el tema siguiente, aunque muchísimo más simple.

# Diversificación de instrucciones

## Modelo de usuario: registros visibles

- ▶ **Todos los registros visibles son de 8 bits**
- ▶ **Los de propósito general se etiquetan R0, R1, R2, R3, R4, R5, R6 y R7.**
- ▶ **Los de propósito específico son los siguientes:**
  - ▶ **PC o contador de programa:** Contiene la dirección de la próxima instrucción que se ejecutará. Se inicializa a cero y se va incrementando a medida que se ejecutan las instrucciones.
  - ▶ **SR o registro de estado:** Sus bits útiles se etiquetan Z (cero), V (desbordamiento), N (negativo) y C (carry). Indican, respectivamente, si la última instrucción aritmética/lógica generó un resultado con todos los bits a cero, un resultado no representable en complemento a 2, un resultado que es negativo al interpretarlo en complemento a 2 o si provocó carry/borrow.
  - ▶ **SP o puntero de pila:** Sirve para implementar la estructura de pila en memoria. Codifica la dirección de la primera posición libre. El puntero de pila se va decrementando a medida que se apilan datos y se incrementa al desapilarlos. Estos incrementos y decrementos se hacen de forma automatizada al realizar llamadas y retornos de subrutina.

---

# Diversificación de instrucciones

## Modelo de usuario: ISP

- ▶ Se pretende que el juego de instrucciones ensamblador del CS2010 sea básicamente un subconjunto muy reducido del de la arquitectura AVR.
- ▶ Esto no implica que el formato del código máquina en ambos sistemas sea similar: El formato de instrucciones del CS2010 es muchísimo más simple.
- ▶ Se usará un código de operación de longitud fija, 5 bits, lo que permite disponer un máximo de 32 códigos de operación.

# Diversificación de instrucciones

**Formatos:** Solo hay tres y comparten campos

<u>formato</u>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>A</b> instrucción con operando registro	código de operación					registro destino (fuente en ST)			-	-	-	-	-	registro fuente (registro base en ST/LD)		
<b>B</b> instrucción con operando memoria o inmediato									dato inmediato / dirección del dato							
<b>C</b> instrucción de salto						condición de salto			dirección de salto							

# Diversificación de instrucciones

## Asignación de códigos de operación: Facilita la decodificación

Bits del código de operación					NEMÓNICO	FORMATO	TIPO	SINTAXIS	EFECTO <sup>1</sup>	VNZC <sup>2</sup>
15	14	13	12	11						
0	0	0	0	0	ST	A	memoria	ST (Rbase), Rfuente	MEM[Rbase] ← Rfuente	----
0	0	0	0	1	LD	A	memoria	LD Rdestino, (Rbase)	Rdestino ← MEM[Rbase]	----
0	0	0	1	0	STS	B	memoria	STS dirección, Rfuente	MEM[dirección] ← Rfuente	----
0	0	0	1	1	LDS	B	memoria	LDS Rdestino, dirección	Rdestino ← MEM[dirección]	----
0	0	1	0	0	CALL	C	salto	CALL dirección	MEM[SP] ← PC, SP ← SP-1, PC ← dirección	----
0	0	1	0	1	RET	-	salto	RET	PC ← MEM[SP+1], SP ← SP+1	----
0	0	1	1	0	BRxx	C	salto	BRxx dirección	xx: PC ← dirección	----
0	0	1	1	1	JMP	C	salto	JMP dirección	PC ← dirección	----
0	1	0	0	0	ADD	A	aritmético/lógica	ADD Rdestino, Rfuente	Rdestino ← Rdestino + Rfuente	****
0	1	0	0	1	-	-	-	-	no documentado	UUUU
0	1	0	1	0	SUB	A	aritmético/lógica	SUB Rdestino, Rfuente	Rdestino ← Rdestino - Rfuente	****
0	1	0	1	1	CP	A	estado	CP Rdestino, Rfuente	NOP	****
0	1	1	0	0	-	-	-	-	no documentado	UUUU
0	1	1	0	1	-	-	-	-	no documentado	UUUU
0	1	1	1	0	-	-	-	-	no documentado	UUUU
0	1	1	1	1	MOV	A	movimiento de datos	MOV Rdestino, Rfuente	Rdestino ← Rfuente	----
1	0	0	0	0	-	-	-	-	no documentado	UUUU
1	0	0	0	1	-	-	-	-	no documentado	UUUU
1	0	0	1	0	CLC	-	estado	CLC	NOP	---*
1	0	0	1	1	SEC	-	estado	SEC	NOP	---*
1	0	1	0	0	ROR	A o B	desplazamiento	ROR Rdestino	Rdestino ← SHR(Rdestino, C)	****
1	0	1	0	1	ROL	A o B	desplazamiento	ROL Rdestino	Rdestino ← SHL(Rdestino, C)	****
1	0	1	1	0	-	-	-	-	no documentado	UUUU
1	0	1	1	1	STOP	-	especial	STOP	lleva el procesador a espera	----
1	1	0	0	0	ADDI	B	aritmético/lógica	ADDI Rdestino, dato	Rdestino ← Rdestino + dato	****
1	1	0	0	1	-	-	-	-	no documentado	UUUU
1	1	0	1	0	SUBI	B	aritmético/lógica	SUBI Rdestino, dato	Rdestino ← Rdestino - dato	****
1	1	0	1	1	CPI	B	estado	CPI Rdestino, dato	NOP	****
1	1	1	0	0	-	-	-	-	no documentado	UUUU
1	1	1	0	1	-	-	-	-	no documentado	UUUU
1	1	1	1	0	-	-	-	-	no documentado	UUUU
1	1	1	1	1	LDI	B	movimiento de datos	LDI Rdestino, dato	Rdestino ← dato	----

<sup>1</sup> (sin tener en cuenta el registro de estado y el incremento del PC)

<sup>2</sup> El caracter '-' denota "no modificado", '\*' denota "modificado de forma definida", 'U' denota "no documentado"

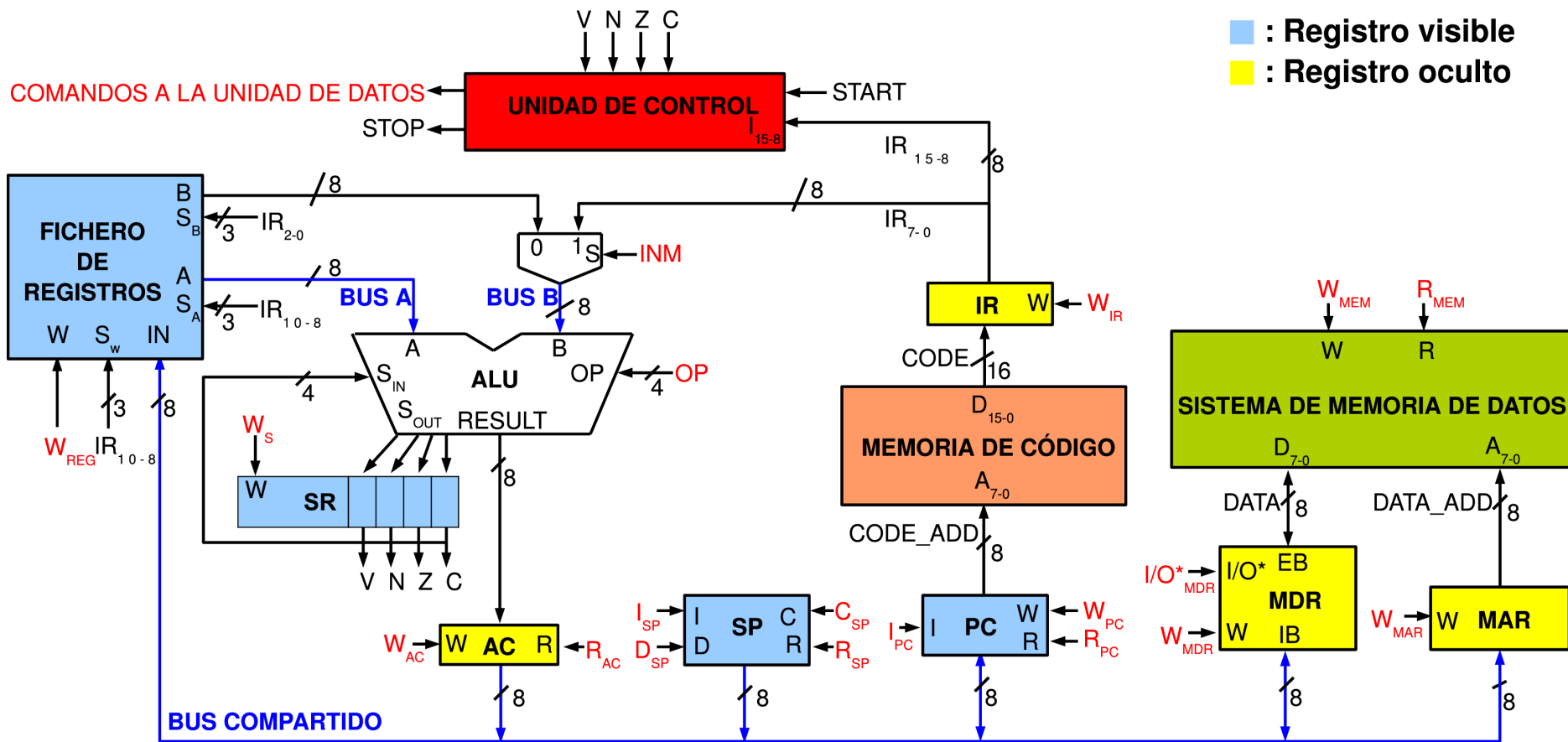
# Diversificación de instrucciones

Códigos de condición de la instrucción de bifurcación condicional: BR<sub>xx</sub>  
Los bits  $I_{10}I_9I_8$  codifican la condición de salto <sub>xx</sub>.

$I_{10}$	$I_9$	$I_8$	CONDICIÓN	nemónico(s) de la condición	notas
0	0	0	Z	ZS, EQ	será cierta justo tras realizar la resta A-B si y solo si A=B
0	0	1	C	CS, LO	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación base 2 sin signo
0	1	0	V	VS	será cierta si y solo si el dato recién calculado no es representable en notación complemento a 2
0	1	1	N xor V	LT	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación complemento a 2
1	-	-	?	-	estas condiciones no están definidas y no deben utilizarse

# Una posible implementación

■ : Registro visible  
■ : Registro oculto



---

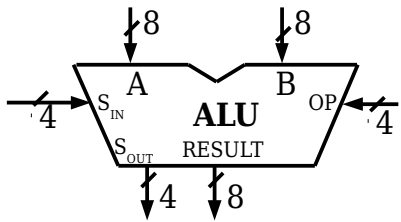
# Una posible implementación

## Registros ocultos

- ▶ **IR:** registro de 16 bits que sirve para almacenar la instrucción que está siendo ejecutada.
- ▶ **MDR.** Registro de 8 bits que sirve para almacenar los datos que se van a intercambiar con la memoria de datos.
- ▶ **MAR.** Registro de 8 bits que direcciona la memoria de datos.
- ▶ **AC.** Registro de 8 bits para almacenar temporalmente el resultado de la ALU

# Una posible implementación

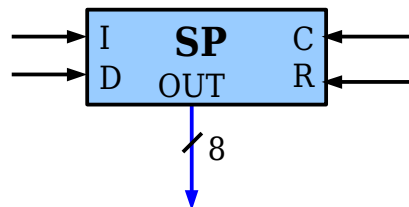
## Descripción RT de los nuevos componentes



$$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$$

$$S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$$

OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	C <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-



R	OUT:=
0	HI
1	SP

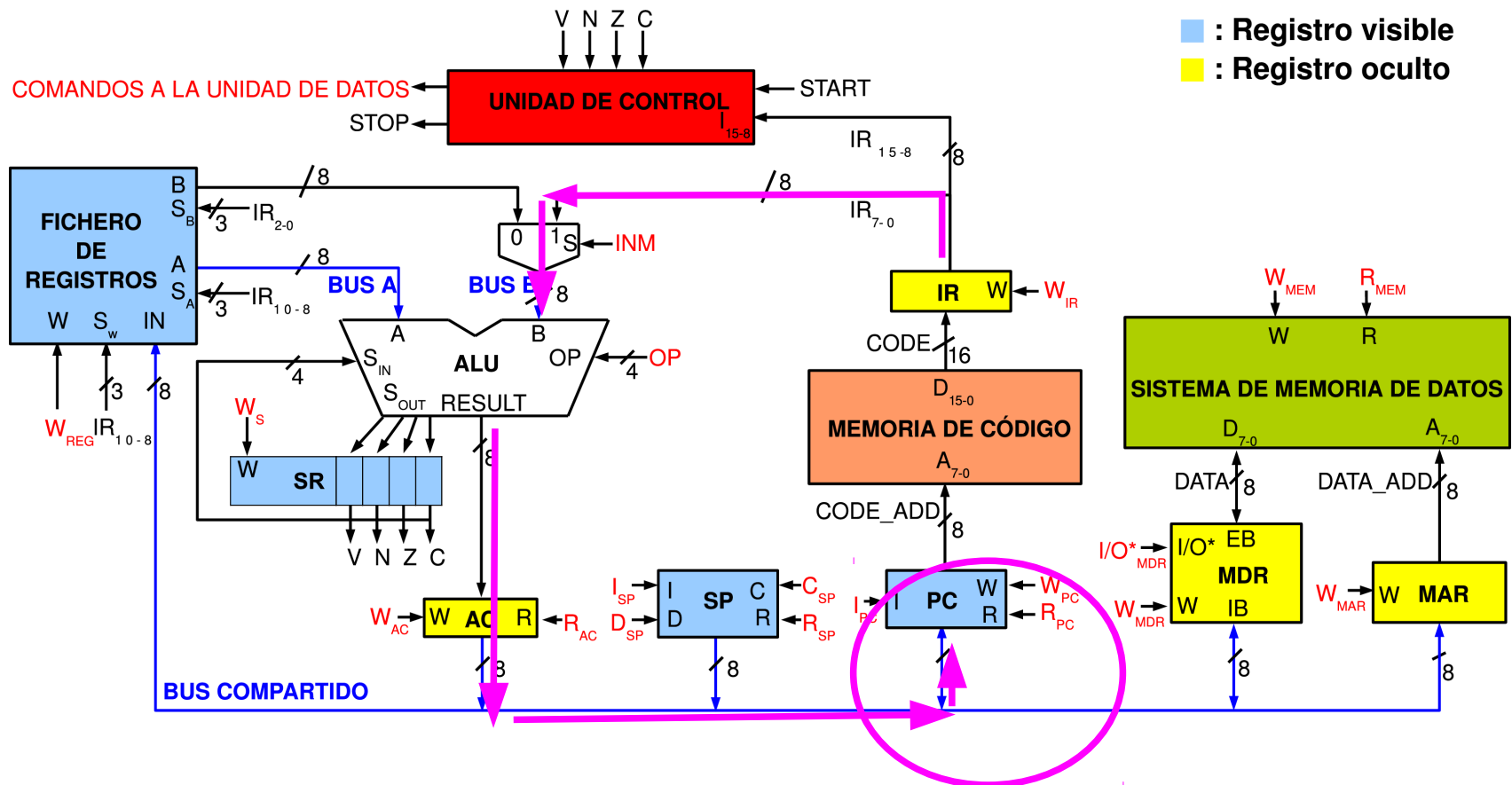
I D C	SP ←
0 0 0	SP
0 0 1	0
0 1 0	SP-1
1 0 0	SP+1
OTRAS	PROHIBIDAS

La descripción de SR es idéntica a la del MAR

---

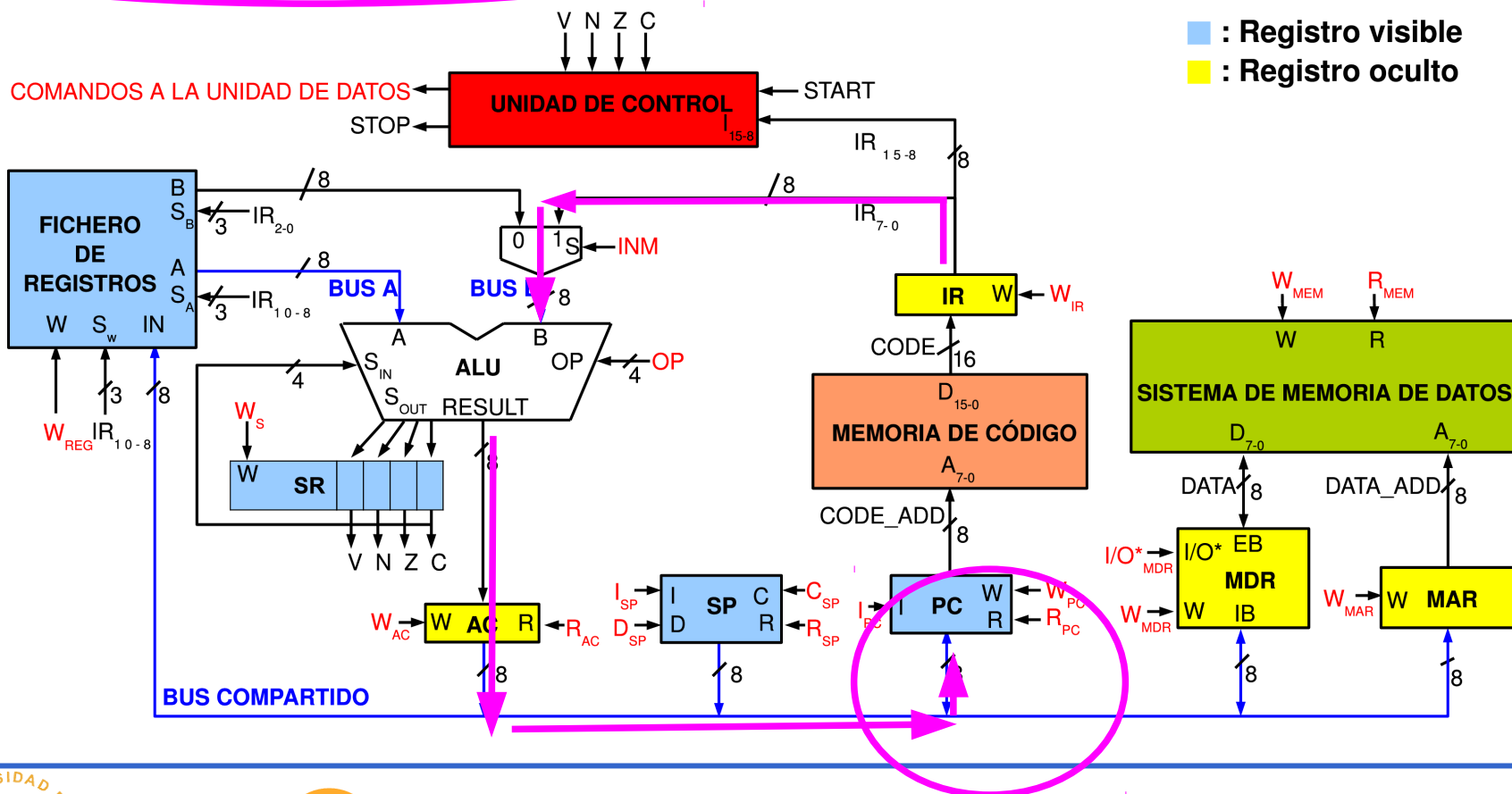
# **Modificaciones en la arquitectura y el formato de instrucción que nos permiten añadir instrucciones de salto**

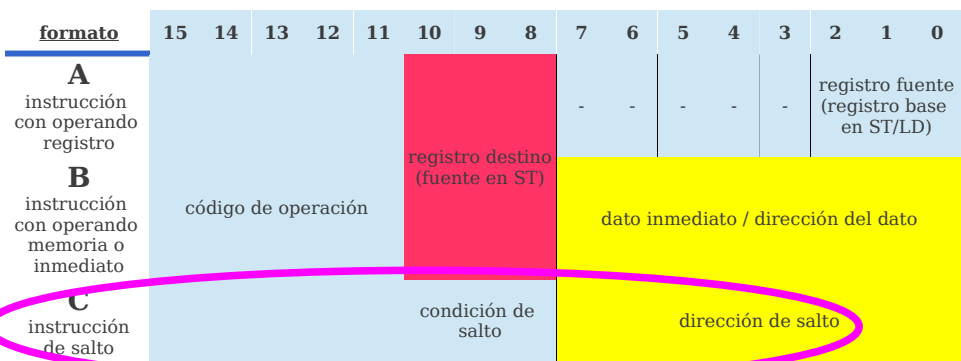
¿Cómo se hace el salto?: El PC se debe actualizar a la dirección del salto.  
 ¿Dónde está la dirección del salto?: guardada en el código de la instrucción.  
 El sistema debe de permitir escribir esa dirección en el PC. Necesitamos conectar el PC al bus para establecer un camino desde IR.



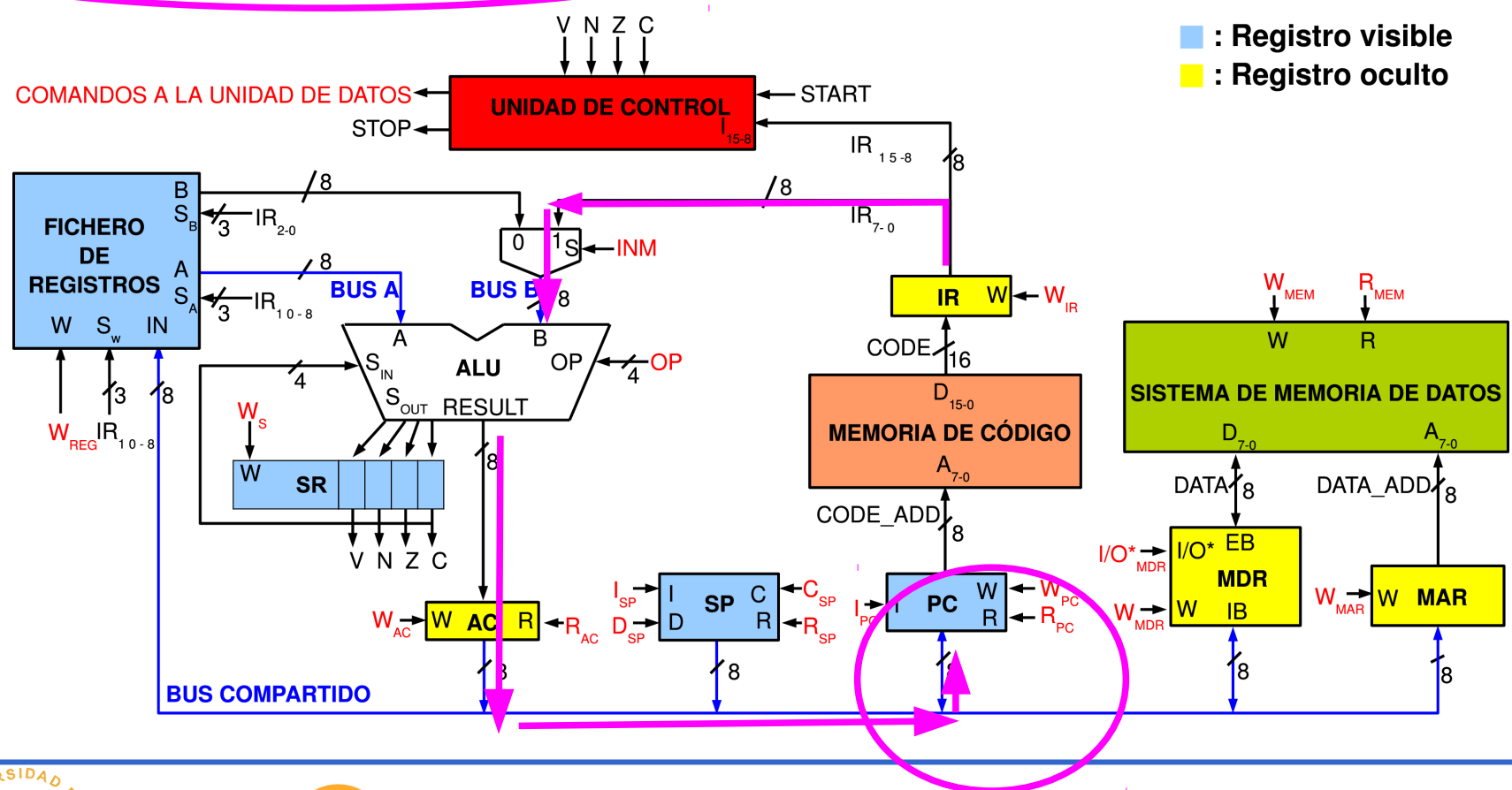
<u>formato</u>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>A</b> instrucción con operando registro	código de operación					registro destino (fuente en ST)			-	-	-	-	-	registro fuente (registro base en ST/LD)		
<b>B</b> instrucción con operando memoria o inmediato									dato inmediato / dirección del dato							
<b>C</b> instrucción de salto	condición de salto					dirección de salto										

Ha sido necesario añadir un nuevo formato que nos permita codificar las instrucciones de salto que explicaremos a continuación.



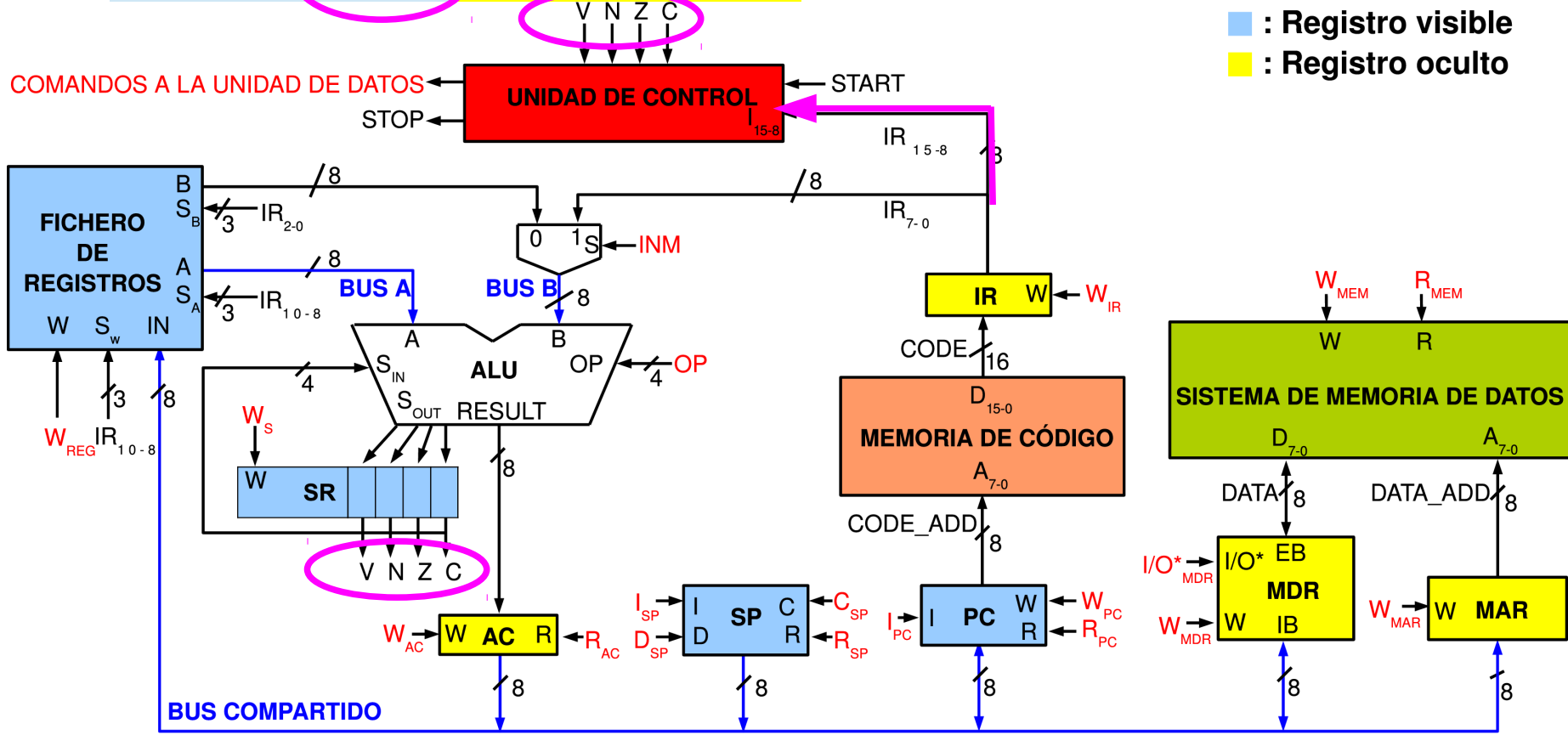


**Salto incondicional.**  
 Salta siempre. No se utiliza el campo condición de salto. Los 8 bits del IR se guardan en el PC y eso produce el salto. La próxima instrucción que busque el sistema será la que se encuentre en la posición indicada por la dirección de salto.





**Salto condicional.**  
Salta solo si se cumple una condición  
codificada en la instrucción y que será  
evaluada por la unidad de control.



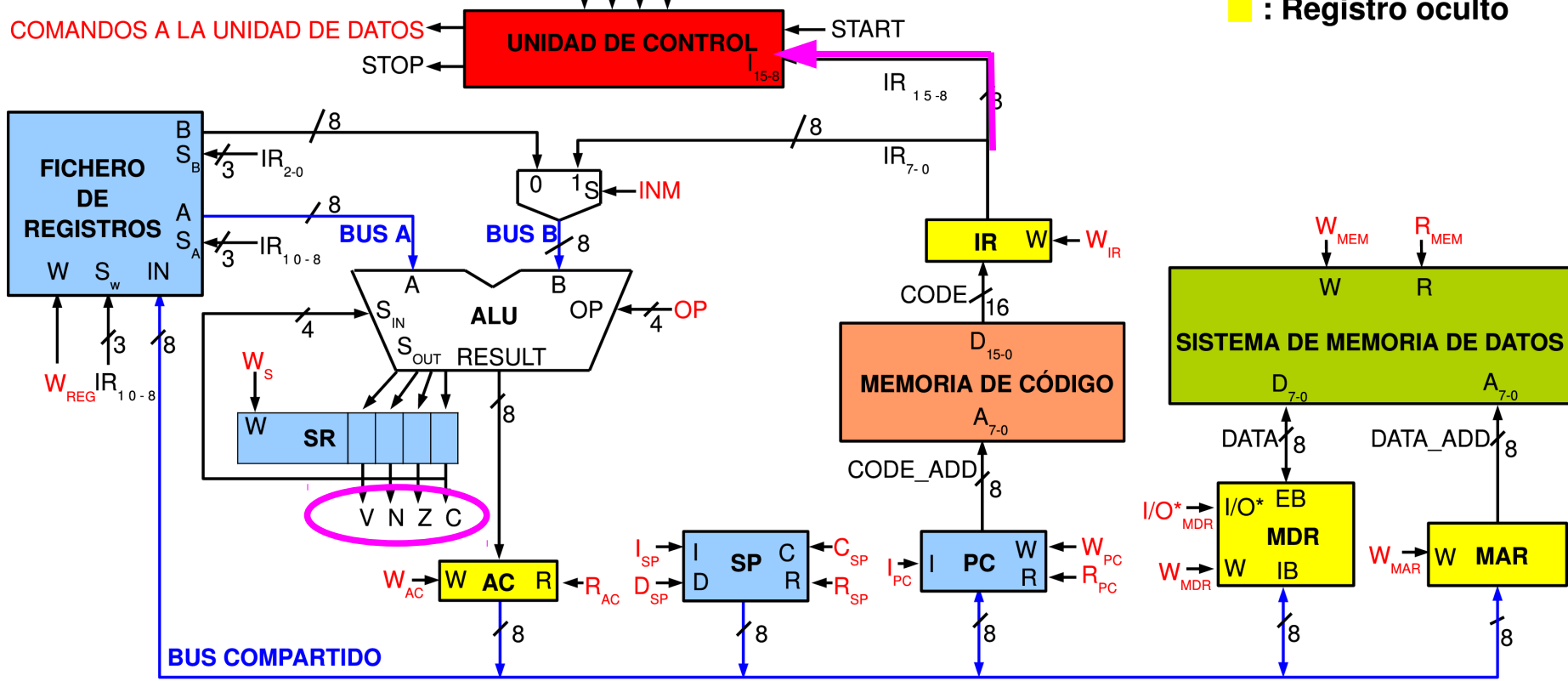


## Salto condicional.

La condición para que se produzca un salto viene dada por el resultado de una operación que se ejecuto previamente. ¿Fue el dato 0, positivo, desbordamiento?

■ : Registro visible  
■ : Registro oculto

COMANDOS A LA UNIDAD DE DATOS





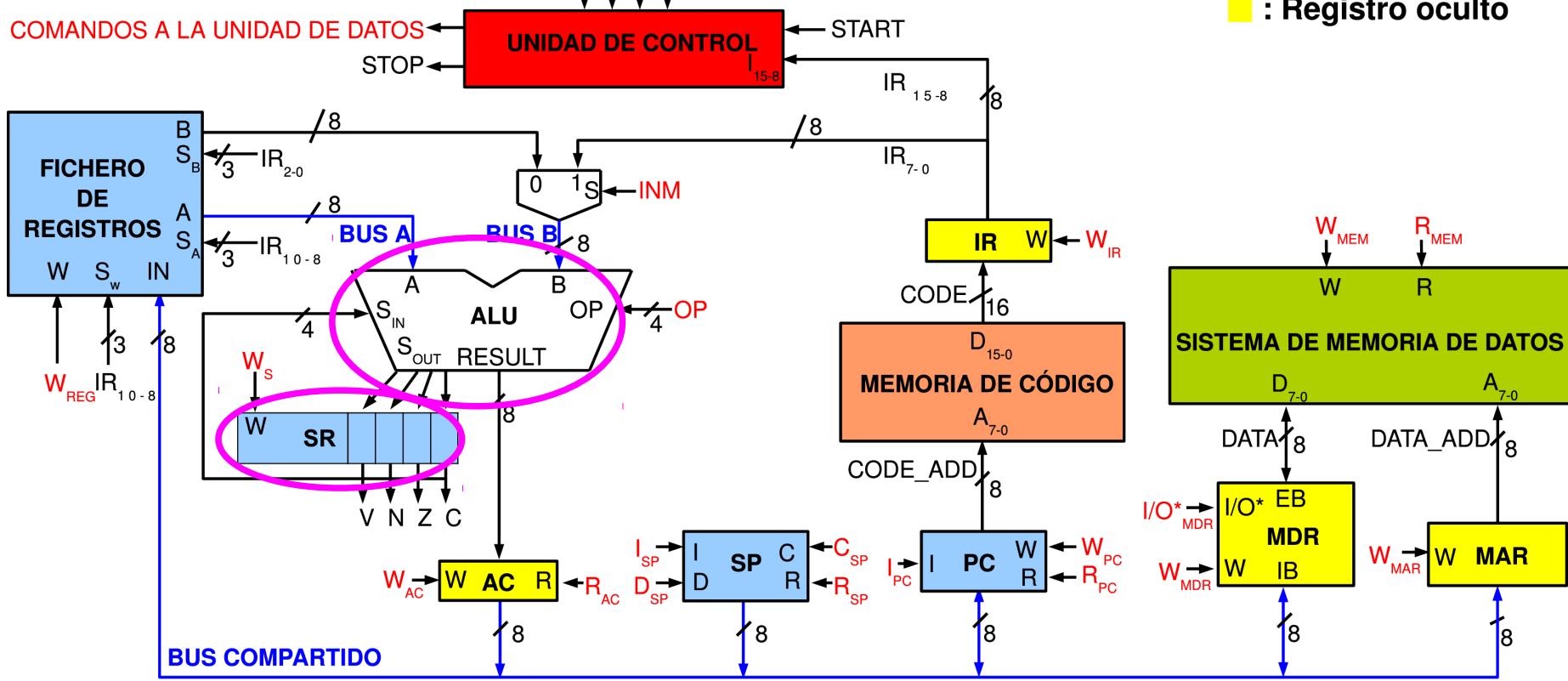
## Salto condicional.

Necesitamos una ALU que genere esta información. La información se guardará en los flags del registro de estado (SR).

La complejidad de la ALU y el registro SR son añadidos a la nueva arquitectura

■ : Registro visible  
■ : Registro oculto

COMANDOS A LA UNIDAD DE DATOS

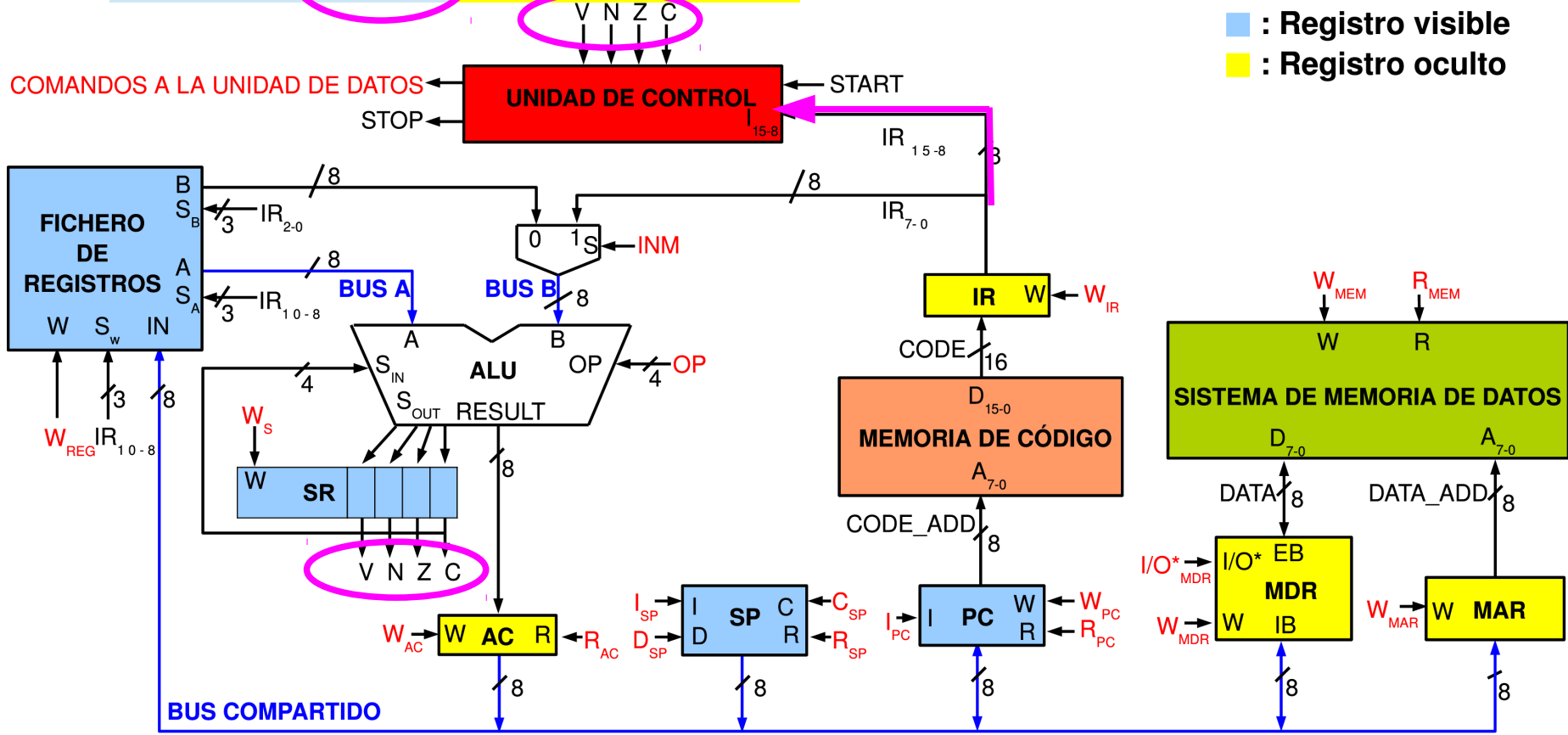


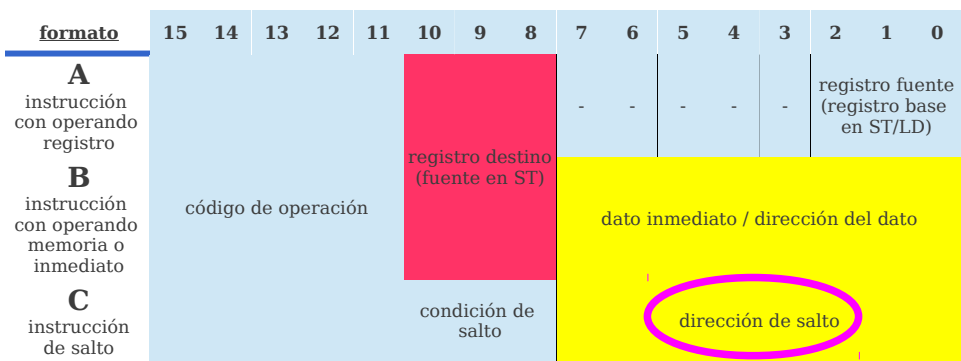
## Información de lo que significan los códigos de condición

$I_7$	$I_9$	$I_8$	CONDICIÓN	nemónico(s) de la condición	notas
0	0	0	Z	ZS, EQ	será cierta justo tras realizar la resta A-B si y solo si A=B
0	0	1	C	CS, LO	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación base 2 sin signo
0	1	0	V	VS	será cierta si y solo si el dato recién calculado no es representable en notación complemento a 2
0	1	1	N xor V	LT	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación complemento a 2
1	-	-	?	-	estas condiciones no están definidas y no deben utilizarse



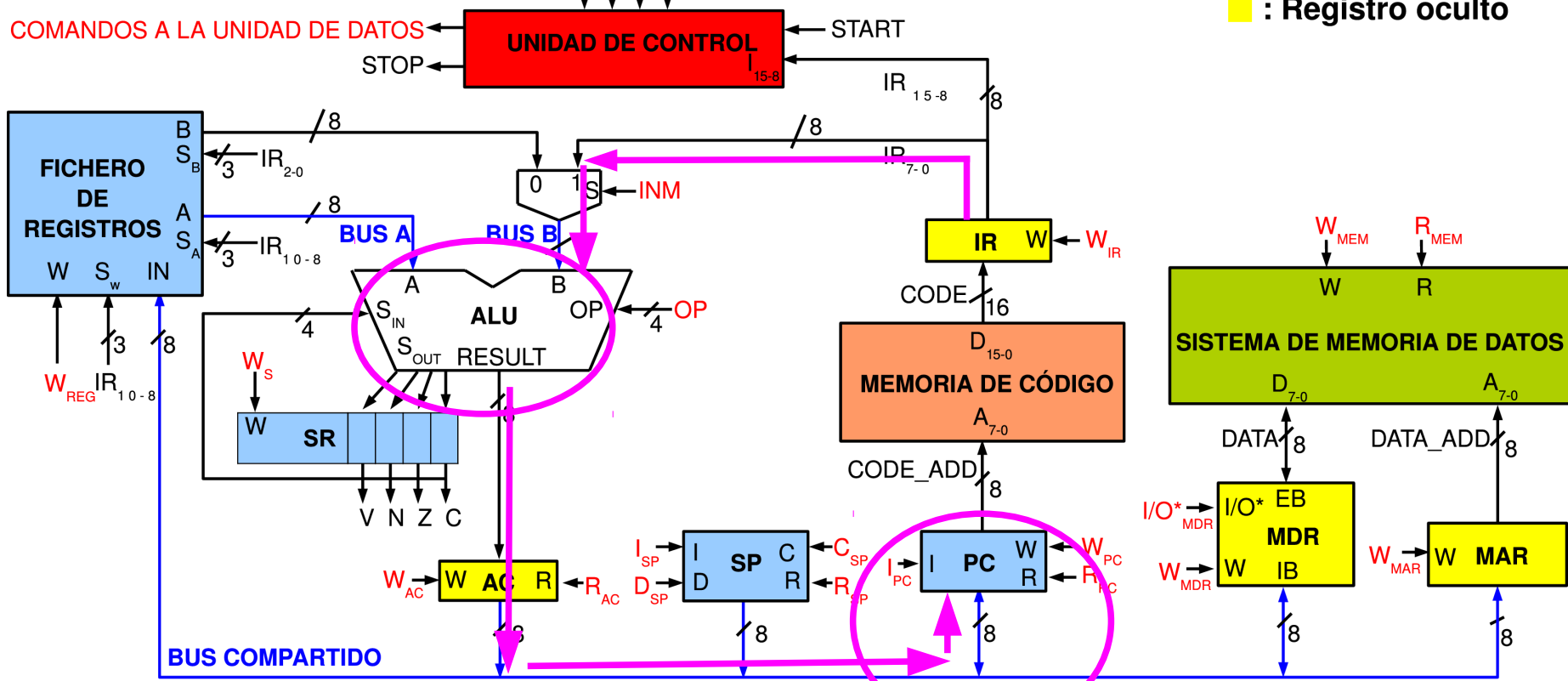
**Salto condicional.**  
Condición falsa: no se modifica el PC.





**Salto condicional.**  
Condición verdadera. El PC es modificado.

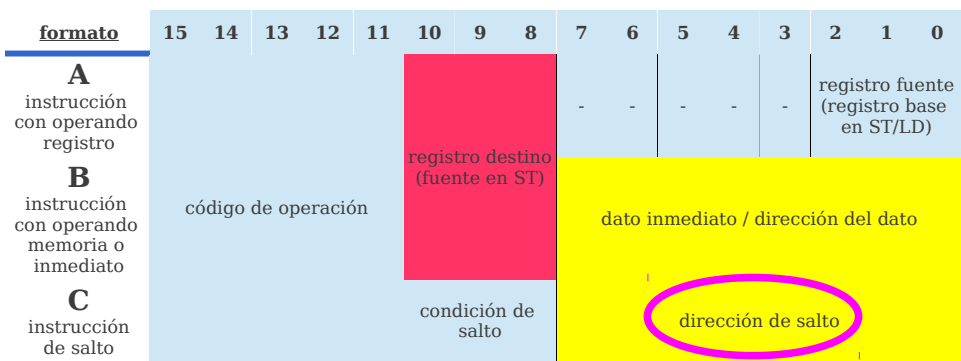
■ : Registro visible  
■ : Registro oculto



---

# Instrucciones de llamada y retorno de subrutinas

- Una subrutina es una secuencia de instrucciones que realizan una tarea que es necesario llevar a cabo en distintos puntos del programa.
- Cuando un programa necesita realizar la tarea hace una llamada a la subrutina. Esto hace que ésta se ejecute, tras lo cual la ejecución del programa se reanuda por la instrucción posterior a la llamada (retorno de subrutina).
- Una subrutina se puede llamar todas las veces que el programa lo necesite.
- Una subrutina puede llamar a su vez a otra subrutina: ejecución anidada de subrutinas.
- Una subrutina puede llamarse a sí misma: subrutinas recursivas.



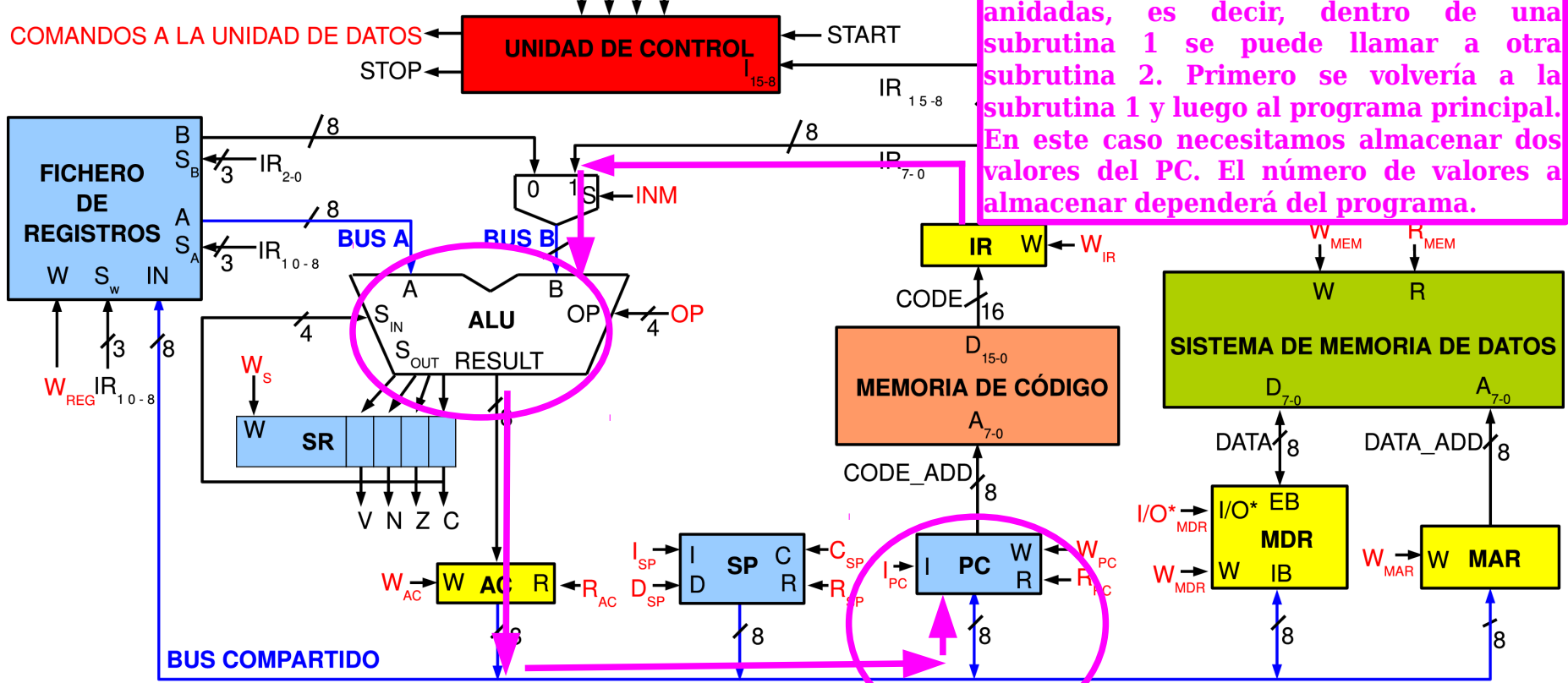
## Salto a subrutina.

Se trata de escribir en el PC la dirección donde comienza una serie de instrucciones que hay que ejecutar y luego volver a la instrucción siguiente a la de salto.

El PC se altera, pero luego hay que restaurar su valor original.

Además pueden existir subrutinas anidadas, es decir, dentro de una subrutina 1 se puede llamar a otra subrutina 2. Primero se volvería a la subrutina 1 y luego al programa principal. En este caso necesitamos almacenar dos valores del PC. El número de valores a almacenar dependerá del programa.

COMANDOS A LA UNIDAD DE DATOS



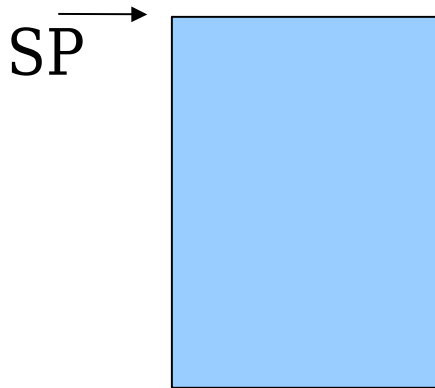


# Estructura de la Pila

Antes de meter un nuevo dato, el SP contiene la dirección de la posición de la pila donde este se guardará.

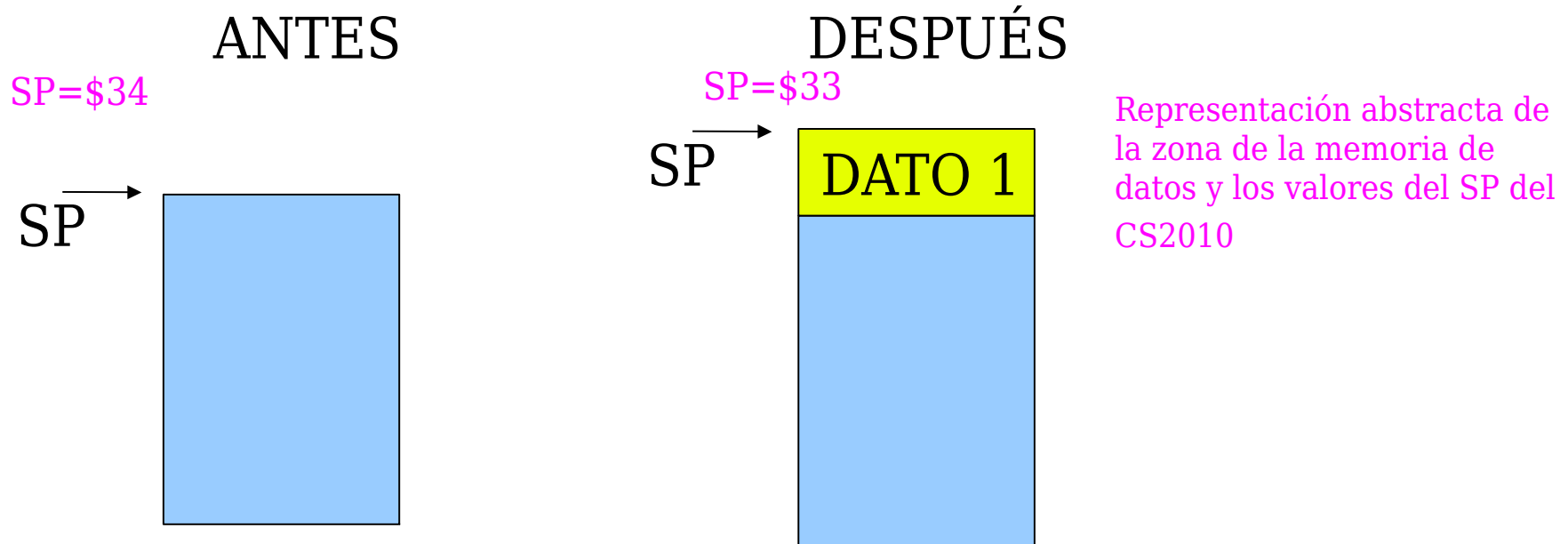
Es como si se hiciera una pila de libros, el primero que quito es el último que he puesto. El primer dato se guarda en una posición determinada y el siguiente en la posición inmediatamente inferior. Nuestra pila crece hacia direcciones decrecientes de la memoria.

Supongamos SP=\$34



Representación abstracta de la zona de la memoria de datos y los valores del SP del CS2010

# Estructura de la Pila



Proceso de guardar datos en la pila. Operación de PUSH

# Estructura de la Pila

ANTES

SP=\$33

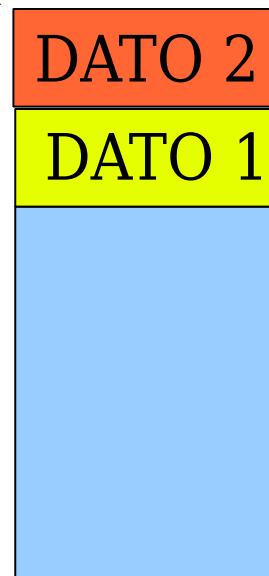
SP →



DESPUÉS

SP →

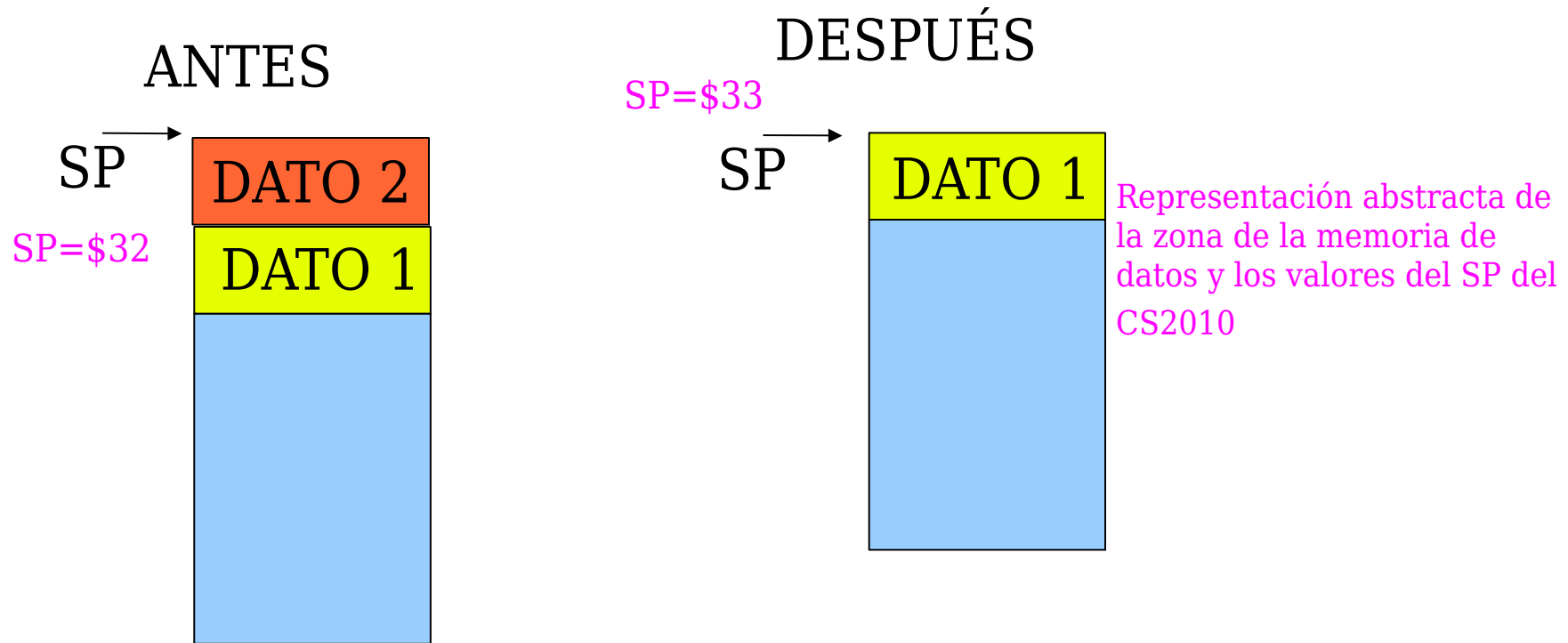
SP=\$32



Representación abstracta de la zona de la memoria de datos y los valores del SP del CS2010

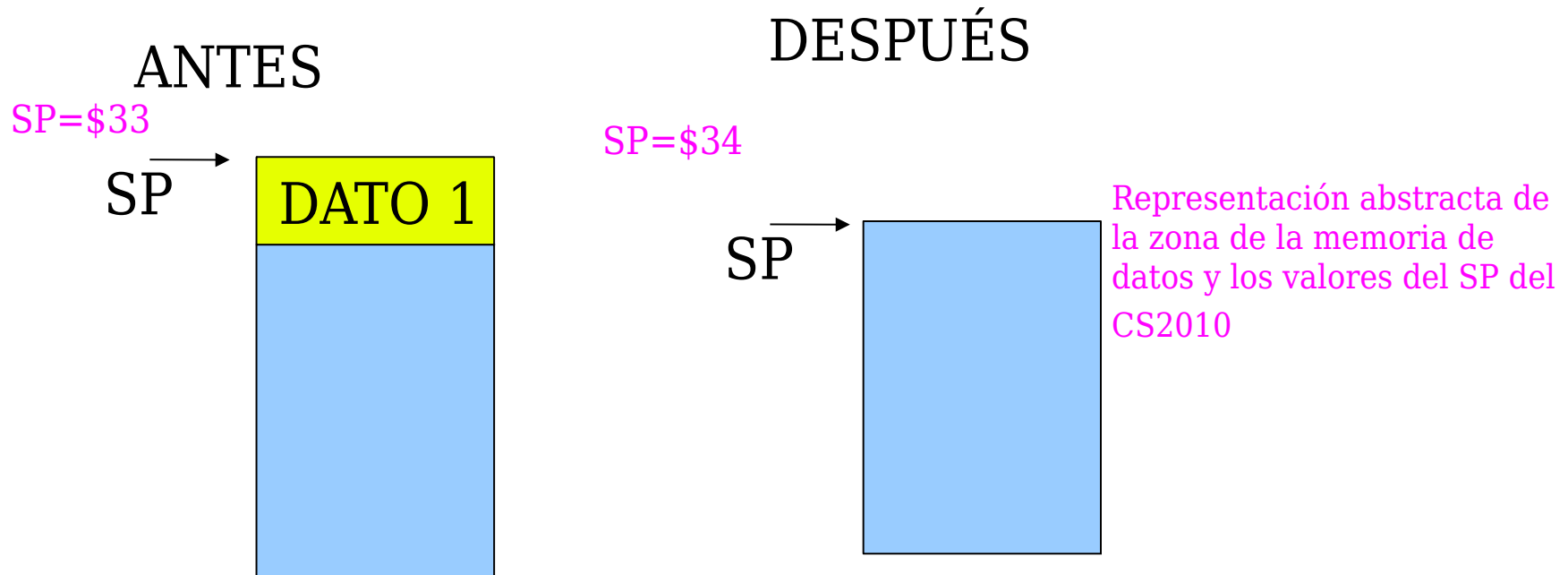
Proceso de guardar datos en la pila. Operación de PUSH

# Estructura de la Pila



Proceso de extracción de datos de la pila. Operación de PULL.  
El primero que se lee es el último que se escribió.

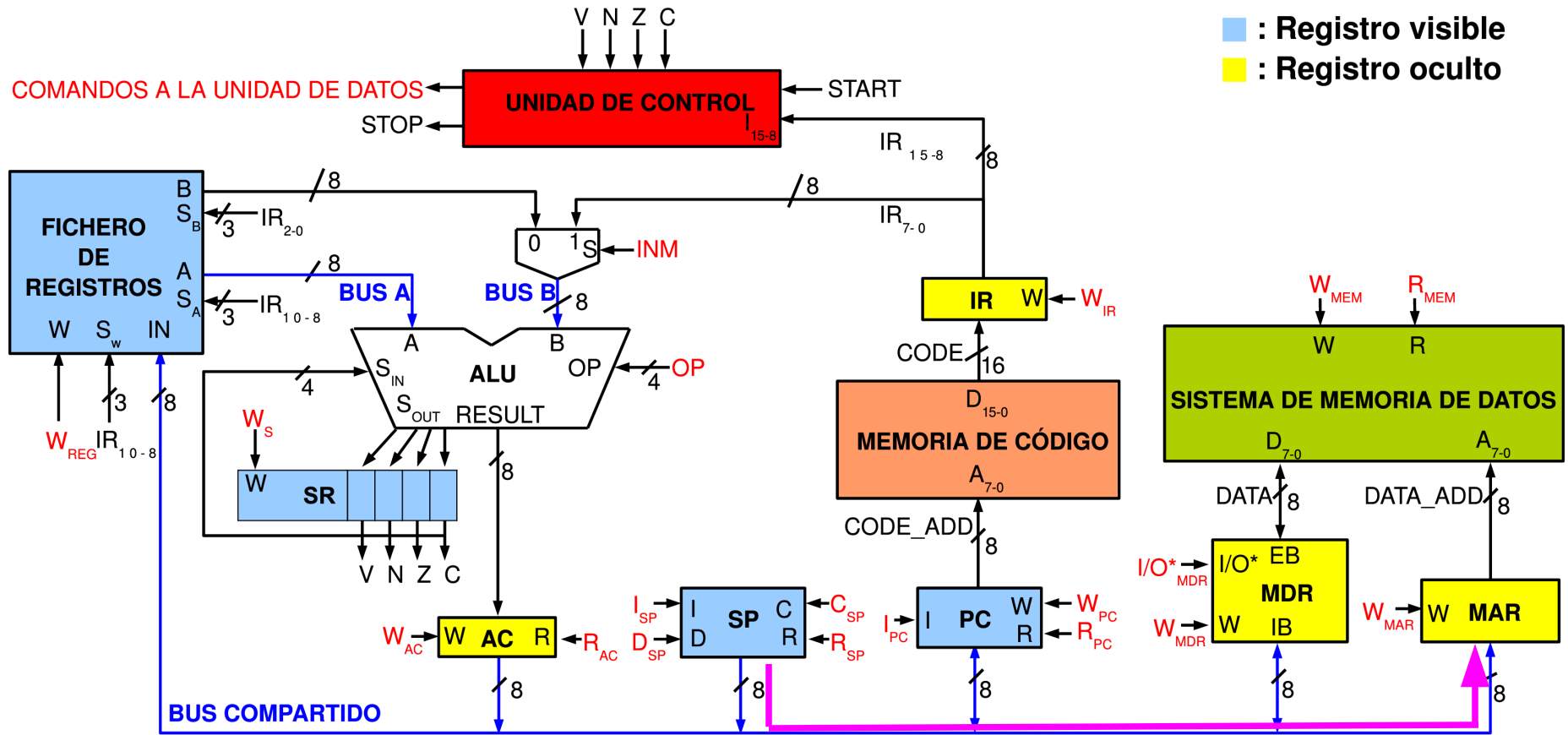
# Estructura de la Pila



Proceso de extracción de datos de la pila. Operación de PULL.  
El primero que se lee es el último que se escribió.

## Retorno de subrutina.

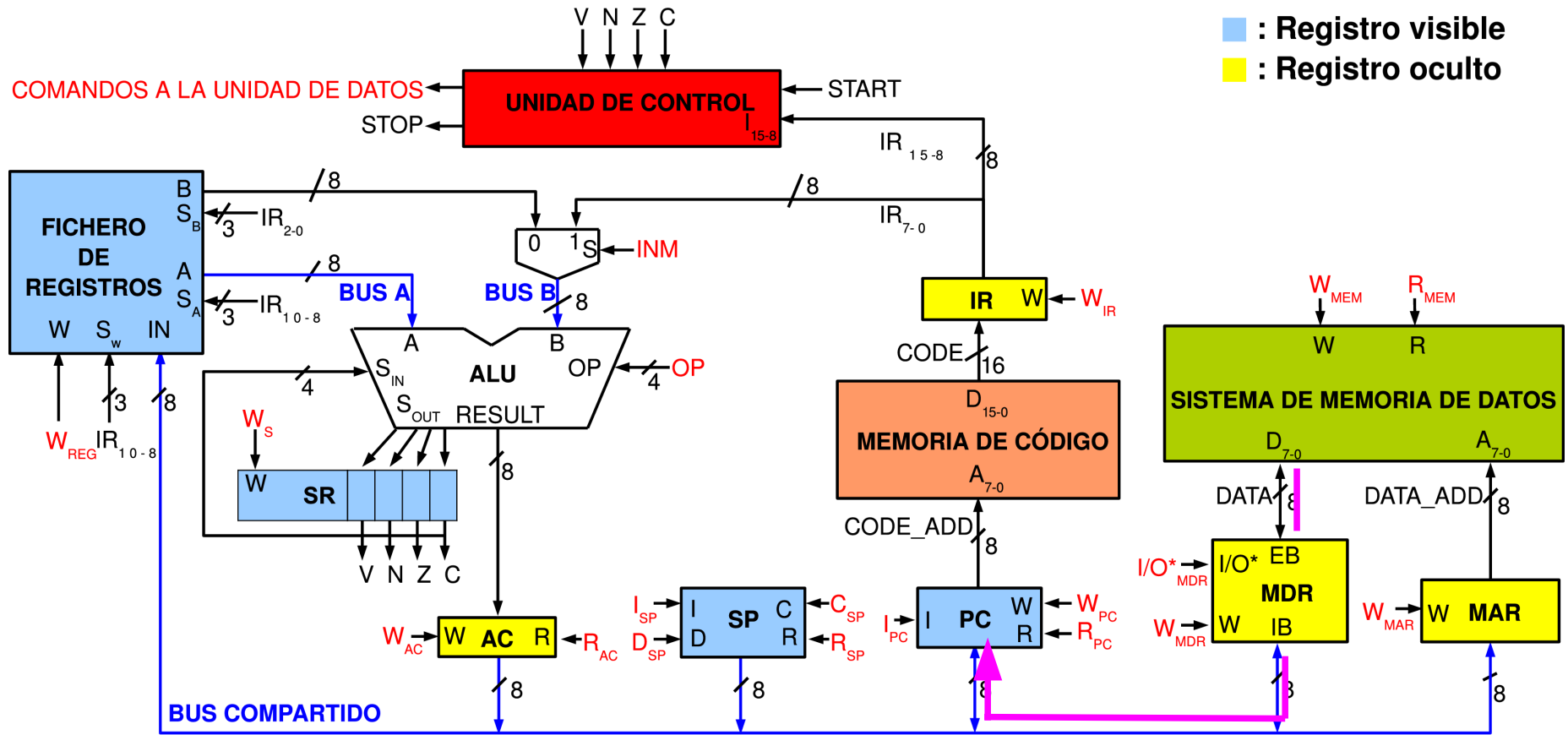
Se recuperan los valores del PC guardados para saber la dirección de salto.  
La dirección de retorno por lo tanto no se encuentra en el código de la instrucción.



1º y 2º paso: Incrementar SP y guardarlo en MAR. Esto permite direccionar la posición que contiene la dirección de retorno.

## Retorno de subrutina.

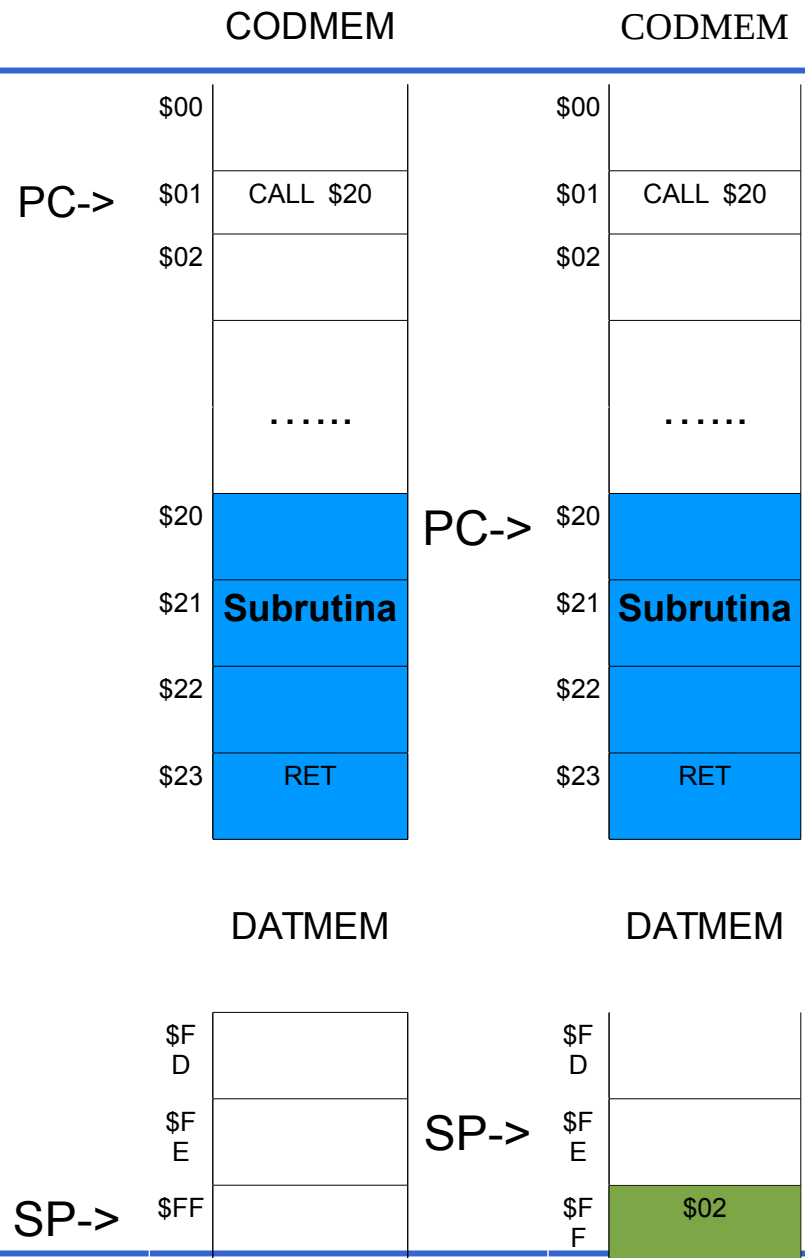
Se recuperan los valores del PC guardados para saber la dirección de salto.  
La dirección de retorno por lo tanto no se encuentra en el código de la instrucción.



**3º paso: Se lee de la pila la dirección de retorno y se escribe en el PC para volver a la instrucción siguiente a la de salto a subrutina.**

# Instrucciones de llamada y retorno de subrutinas

- Situación inicial (a). El computador ejecuta instrucciones y llega a CALL \$20.
- La ejecución de la instrucción CALL \$20 hace que:
  - Se guarde en la pila el PC (que contiene \$02)
  - Se “salte” a la dirección \$20
- Observe el comportamiento del puntero de pila o registro SP (b).

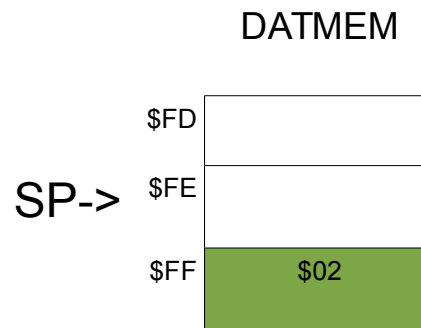
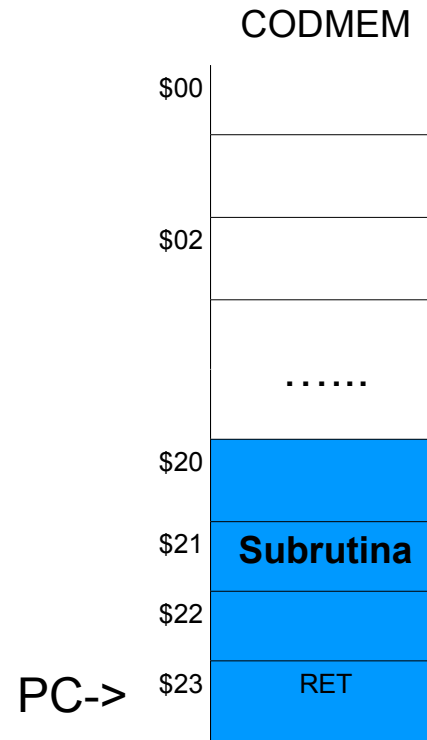


(a)

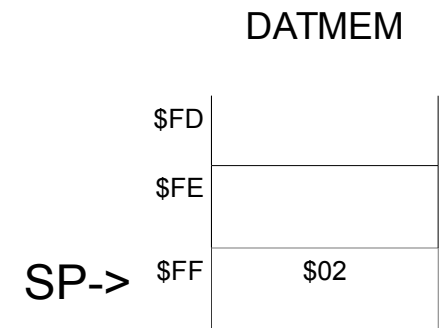
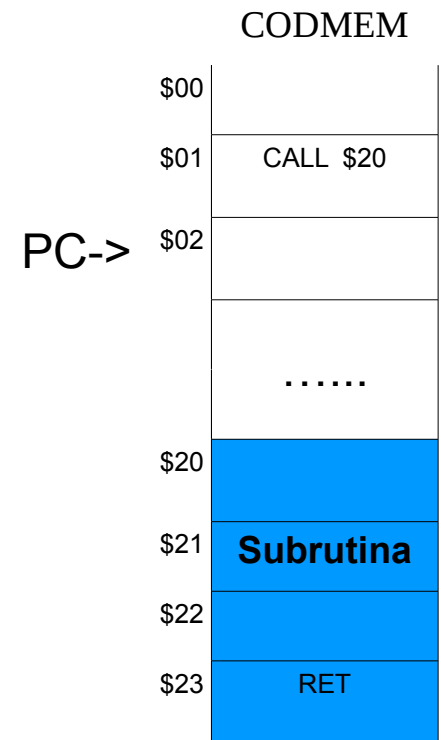
(b)

# Instrucciones de llamada y retorno de subrutinas

- Se ejecutan las instrucciones de la subrutina hasta que se procede a ejecutar la última instrucción o RET (c)
- La ejecución de RET hace que se saque el dato que está en la cima de la pila y que se guarde sobre el PC (d)
- A continuación se sigue ejecutando instrucciones. En este caso a partir de la dirección \$02.
- Observe que \$02 sigue en la PILA pero ya no es accesible y en la próxima operación de PUSH será sobrescrito.



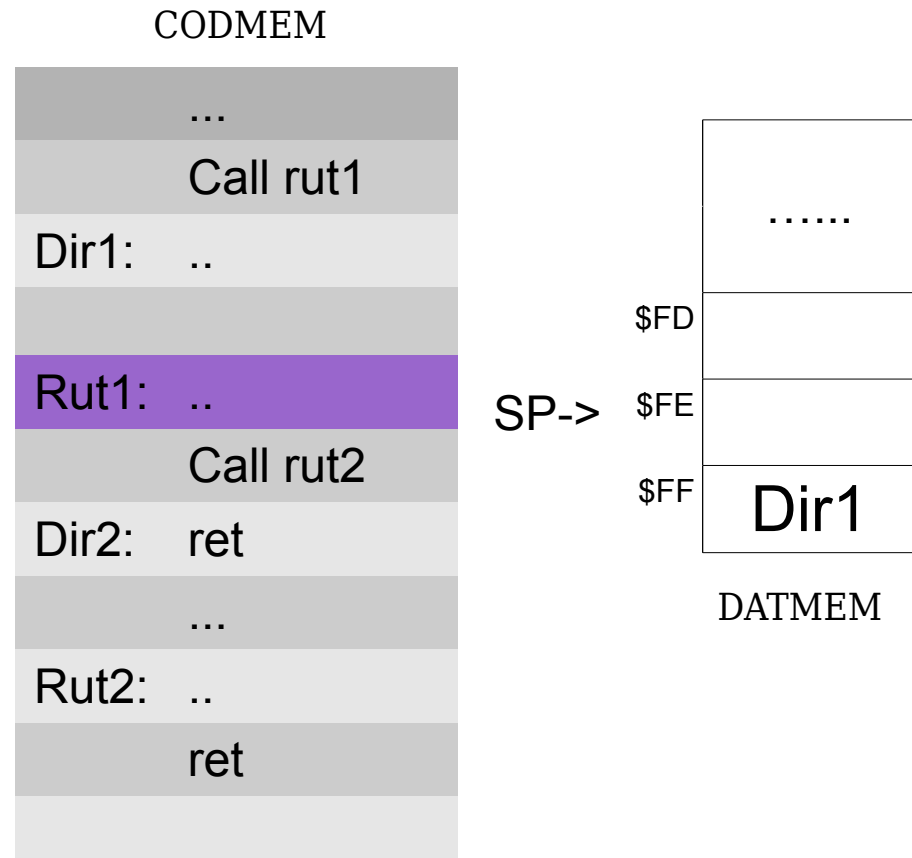
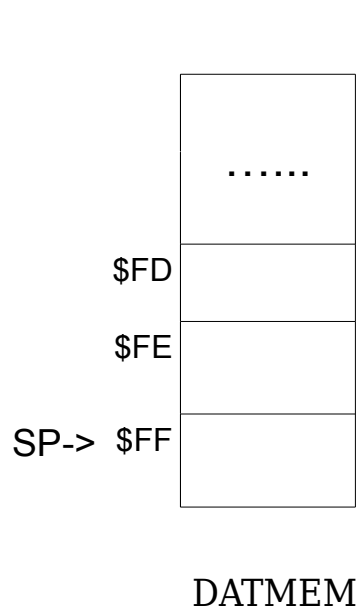
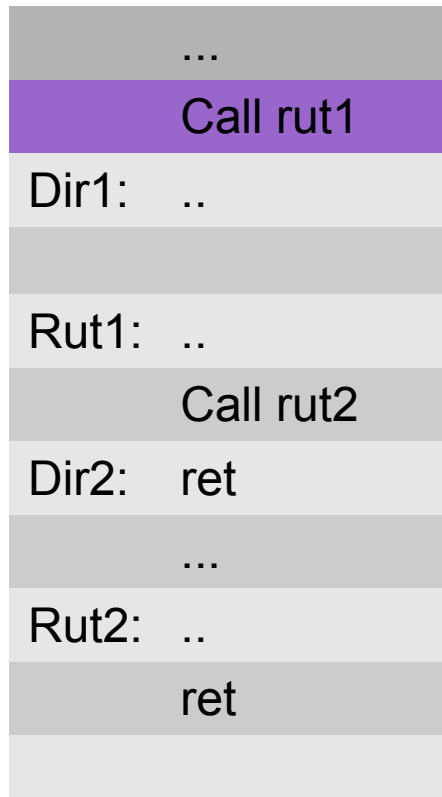
(c)



(d)

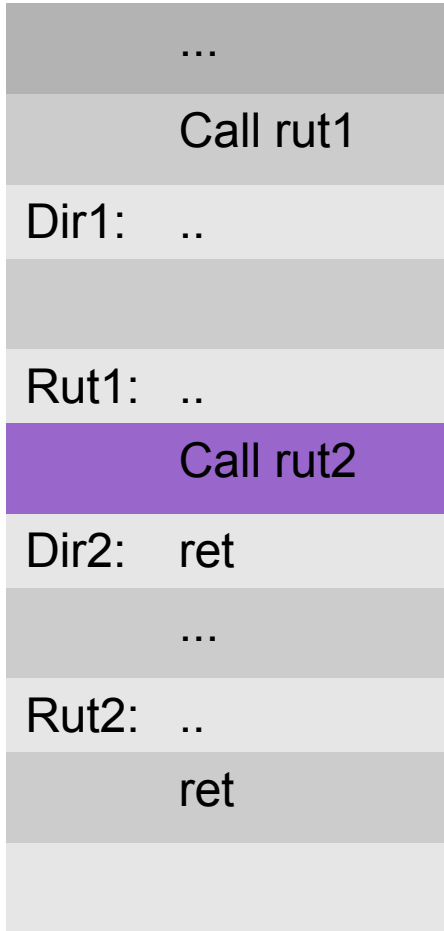
# Instrucciones de llamada y retorno de subrutinas

- Anidamiento de subrutinas



# Instrucciones de llamada y retorno de subrutinas

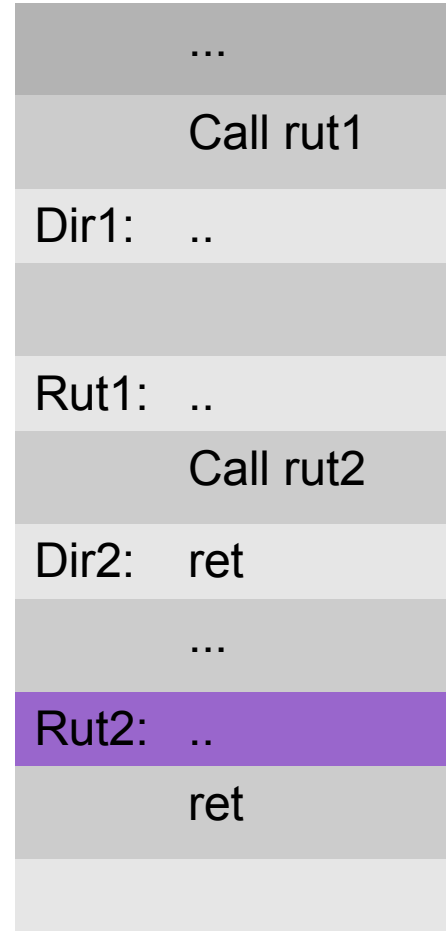
## Anidamiento de subrutinas



SP->



CODMEM

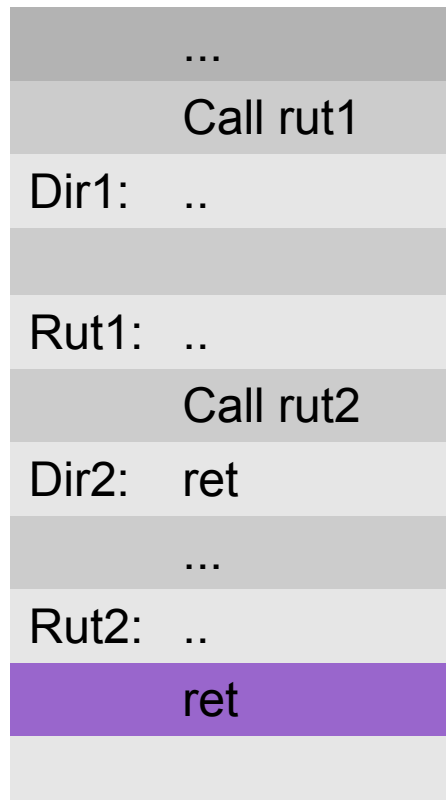


SP->



# Instrucciones de llamada y retorno de subrutinas

## Anidamiento de subrutinas

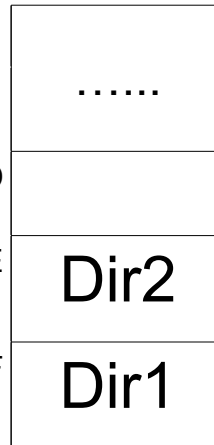


SP->

\$FD

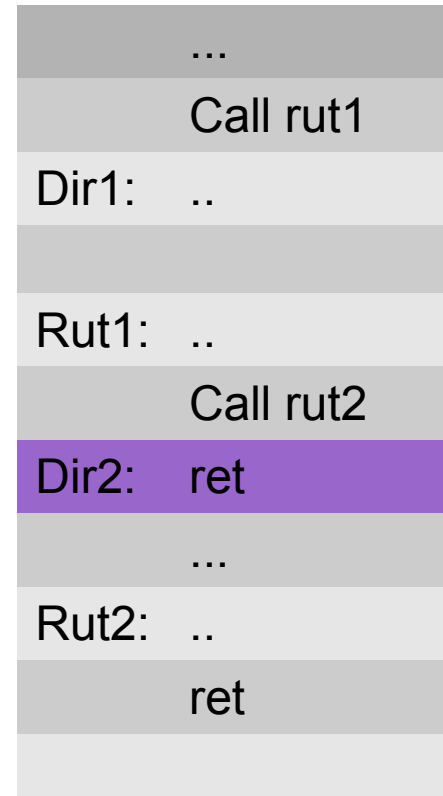
\$FE

\$FF



DATMEM

CODMEM



SP->

\$FD

\$FE

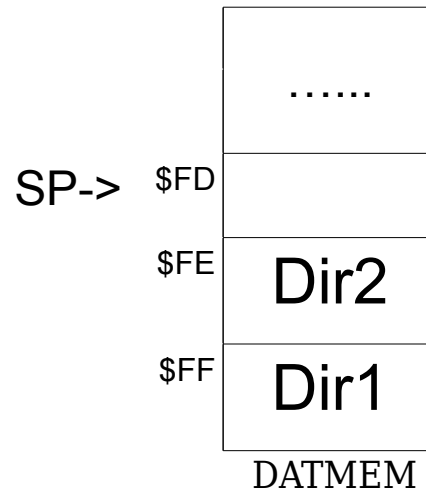
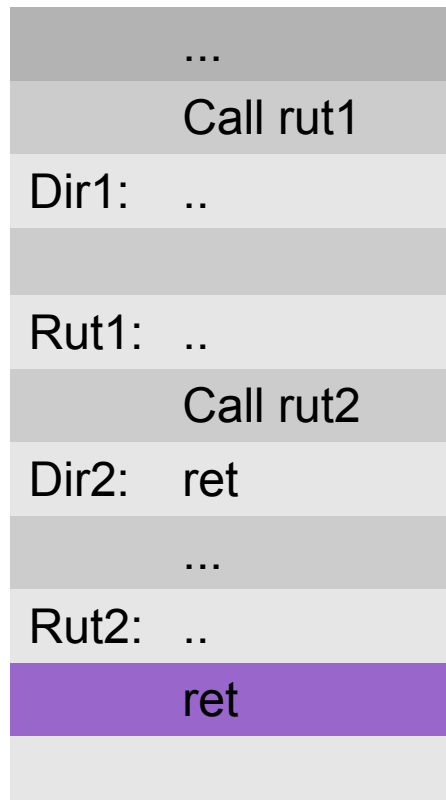
\$FF



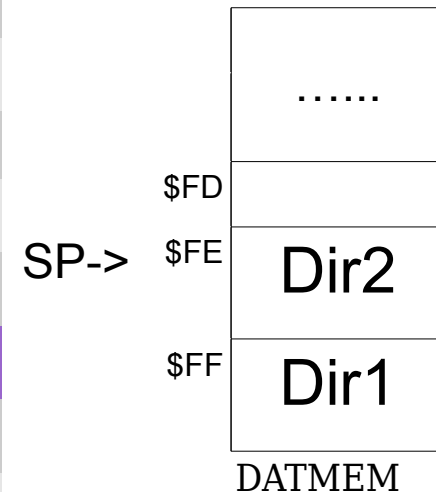
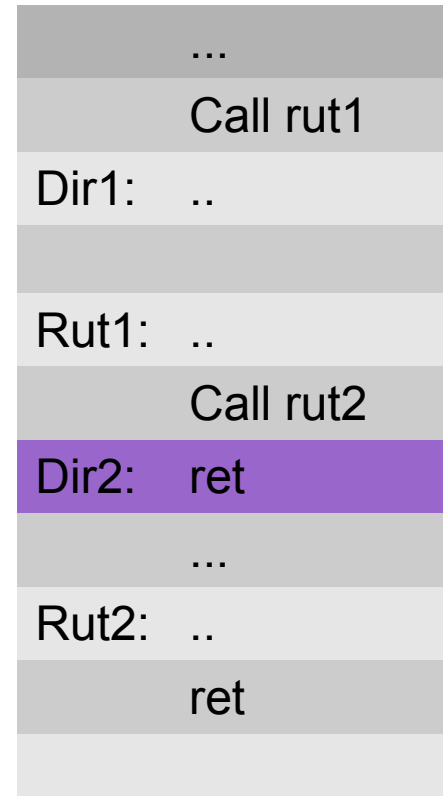
DATMEM

# Instrucciones de llamada y retorno de subrutinas

## Anidamiento de subrutinas



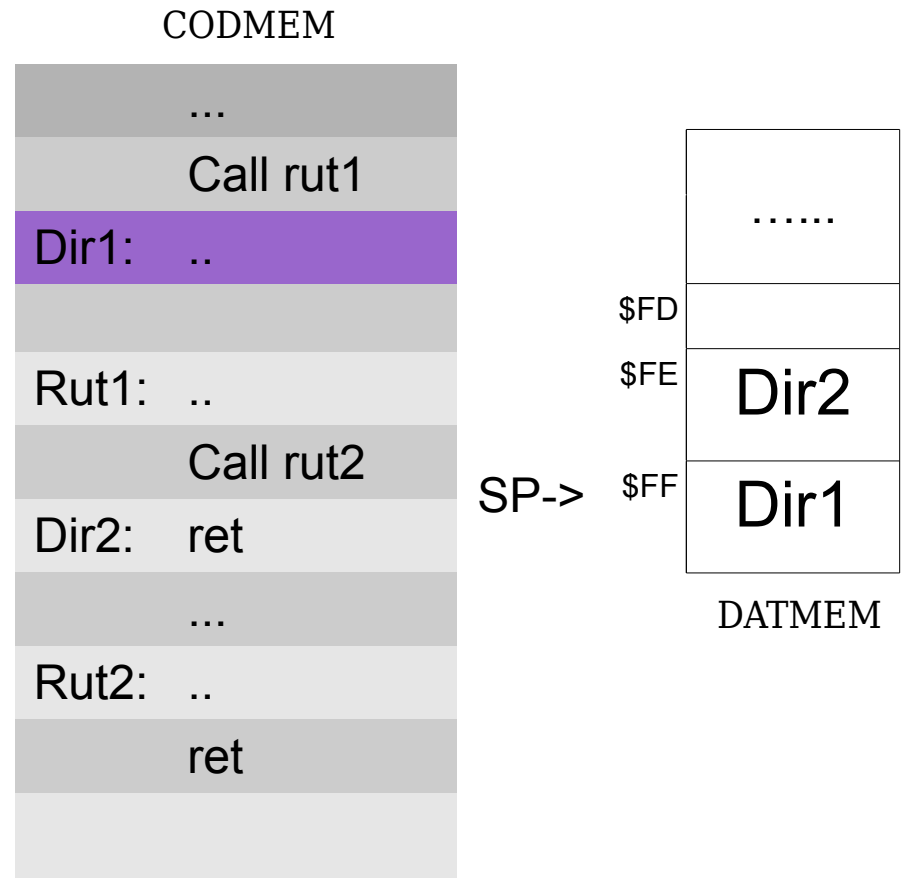
## CODMEM



# Instrucciones de llamada y retorno de subrutinas

## Anidamiento de subrutinas

- Observe que el orden en el que se deben extraer los contenidos del PC de la PILA debe ser en sentido inverso a como éstos se introdujeron en ella. (LIFO)



---

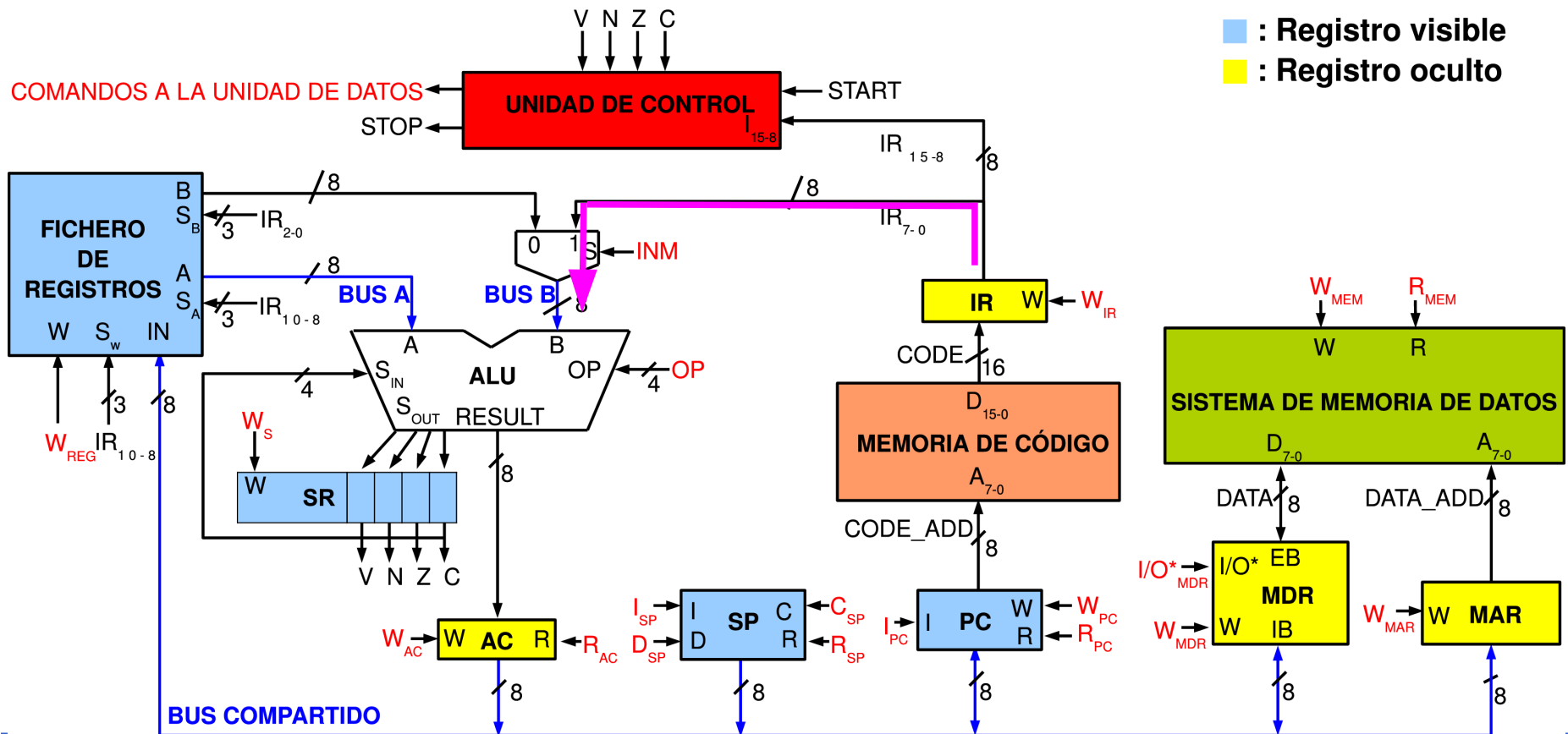
# Modo inmediato

- ▶ El dato es parte del propio código de la instrucción y por tanto, tras finalizar el ciclo de búsqueda, se encontrará en el registro IR.



## Modo inmediato.

Tras el ciclo de búsqueda el dato con el que se va a operar está en el registro de instrucción. Se dispone de un camino a la ALU para operar con el dato dependiendo de la instrucción que utilice este modo.



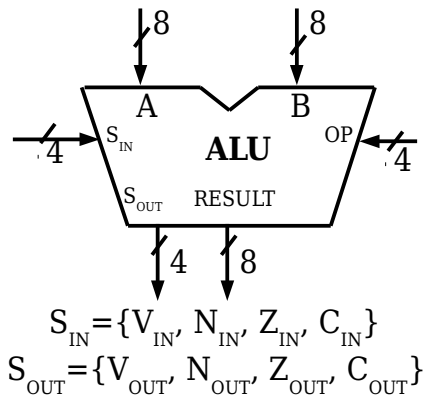
# Códigos de condición de la instrucción de bifurcación condicional: BR $xx$

Los bits  $I_{10}I_9I_8$  codifican la condición de salto  $xx$ .

$I_{10}$	$I_9$	$I_8$	CONDICIÓN	nemónico(s) de la condición	notas
0	0	0	Z	ZS, EQ	será cierta justo tras realizar la resta A-B si y solo si A=B
0	0	1	C	CS, LO	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación base 2 sin signo
0	1	0	V	VS	será cierta si y solo si el dato recién calculado no es representable en notación complemento a 2
0	1	1	N xor V	LT	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación complemento a 2
1	-	-	?	-	estas condiciones no están definidas y no deben utilizarse

BR $ZS$ , BR $EQ$ , BR $CS$ ,BR $LO$ , BR $VS$ ,BR $LT$

# Relación de la ALU con las instrucciones

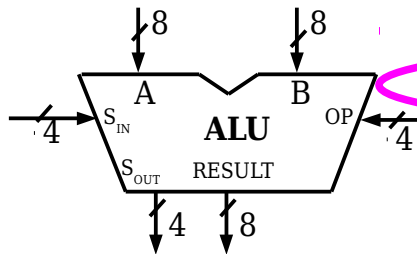


OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	C <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

La ALU de este computador es mucho más compleja que la del CS2 debido a que ha sido la solución adoptada para poder realizar algunas de las nuevas instrucciones propuestas. Esta solución de complicar la ALU no es la única opción.

A continuación justificaremos la complejidad de la ALU en base a las instrucciones añadidas.

# Relación de la ALU con las instrucciones



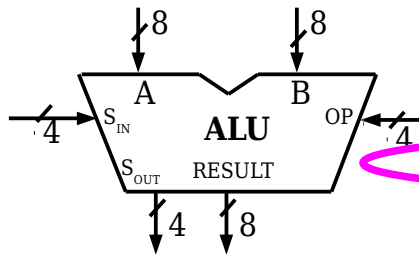
$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$   
 $S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	C <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos estas operaciones?

Instrucciones **CLC** y **SEC**: ponen a 0 y 1 el flag de acarreo C. La ALU no hace nada salvo cambiar este flag y dejar el resto inalterado. Vemos que en la ALU además de entrar los 4 códigos de operación entran los 4 flags.

# Relación de la ALU con las instrucciones



OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	c <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$   
 $S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

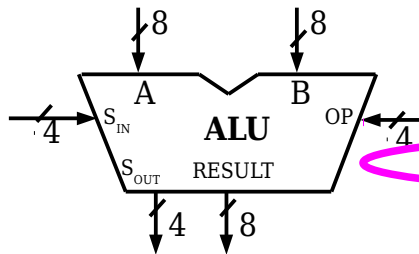
¿Porqué necesitamos estas operaciones?

Instrucciones **ROR Rd** y **ROL Rd**: Como vemos la ALU rota el dato que entra por su entrada A usando la entrada de acarreo como bit de entrada serie. El resultado de la rotación aparece en la salida RESULT.

¿Qué información nos proporcionan los flags?

**V**: La operación EXOR de C<sub>in</sub> con A<sub>0</sub> en el caso de desplazamiento a la derecha, y de A<sub>7</sub> con A<sub>6</sub> en el caso de desplazamiento a la izquierda. En este último se nos informa de un cambio de signo.

# Relación de la ALU con las instrucciones



$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$   
 $S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	C <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos estas operaciones?

Instrucciones **ROR Rd** y **ROL Rd**

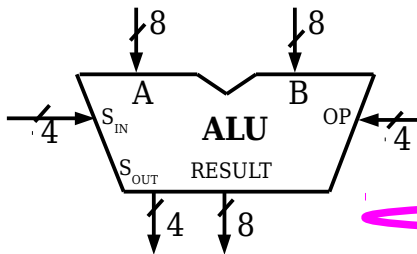
¿Qué información nos proporcionan los flags?

**N:** Bit más significativo de los números, nos informa del signo.

**Z:** Operación NOR de todos los bits del resultado. Vale 1 si el resultado es =0.

**C:** Bit que se desplaza.

# Relación de la ALU con las instrucciones



$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$   
 $S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

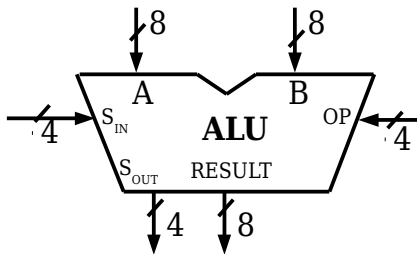
OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	C <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos esta operación?

Instrucciones **ST (Rb),Rf** y **STS dirección,Rf**: Lo que entra por A lo direccionan los bits IR<sub>10-8</sub>:  
fuente en ST (mirar u.datos)

<u>formato</u>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>A</b> instrucción con operando registro	código de operación					registro destino (fuente en ST)			-	-	-	-	-	registro fuente (registro base en ST/LD)		
<b>B</b> instrucción con operando memoria o inmediato									dato inmediato / dirección del dato							
<b>C</b> instrucción de salto						condición de salto										

# Relación de la ALU con las instrucciones



OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	c <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$

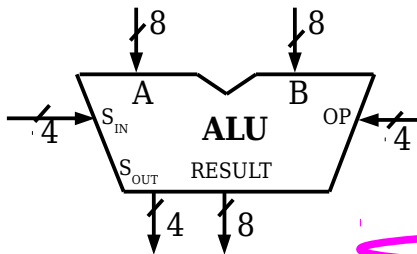
$S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

¿Porqué necesitamos esta operación?

Instrucciones **ST (Rb),Rf**; **STS dirección,Rf**; **LD Rd, (Rb)**; **LDS Rd,dirección**; **CALL dirección**, **BRxx dirección**; **JMP dirección** y **LDI Rd,dato** y **MOV Rd,Rf**: necesitan que un dato pase por la entrada B sin alterarse (mirar u. datos).

formato	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>A</b> instrucción con operando registro	código de operación					registro destino (fuente en ST)			-	-	registro fuente (registro base en ST/LD)					
<b>B</b> instrucción con operando memoria o inmediato									dato inmediato / dirección del dato							
<b>C</b> instrucción de salto						condición de salto			dirección de salto							

# Relación de la ALU con las instrucciones



$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$   
 $S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	c <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos esta operación?

Instrucciones **ADD Rd,Rf** y **ADDI Rd,dato**: requieren operación de suma

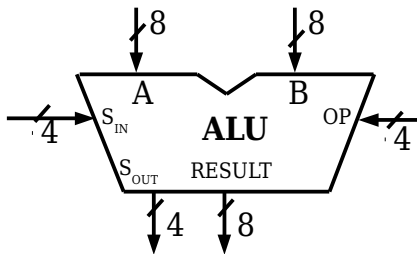
**V**: desbordamiento en suma de números con signo.

**N**: MSB del resultado

**Z**: NOR del resultado indica si es cero

**C**: acarreo

# Relación de la ALU con las instrucciones



$S_{IN} = \{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$   
 $S_{OUT} = \{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP <sub>3</sub>	OP <sub>2</sub>	OP <sub>1</sub>	OP <sub>0</sub>	RESULT=	V <sub>OUT</sub> =	N <sub>OUT</sub> =	Z <sub>OUT</sub> =	C <sub>OUT</sub> =
0	0	-	0	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V <sub>IN</sub>	N <sub>IN</sub>	Z <sub>IN</sub>	1
0	1	0	0	SHR(A, C <sub>IN</sub> )	c <sub>IN</sub> EXOR A <sub>0</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>0</sub>
0	1	0	1	SHL(A, C <sub>IN</sub> )	A <sub>7</sub> EXOR A <sub>6</sub>	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	A <sub>7</sub>
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2 <sup>8</sup>	overflow(A+B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	carry(A+B)
1	0	1	-	(A - B) mod 2 <sup>8</sup>	underflow(A-B)	RESULT <sub>7</sub>	NOT OR <sub>i=0</sub> <sup>7</sup> (RESULT <sub>i</sub> )	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos esta operación?

Instrucciones ***SUB Rd,Rf***; ***SUBI Rd,Rf***; ***CP Rd,Rf*** y ***CPI Rd,dato***: requieren operación de resta

**V**: desbordamiento en suma de números con signo.

**N**: MSB del resultado

**Z**: NOR del resultado indica si es cero

**C**: borrow

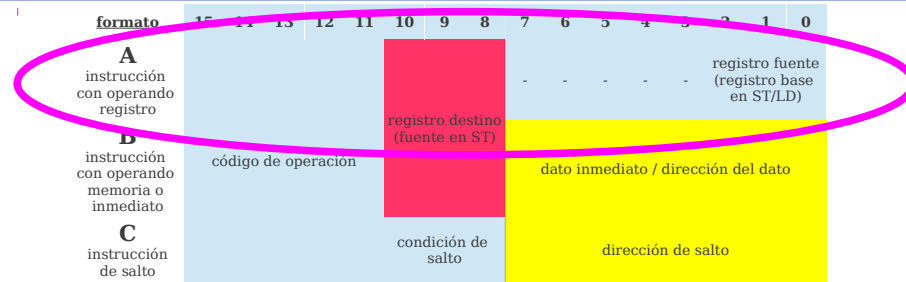
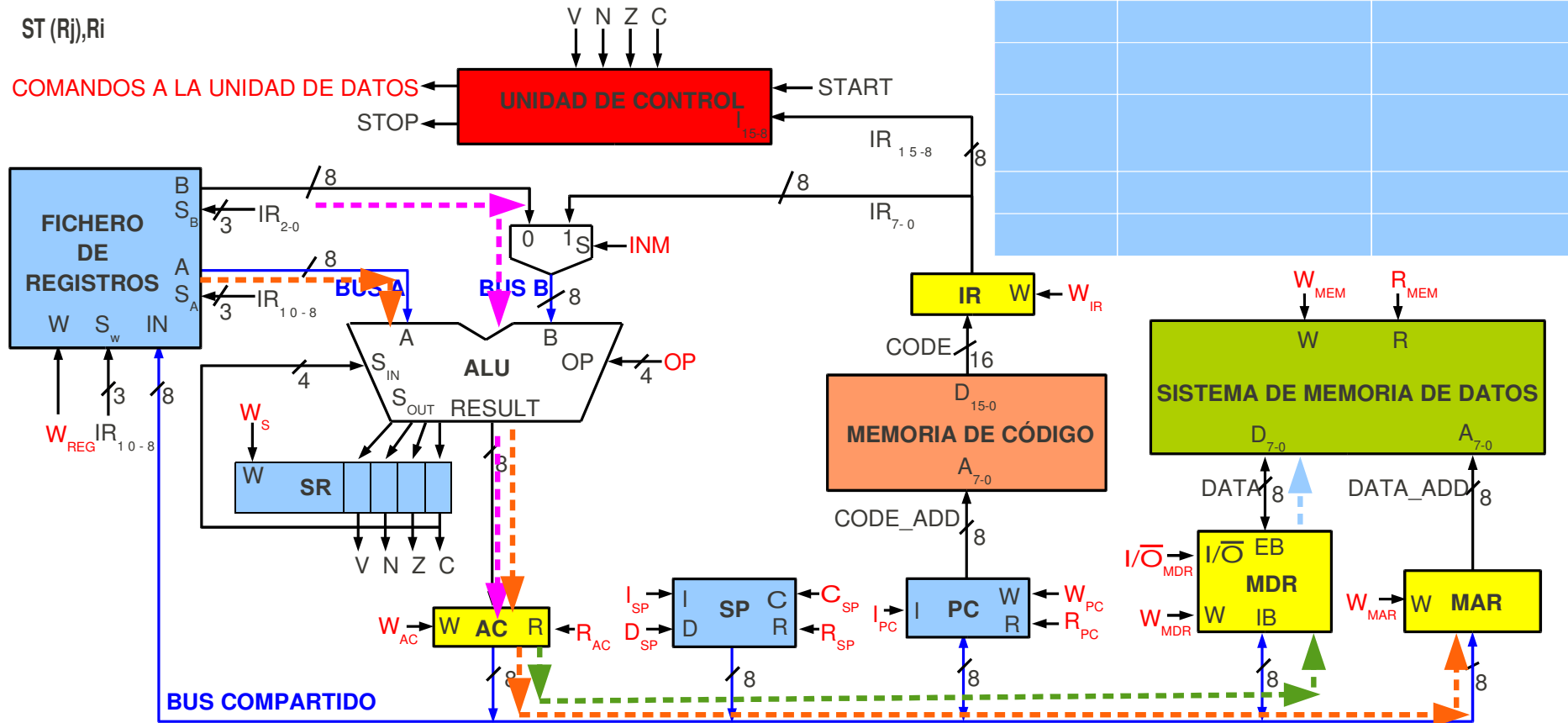
---

# **Descomposición de las instrucciones en microoperaciones para algunos casos**

## Ejemplo de instrucción de acceso a memoria Direccionamiento indirecto de registro

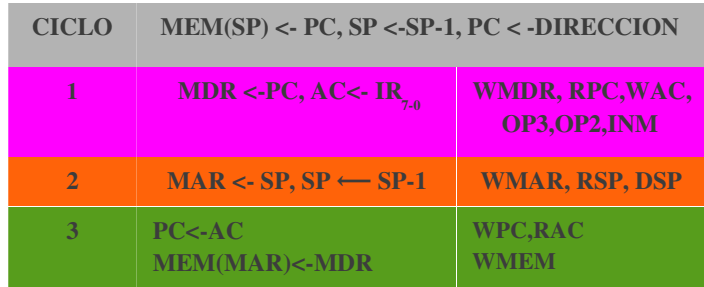
ST (Rj),Ri

COMANDOS A LA UNIDAD DE DATOS



condición de salto	dirección de salto
--------------------	--------------------

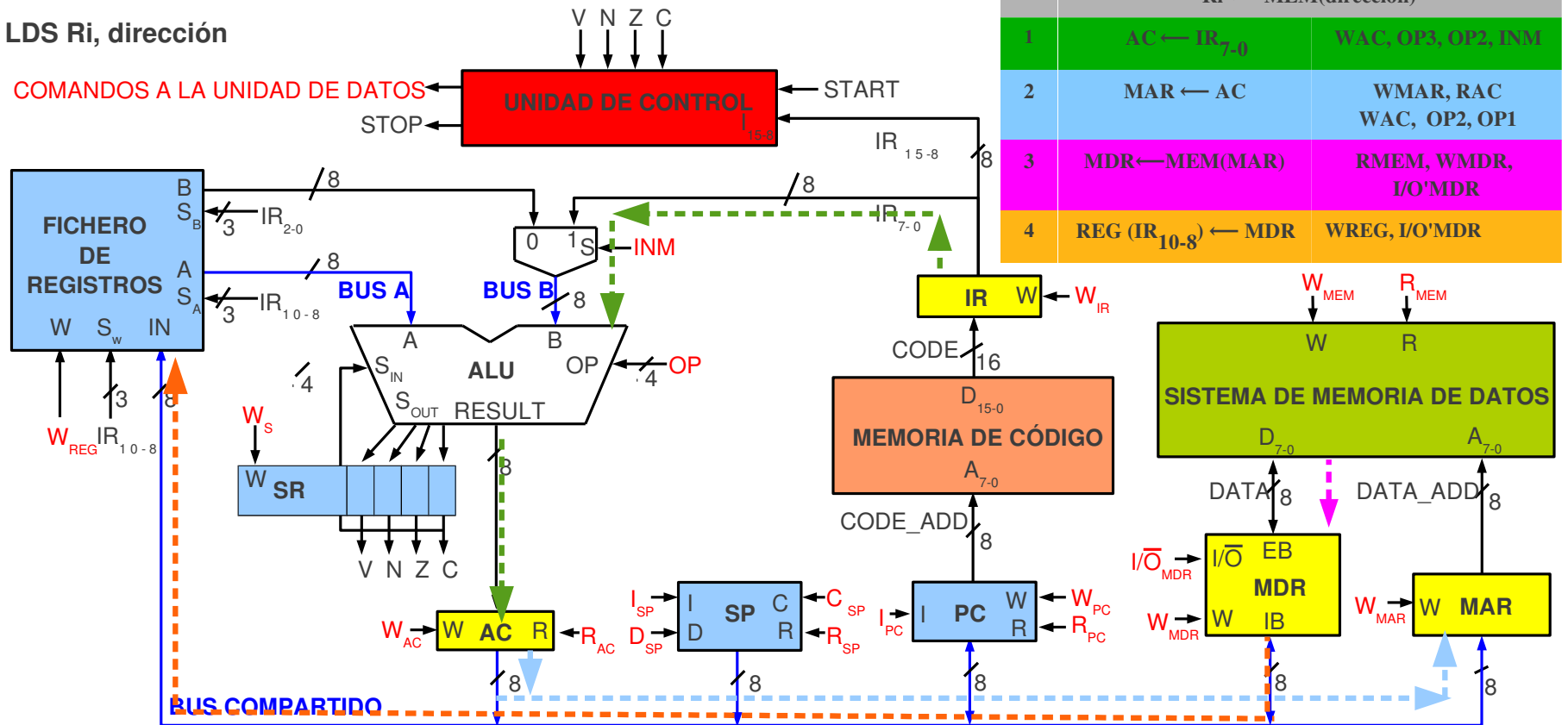
## COMANDOS A LA UNIDAD DE DATOS



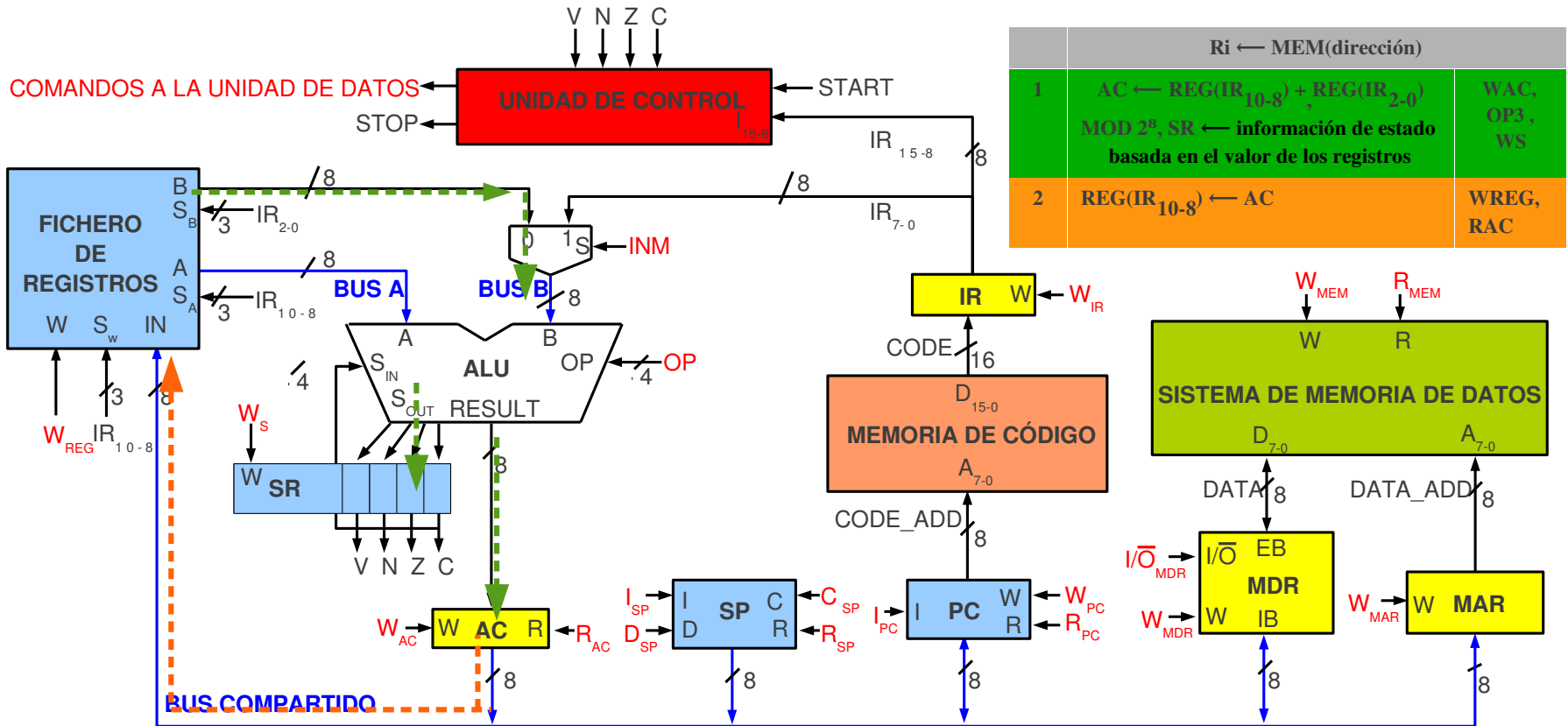
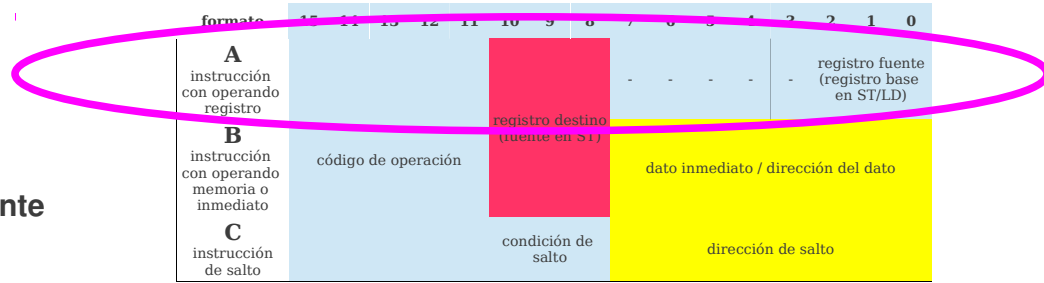
## Ejemplo de instrucción de transferencia desde la memoria con modo de direccionamiento absoluto

LDS Ri, dirección

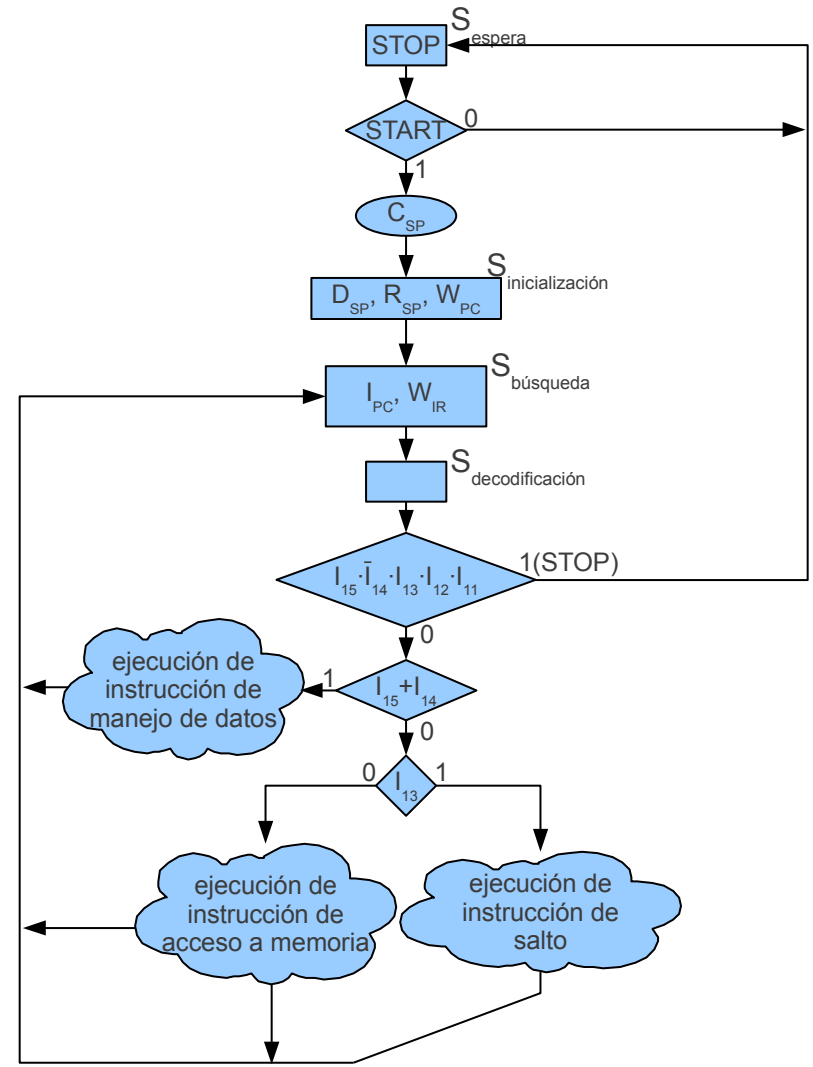
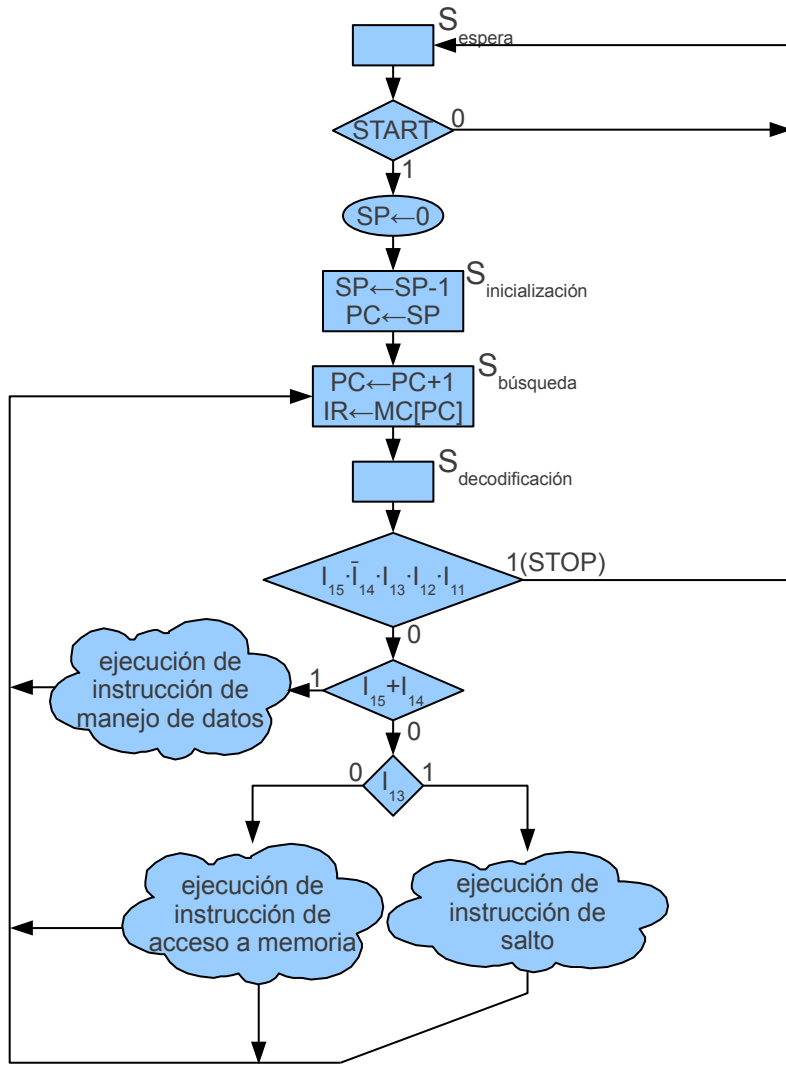
COMANDOS A LA UNIDAD DE DATOS



**Ejemplo de instrucción aritmética que usa exclusivamente direccionamiento directo de registro**  
**ADD Ri, Rj**



# Cartas ASM



---

NOTA: La descripción RT del resto de las instrucciones se encuentra en un documento anexo a este. El alumno debe trabajar con ese documento.

Al final de este mismo documento se encuentran las cartas detalladas correspondientes a las descripciones RT de todas las instrucciones.

# Ejemplos de uso

- Escriba una subrutina para el cálculo de la multiplicación mediante el algoritmo de sumas sucesivas.

```
MULT:      LDI R0,0
           CPI R2,0
           BRZS RETORNA
BUCLE:     ADD R0,R1
           SUBI R2,1
           BRZS RETORNA
           JMP BUCLE
RETORNA:   RET
```

# Ejemplos de uso

- **Escriba una subrutina que devuelva el mayor de dos números escritos en complemento a 2.**

```
MAX:      CP R1,R2
           BRLT R1MENOR
           MOV R0,R1
           RET
R1MENOR:  MOV R0,R2
           RET
```

# Ejemplos de uso

- ▶ **Desarrolle una subrutina que escriba, de forma descendente, los números del 100 al 1 en una tabla situada a partir de la posición de memoria 123.**

```
CUENTA:    LDI R0,100
            LDI R1,123
BUCLE:     ST (R1),R0
            ADDI R1,1
            SUBI R0,1
            BRZS RETORNA
            JMP BUCLE
RETORNA:   RET
```

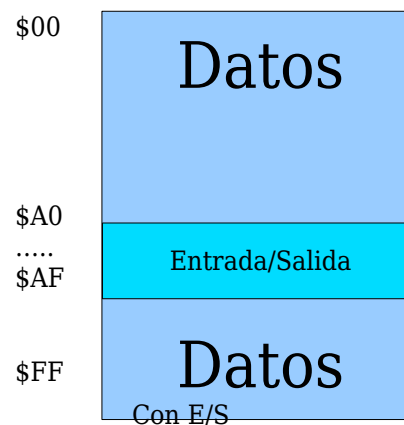
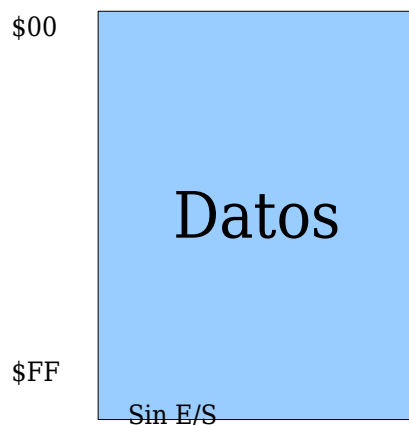
---

# Entrada/Salida

- ▶ El CS 2010 no ha permitido solucionar unas de las deficiencias del CS2 planteadas al principio de este tema: la incapacidad de comunicación con el exterior.
- ▶ Dicho de otra forma: no posee capacidad de entrada salida (E/S).
- ▶ A continuación se expone un procedimiento que permitiría dotarlo de dicha capacidad.

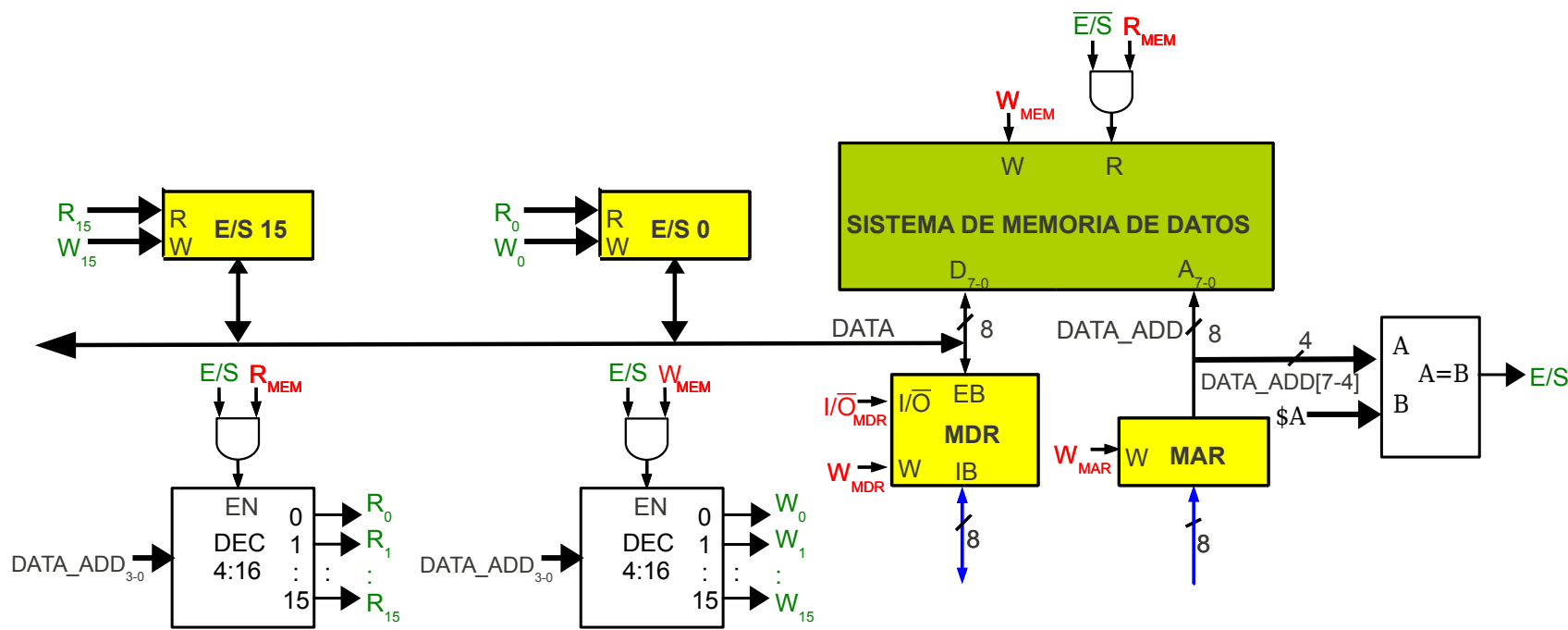
# Entrada/Salida

- Desde el punto de vista del programador la memoria es vista como un conjunto de posiciones de memoria; cada una de las cuales tiene asociada una dirección única.
- Parte del espacio de direccionamiento puede reservarse para que el procesador pueda acceder al exterior (entrada/salida mapeada en memoria).



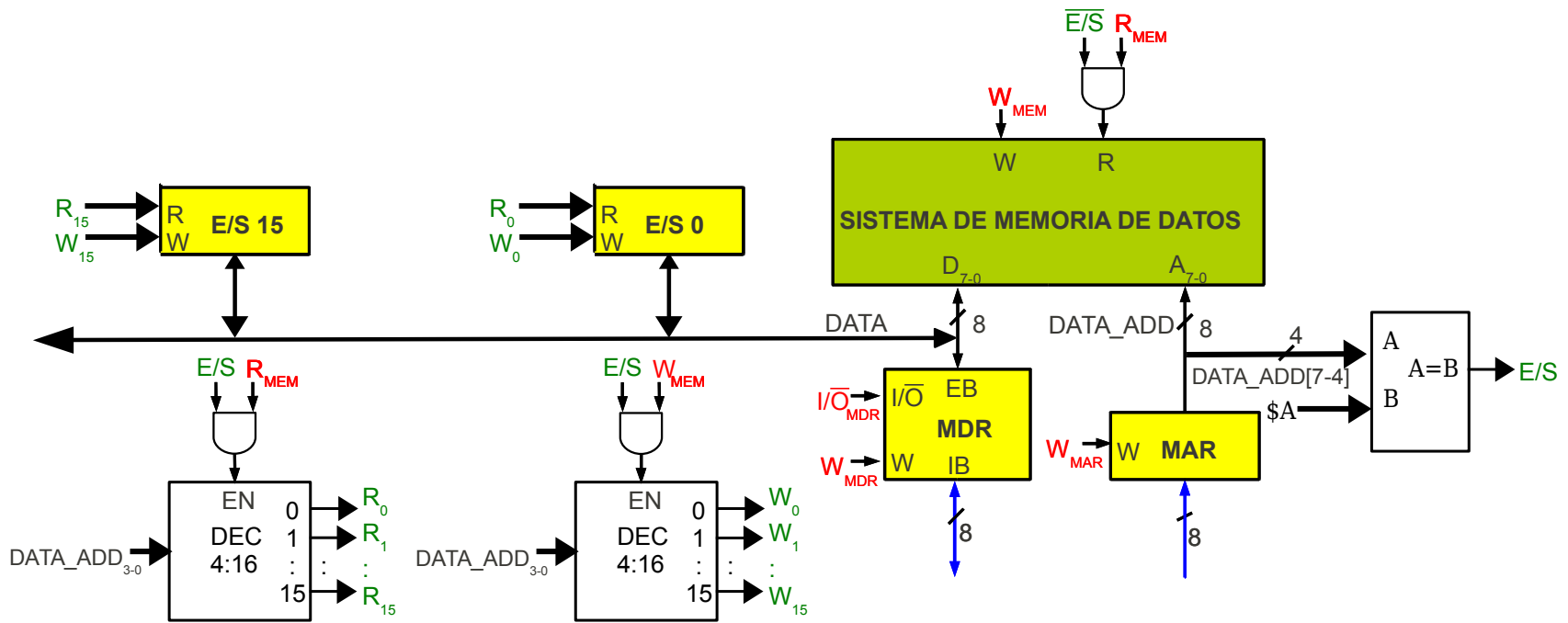
# Entrada/Salida

- Reservaremos un subrango del espacio de direccionamiento (ej: \$A0-\$AF) para entrada/salida.
- Para ese subrango no se accede a la memoria de datos sino a los dispositivos de entrada/salida.



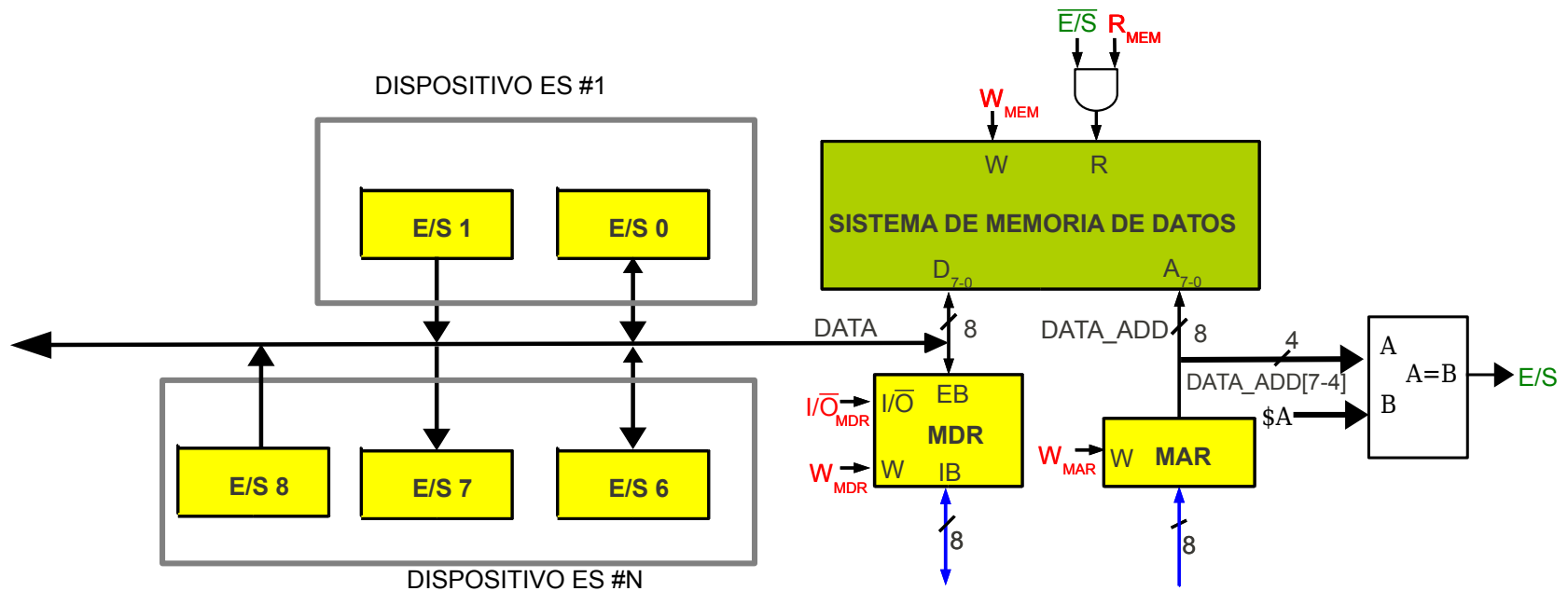
# Entrada/Salida

- Los cuatro bits más significativos de DATA\_ADD indican si la dirección pertenece al rango de entrada/salida .
- Para dicho rango, los bits restantes de DATA\_ADD se utilizarán para seleccionar la posición de E/S concreta.



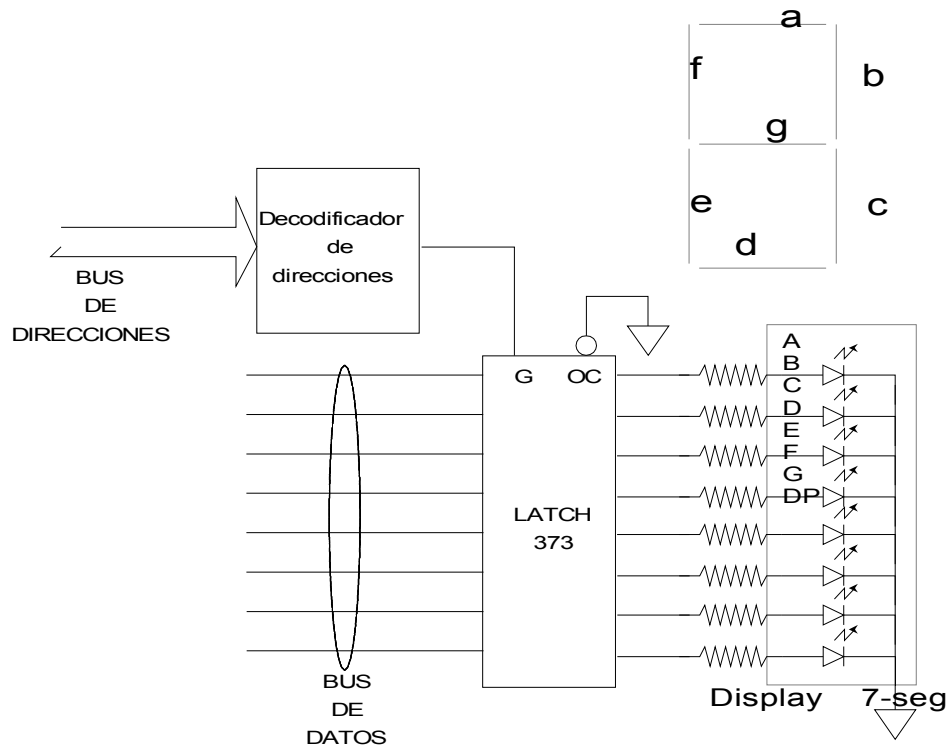
# Entrada/Salida

- Las posiciones de E/S puede ser de lectura/escritura, de sólo lectura o de solo escritura
- Un dispositivo de E/S puede agrupar varias posiciones de E/S.



# Entrada/Salida

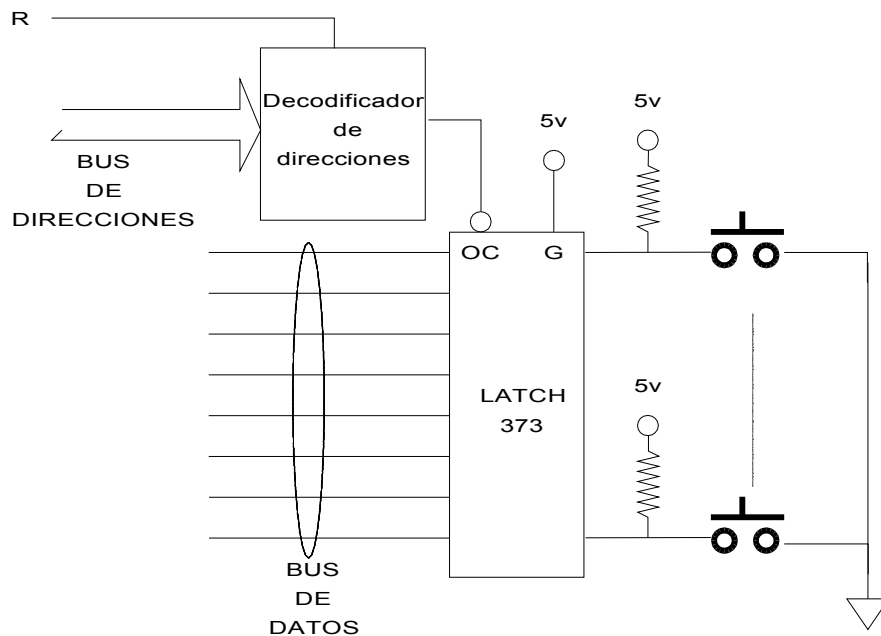
- Dispositivo de salida (display de 7 segmentos)



```
LDI R1,$FF /*Código 7seg del número 8*/  
STS $A0,R1 /*Muestra el número 8*/
```

# Entrada/Salida

- Dispositivo de entrada (pulsadores)



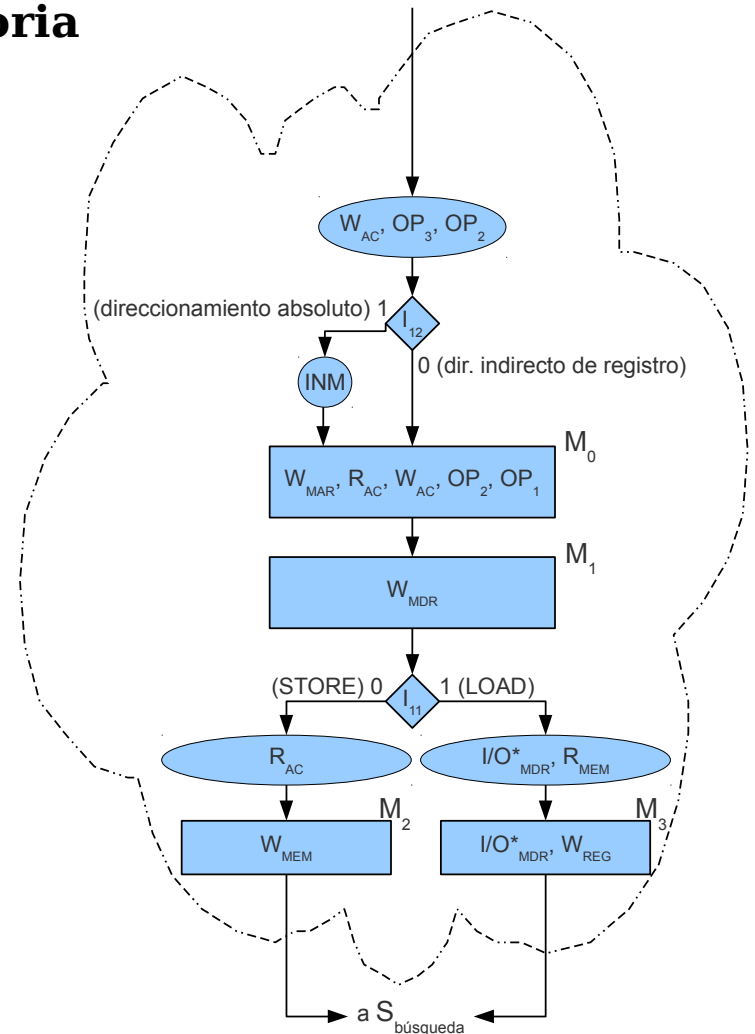
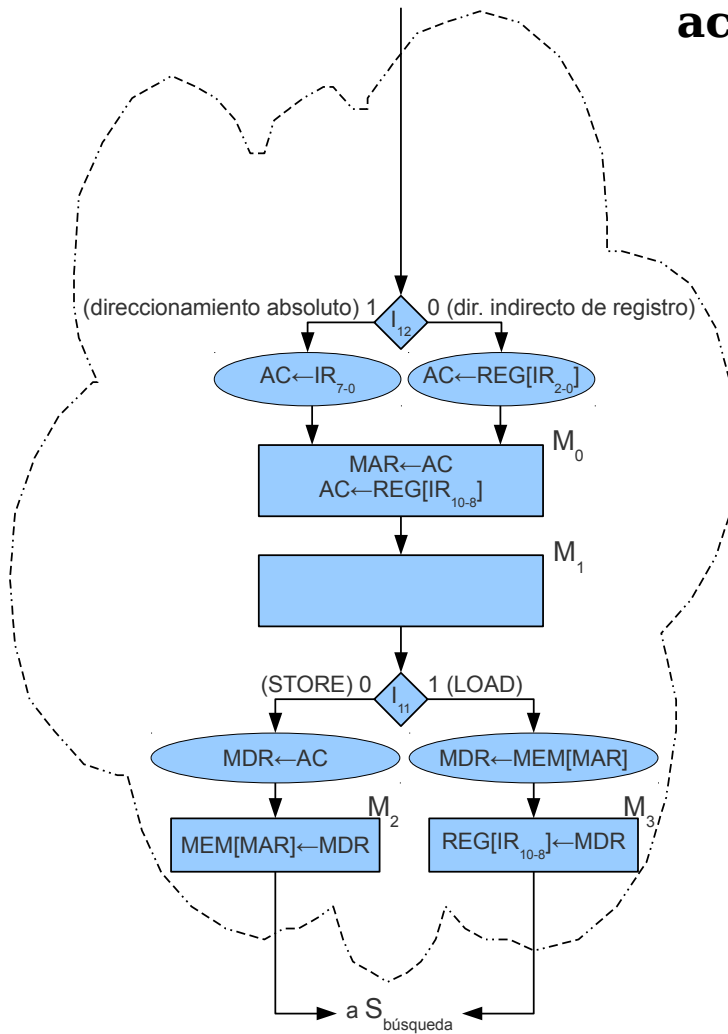
LDS R1, \$A1 /\*Almacena el estado de los pulsadores\*/

---

# ANEXO: CARTAS ASM

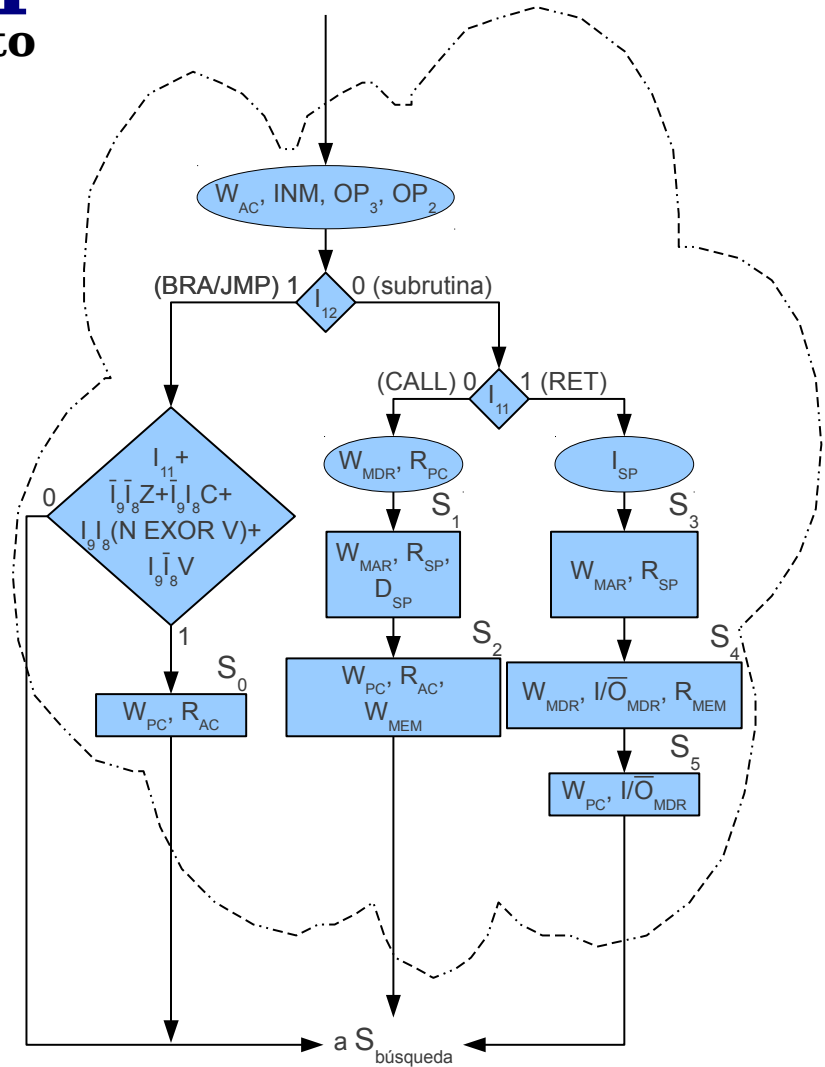
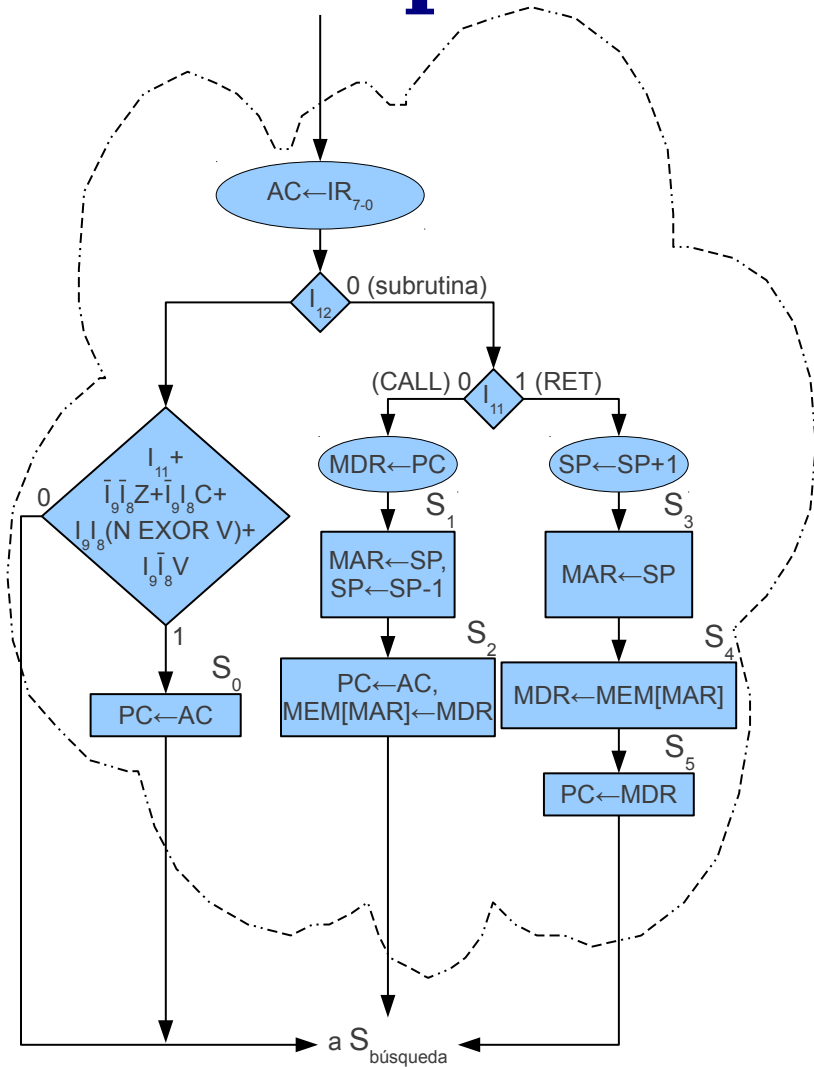
# Una posible implementación

## acceso a memoria



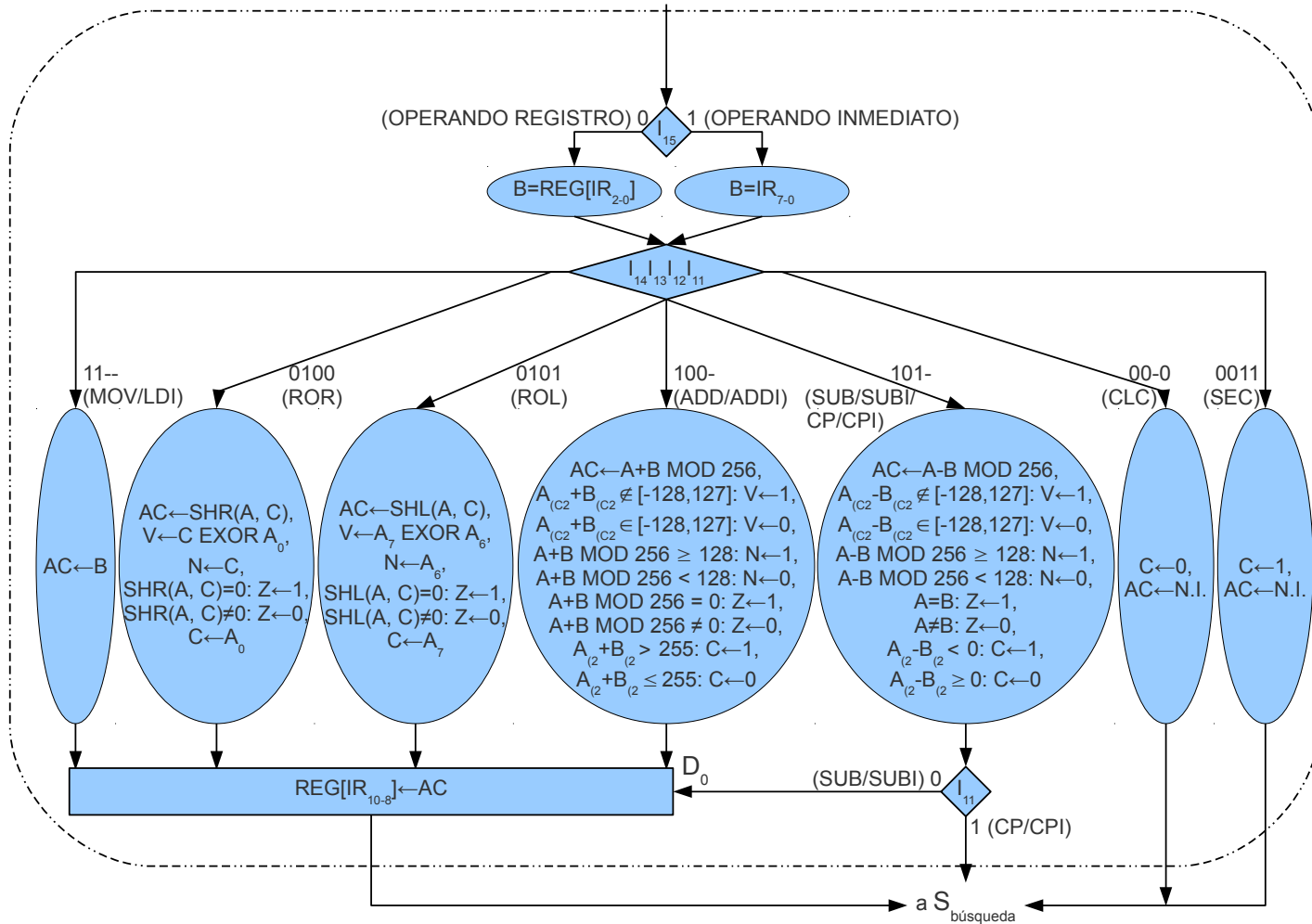
# Una posible implementación

salto



# Una posible implementación

## manejo de datos (trozo de la carta de datos)



# Una posible implementación

## manejo de datos (trozo de la carta de control)

