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.

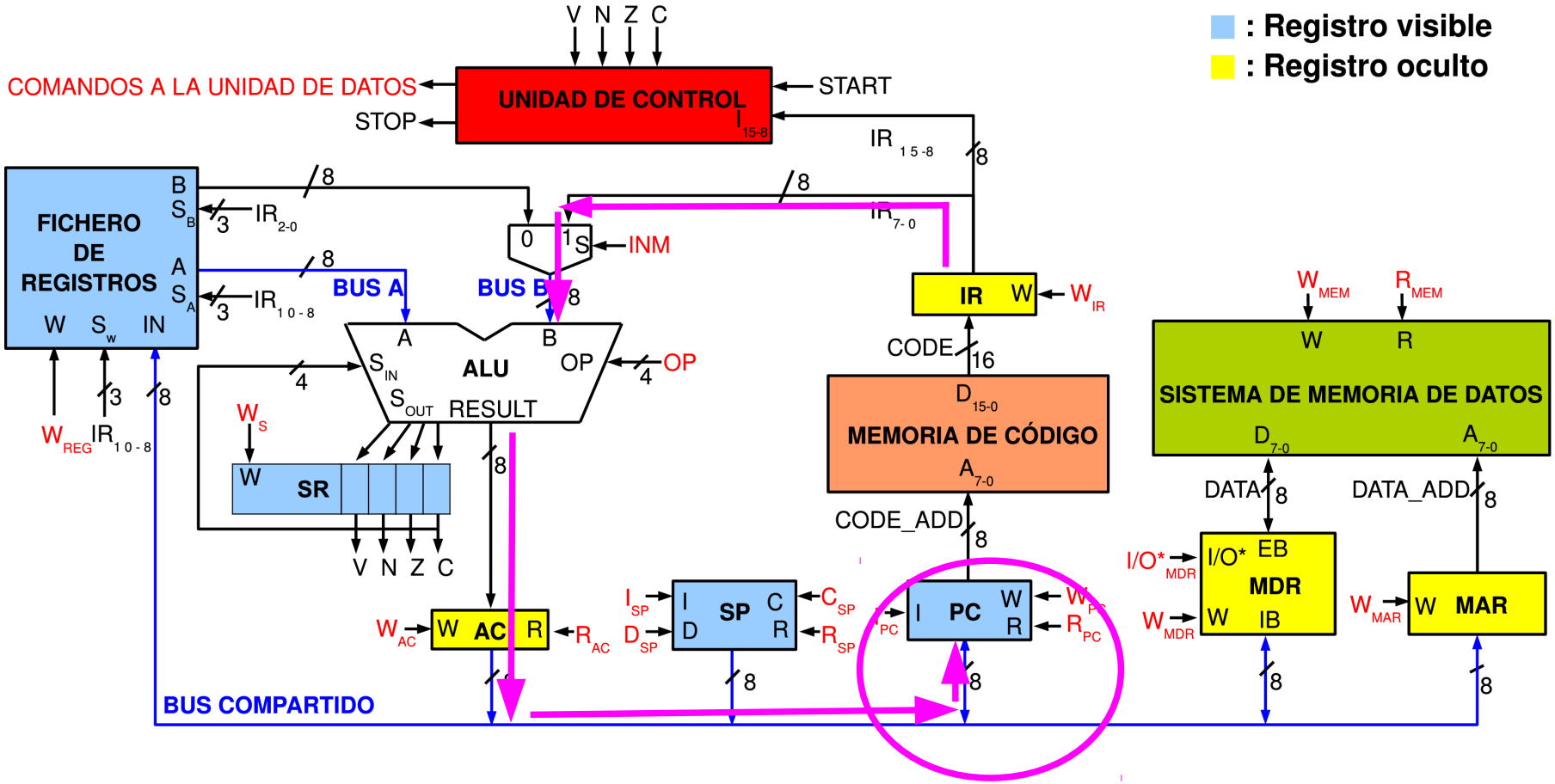
Diversificación de instrucciones

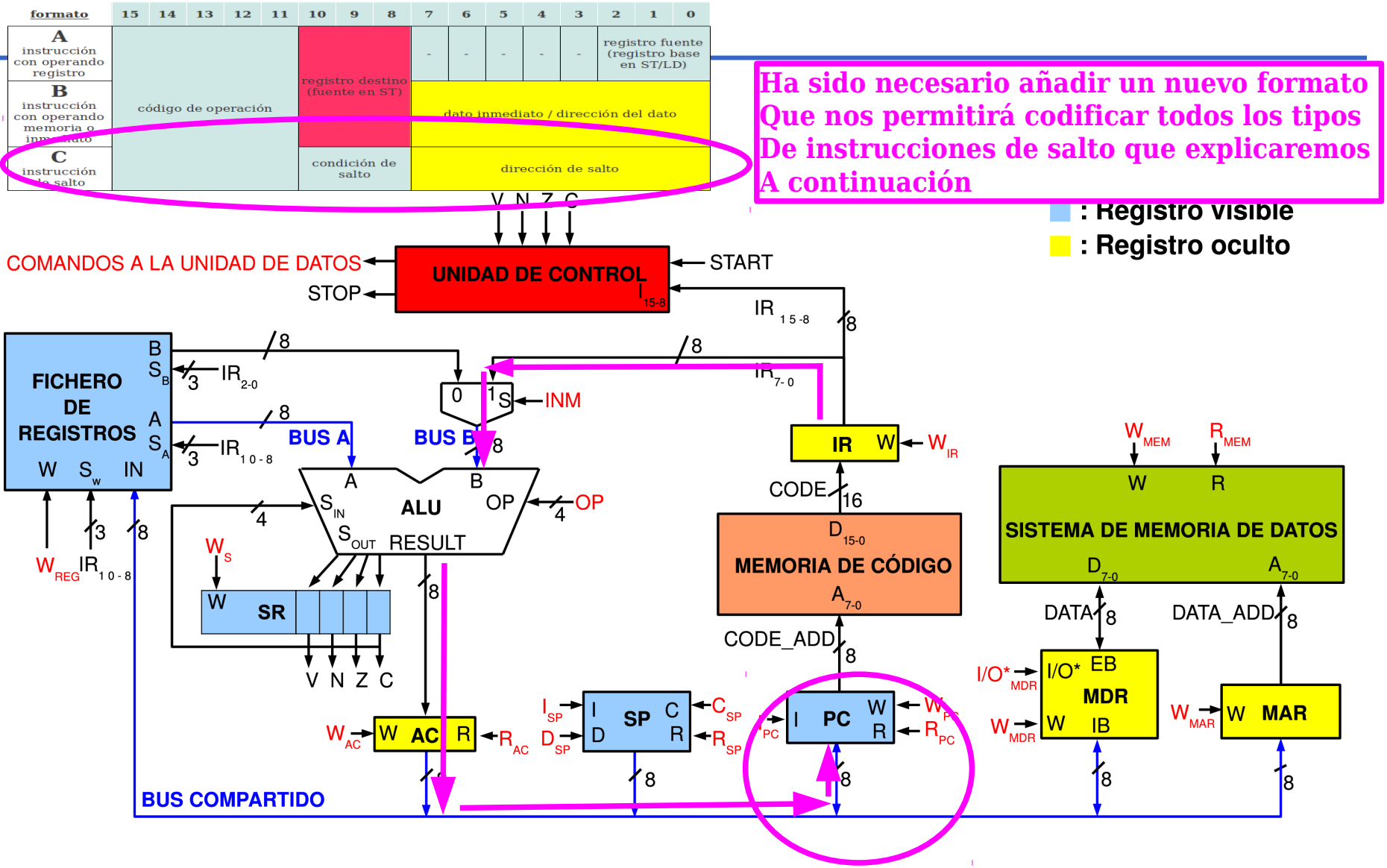
Modelo de usuario: ISP

- 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.

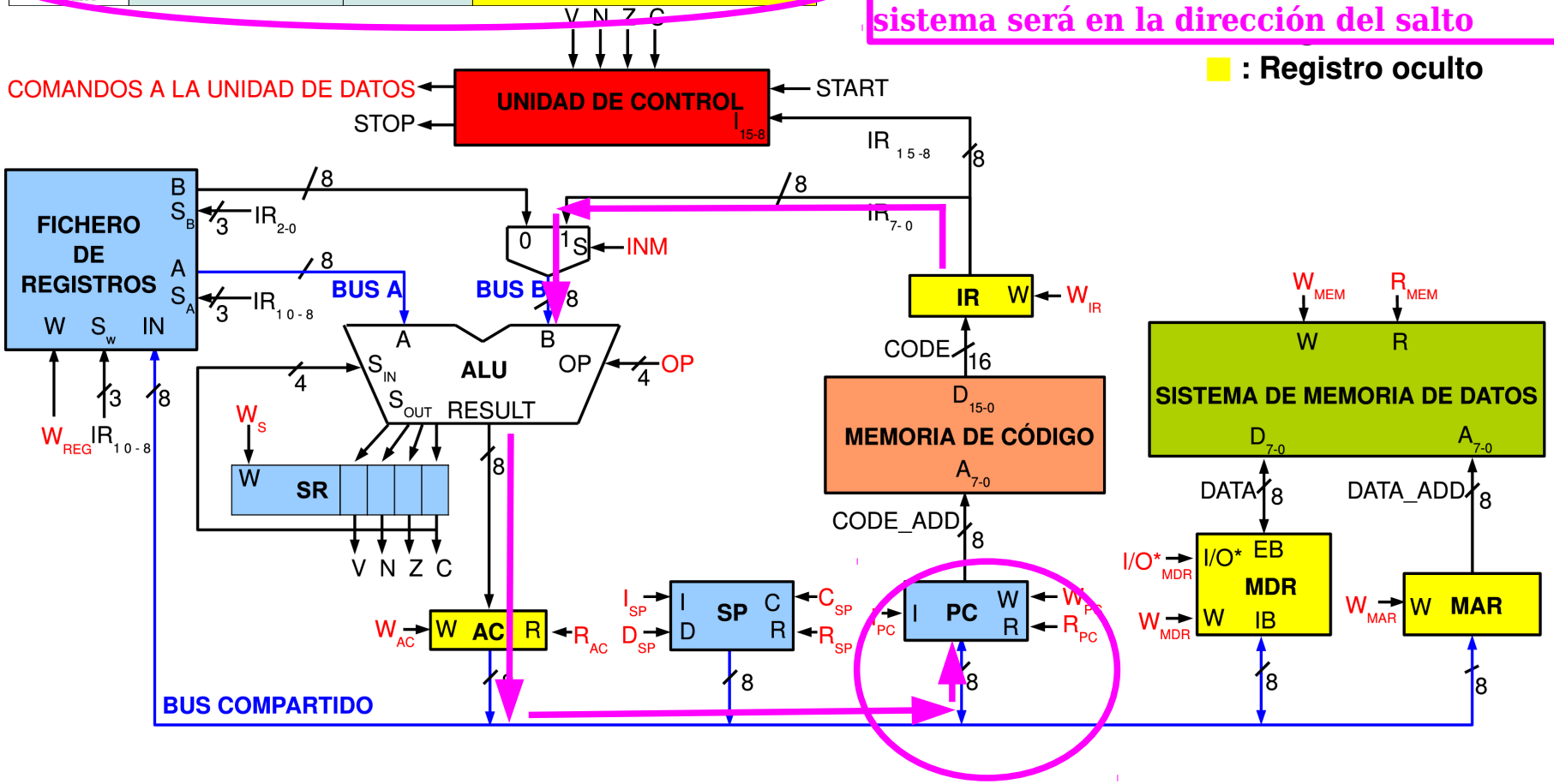
Modificaciones en la arquitectura y el formato de instrucción que nos permiten añadir instrucciones de salto

¿Cómo se hace el salto?: en el PC, ya no cambia siempre de forma secuencial, 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 guardar este dato en el PC. Necesitamos conectar el PC al bus para establecer un camino desde IR

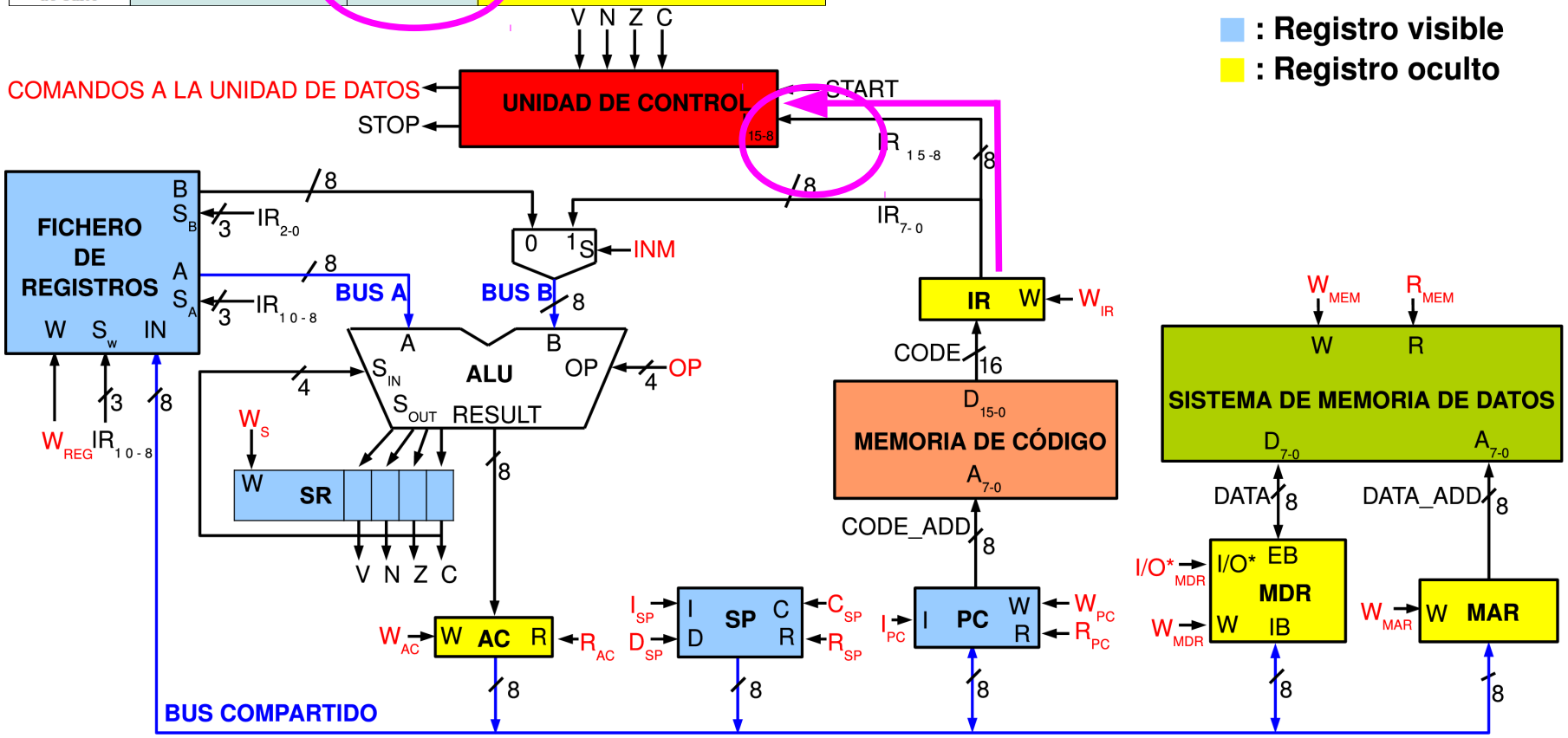


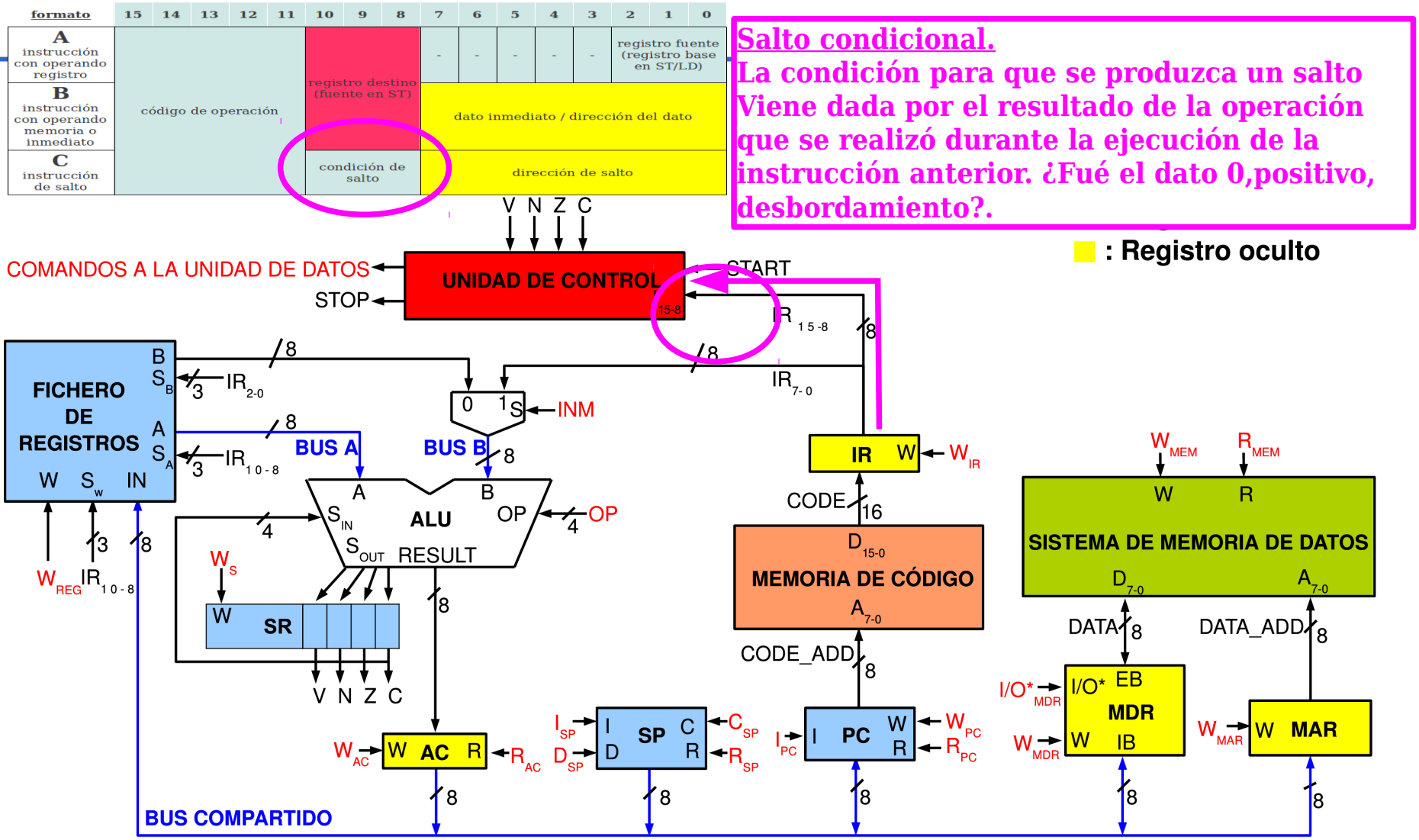

formato
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
A instrucción con operando registro
código de operación
registro destino (fuente en ST)
- - - - - registro fuente (registro base en ST/LD)
B instrucción con operando memoria o inmediato
dato inmediato / dirección del dato
C instrucción de salto
condición de salto
dirección de salto
Ha sido necesario añadir un nuevo formato
Que nos permitirá codificar todos los tipos
De instrucciones de salto que explicaremos
A continuación
: Registro visible
: Registro oculto
V N Z C
COMANDOS A LA UNIDAD DE DATOS
UNIDAD DE CONTROL
I 15-8
START
STOP
IR 15-8
8
FICHERO DE REGISTROS
B S B
A S A
W S w IN
3
IR 2-0
IR 10-8
W REG
IR 10-8
8
BUS A
BUS B
0 1 S
INM
IR 7-0
IR W
W IR
CODE
16
A
B
ALU
S IN
S OUT
RESULT
OP
4
W S
W SR
V N Z C
D 15-0
MEMORIA DE CÓDIGO
A 7-0
CODE_ADD
W MEM
R MEM
W R
SISTEMA DE MEMORIA DE DATOS
D 7-0
A 7-0
DATA
DATA_ADD
I/O*
MDR
EB
I/O*
MDR
W IB
W MDR
W MAR
W MAR
W AC R
W AC
R AC
I SP C
D SP R
SP
C SP
R SP
I SP
D SP
I PC W
PC R
W PC
R PC
BUS COMPARTIDO

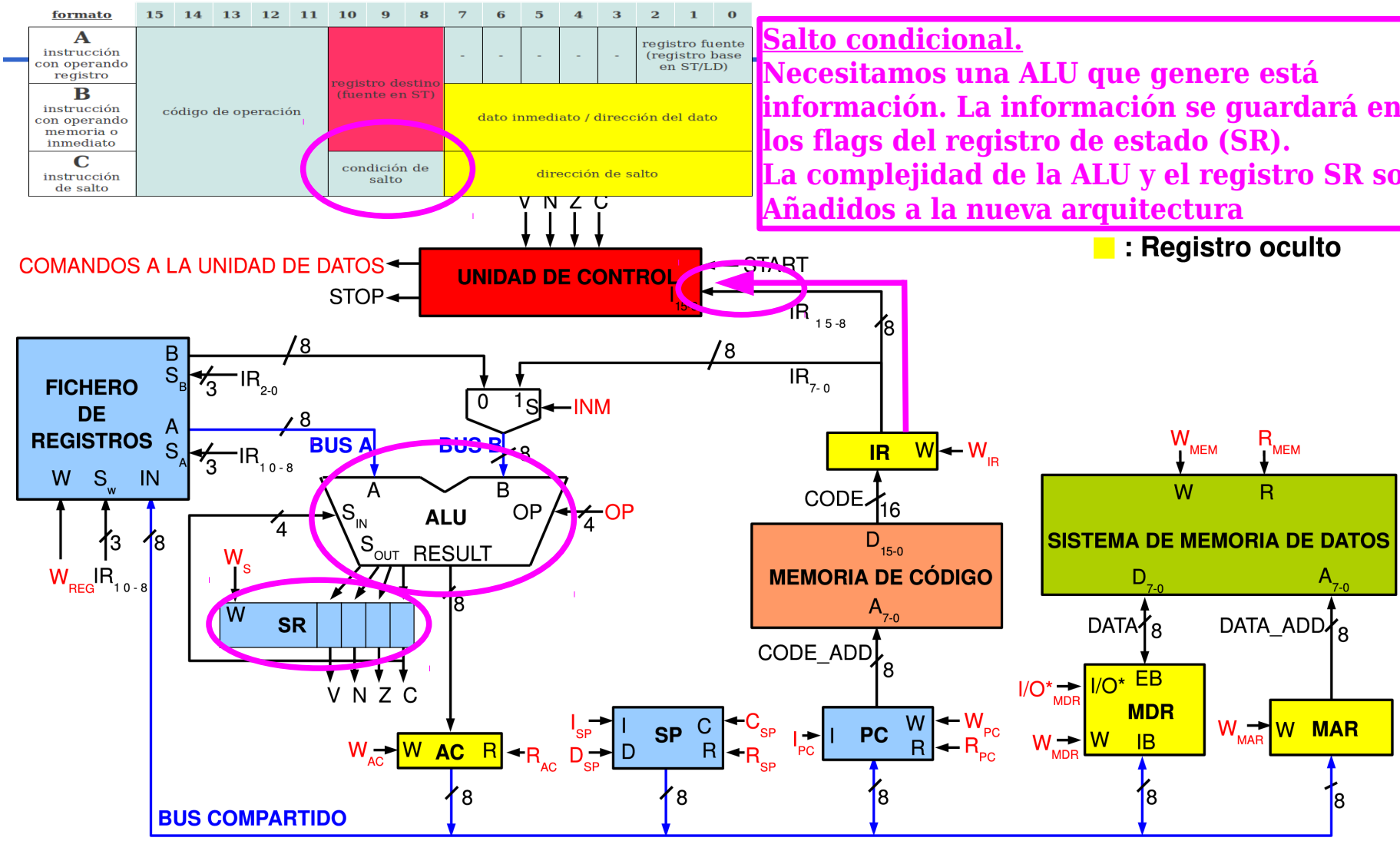
Salto incondicional.
Salta siempre, no se utiliza el campo condición de salto. Los 8 bits del IR se guardan en el PC y eso produce el salto. La próxima instrucción que busque el sistema será en la dirección del salto



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



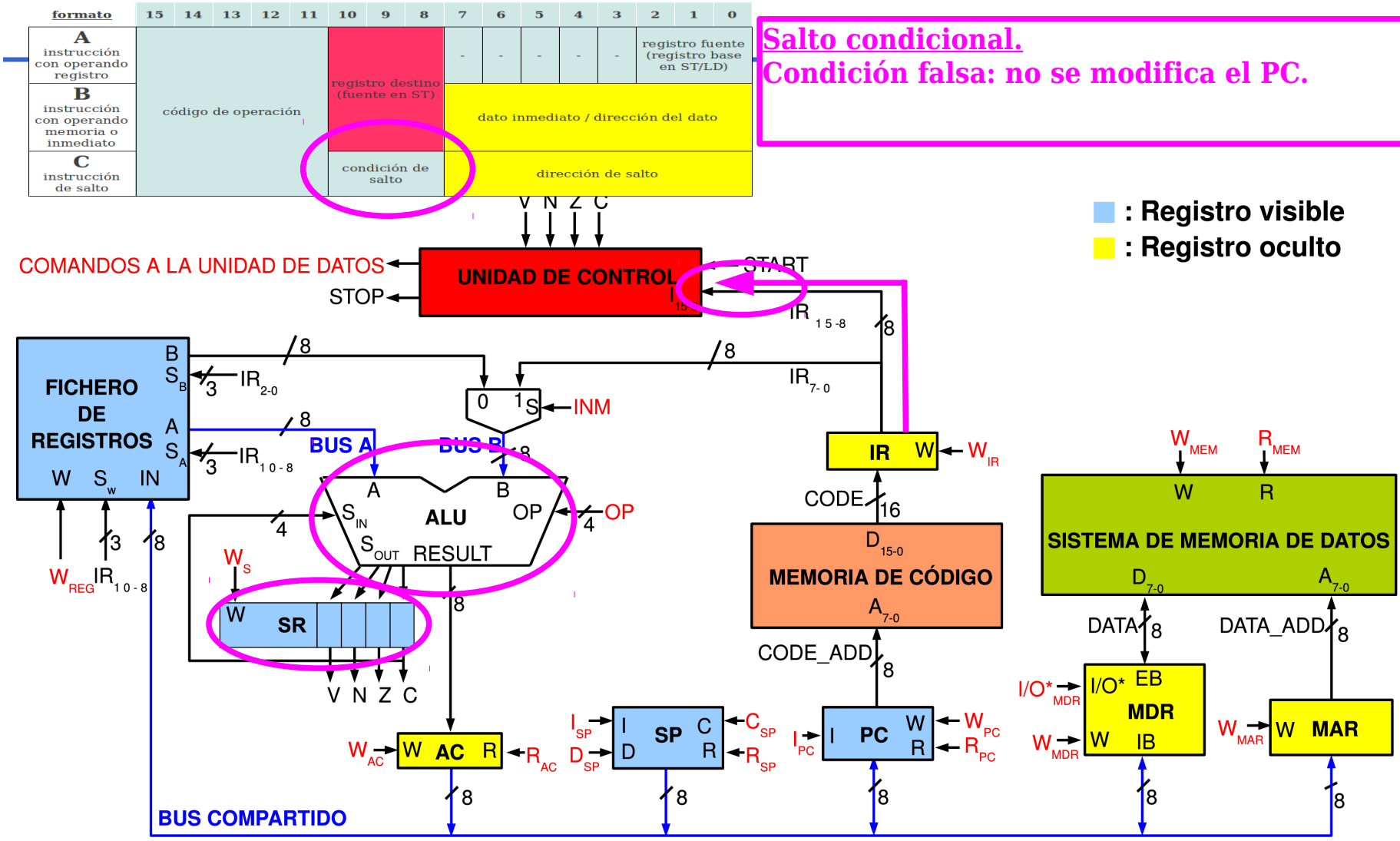

formato
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
A instrucción con operando registro
B instrucción con operando memoria o inmediato
C instrucción de salto
código de operación
registro destino (fuente en ST)
condición de salto
- - - - - registro fuente (registro base en ST/LD)
dato inmediato / dirección del dato
dirección de salto
Salto condicional.
La condición para que se produzca un salto Viene dada por el resultado de la operación que se realizó durante la ejecución de la instrucción anterior. ¿Fué el dato 0,positivo, desbordamiento?.
: Registro oculto
V N Z C
COMANDOS A LA UNIDAD DE DATOS
UNIDAD DE CONTROL
START
STOP
IR 15-8
IR 7-0
FICHERO DE REGISTROS
B S_B
A S_A
W S_W IN
IR 2-0
IR 10-8
W_REG
0 1 S
INM
BUS A
BUS B
A
B
ALU
S_IN
S_OUT
RESULT
OP
W_S
W SR
V N Z C
IR W
W_IR
CODE 16
D 15-0
MEMORIA DE CÓDIGO
A 7-0
CODE_ADD
W_MEM
R_MEM
W R
SISTEMA DE MEMORIA DE DATOS
D 7-0
A 7-0
DATA
DATA_ADD
I/O* EB
MDR
W IB
I/O*_MDR
W_MDR
W_MAR
W MAR
W_AC
W AC R
R_AC
I_SP
D_SP
I D SP C R
C_SP
R_SP
I_PC
I PC W R
W_PC
R_PC
BUS COMPARTIDO


formato
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
A instrucción con operando registro
B instrucción con operando memoria o inmediato
C instrucción de salto
código de operación
registro destino (fuente en ST)
registro fuente (registro base en ST/LD)
dato inmediato / dirección del dato
condición de salto
dirección de salto
Salto condicional.
Necesitamos una ALU que genere está 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 oculto
V N Z C
COMANDOS A LA UNIDAD DE DATOS
STOP
UNIDAD DE CONTROL
START
IR 15-8
FICHERO DE REGISTROS
W S_w IN
B S_B
A S_A
IR 2-0
IR 10-8
W_REG
BUS A
BUS B
0 1 S
INM
IR 7-0
IR W
W_IR
ALU
A B
S_IN S_OUT
OP
RESULT
W_S
SR
V N Z C
CODE 16
D 15-0
MEMORIA DE CÓDIGO
A 7-0
CODE_ADD
W_MEM R_MEM
SISTEMA DE MEMORIA DE DATOS
D 7-0 A 7-0
DATA
DATA_ADD
I/O* EB
MDR
W IB
I/O* MDR
W_MDR
W_MAR
W MAR
W_AC
W AC R
R_AC
I_SP D_SP
I D SP C R
C_SP R_SP
I_PC
I PC W R
W_PC R_PC
BUS COMPARTIDO

Información de lo que significan los códigos de condición

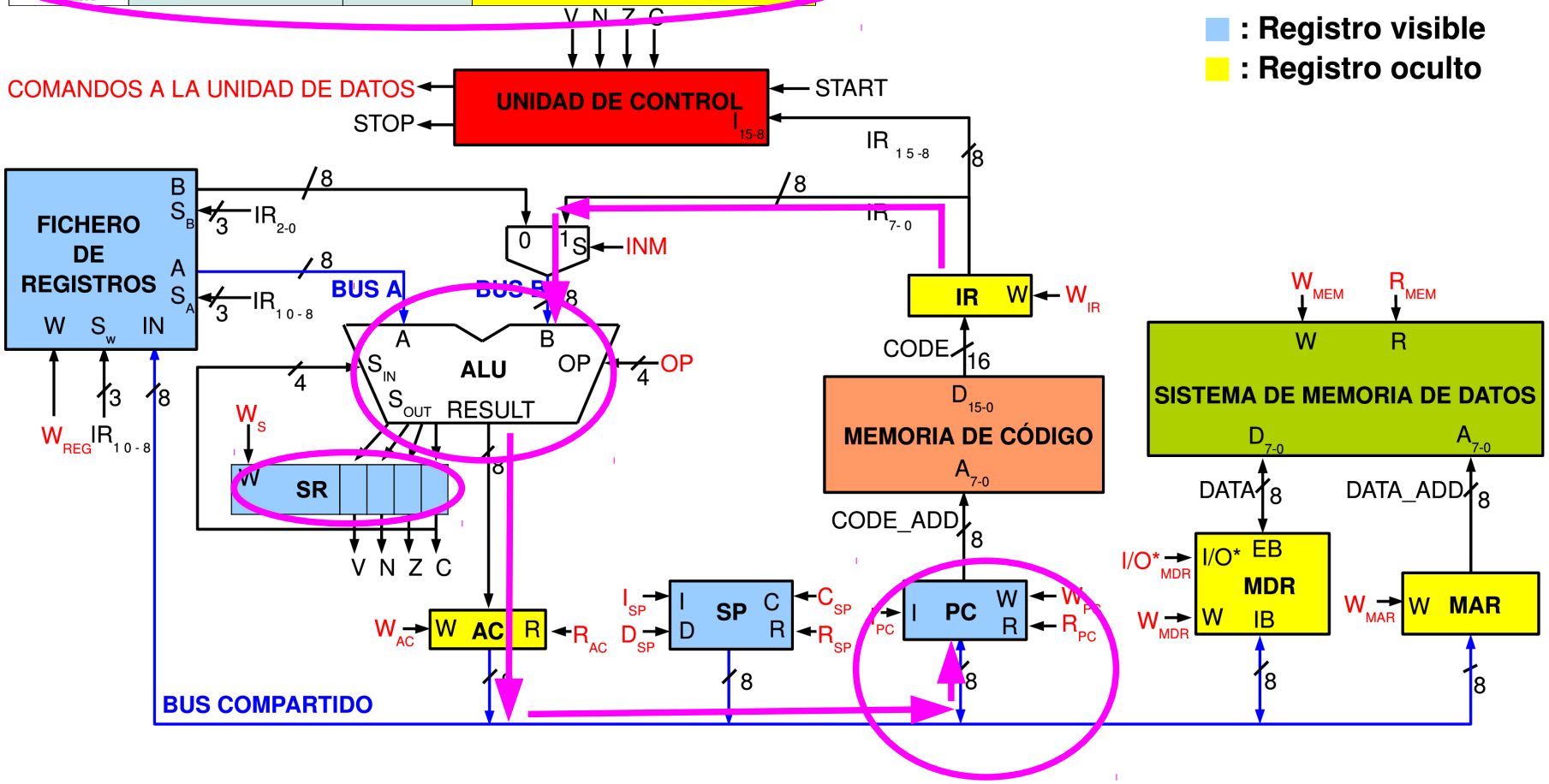
I_{10}	I_9	I_8	CONDICIÓN	nemónico(s) de la condición	notas
0	0	0	Z	ZS, EQ	será cierta justo tras realizar la resta A-B si y solo si A=B
0	0	1	C	CS, LO	será cierta justo tras realizar la resta A-B si y solo si A<B asumiendo notación base 2 sin signo
0	1	0	V	VS	será cierta si y solo si el dato recién calculado 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

La descripción detallada de la ALU y el mnemónico de la instrucción de salto condicional será expuesto más adelante. Se trata ahora de entender la relación entre los bits del código de condición y el flag generado por la ALU y que se guarda en el registro de estado.


formato
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
A instrucción con operando registro
B instrucción con operando memoria o inmediato
C instrucción de salto
código de operación
registro destino (fuente en ST)
registro fuente (registro base en ST/LD)
dato inmediato / dirección del dato
condición de salto
dirección de salto
Salto condicional.
Condición falsa: no se modifica el PC.
: Registro visible
: Registro oculto
V N Z C
COMANDOS A LA UNIDAD DE DATOS
STOP
UNIDAD DE CONTROL
START
IR 15-8
IR 7-0
FICHERO DE REGISTROS
W S_w IN
B S_B
A S_A
IR 2-0
IR 10-8
W_REG
0 1 S
INM
BUS A
BUS B
A
B
ALU
S_IN
S_OUT
RESULT
OP
W_S
SR
V N Z C
IR W
W_IR
CODE 16
D 15-0
MEMORIA DE CÓDIGO
A 7-0
CODE_ADD
W_MEM R_MEM
W R
SISTEMA DE MEMORIA DE DATOS
D 7-0 A 7-0
DATA
DATA_ADD
I/O* MDR
I/O* EB
MDR
W IB
W_MDR
W_MAR
W MAR
W_AC W AC R R_AC
I_SP D_SP
I D SP C R
C_SP R_SP
I_PC
I PC W R
W_PC R_PC
BUS COMPARTIDO

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

Salto condicional.
Condición verdadera. El PC es modificado.



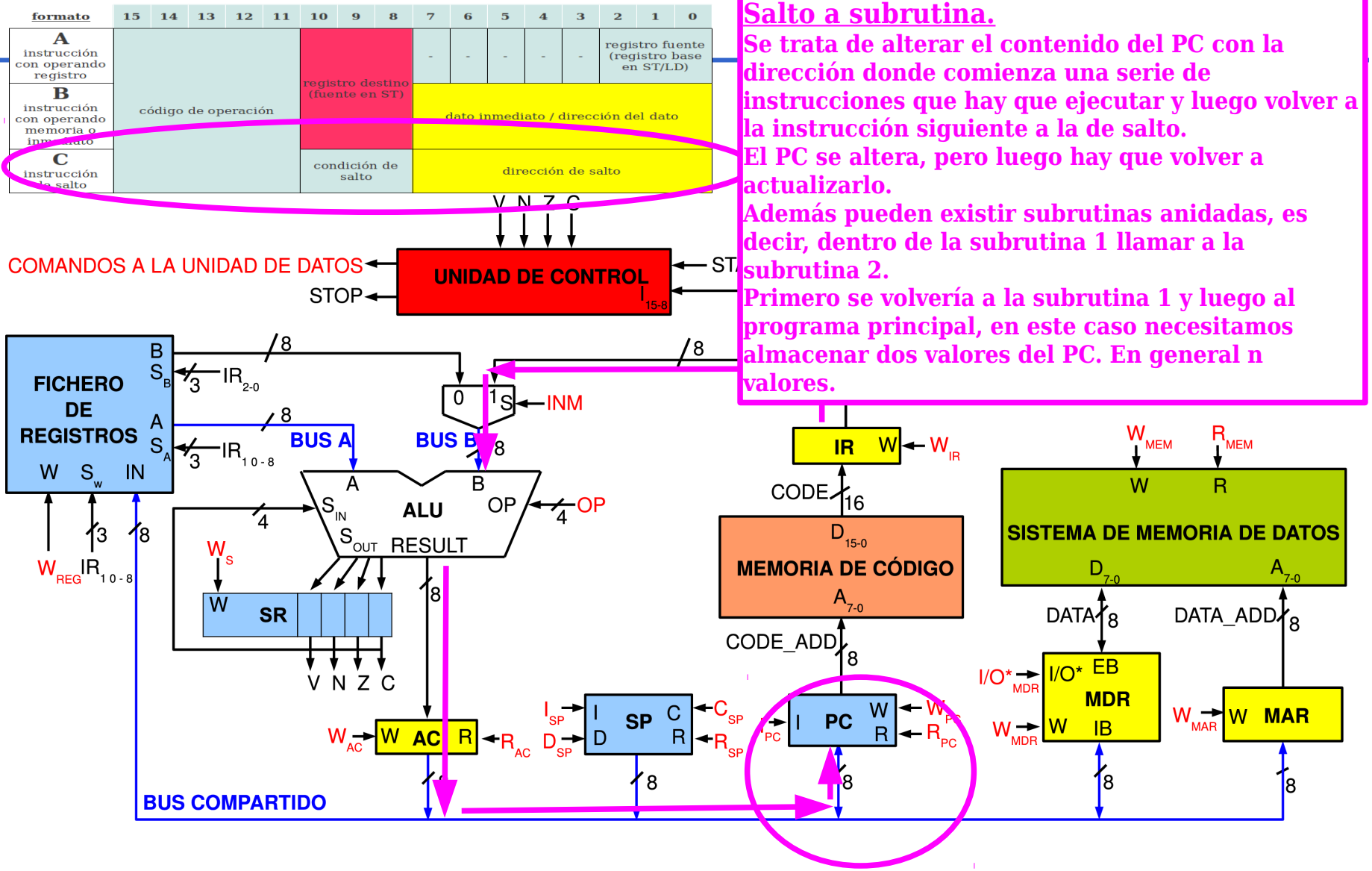
Salto a subrutina.

Se trata de alterar el contenido del PC con 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 volver a actualizarlo.

Además pueden existir subrutinas anidadas, es decir, dentro de la subrutina 1 llamar a la subrutina 2.

Primero se volvería a la subrutina 1 y luego al programa principal, en este caso necesitamos almacenar dos valores del PC. En general n valores.

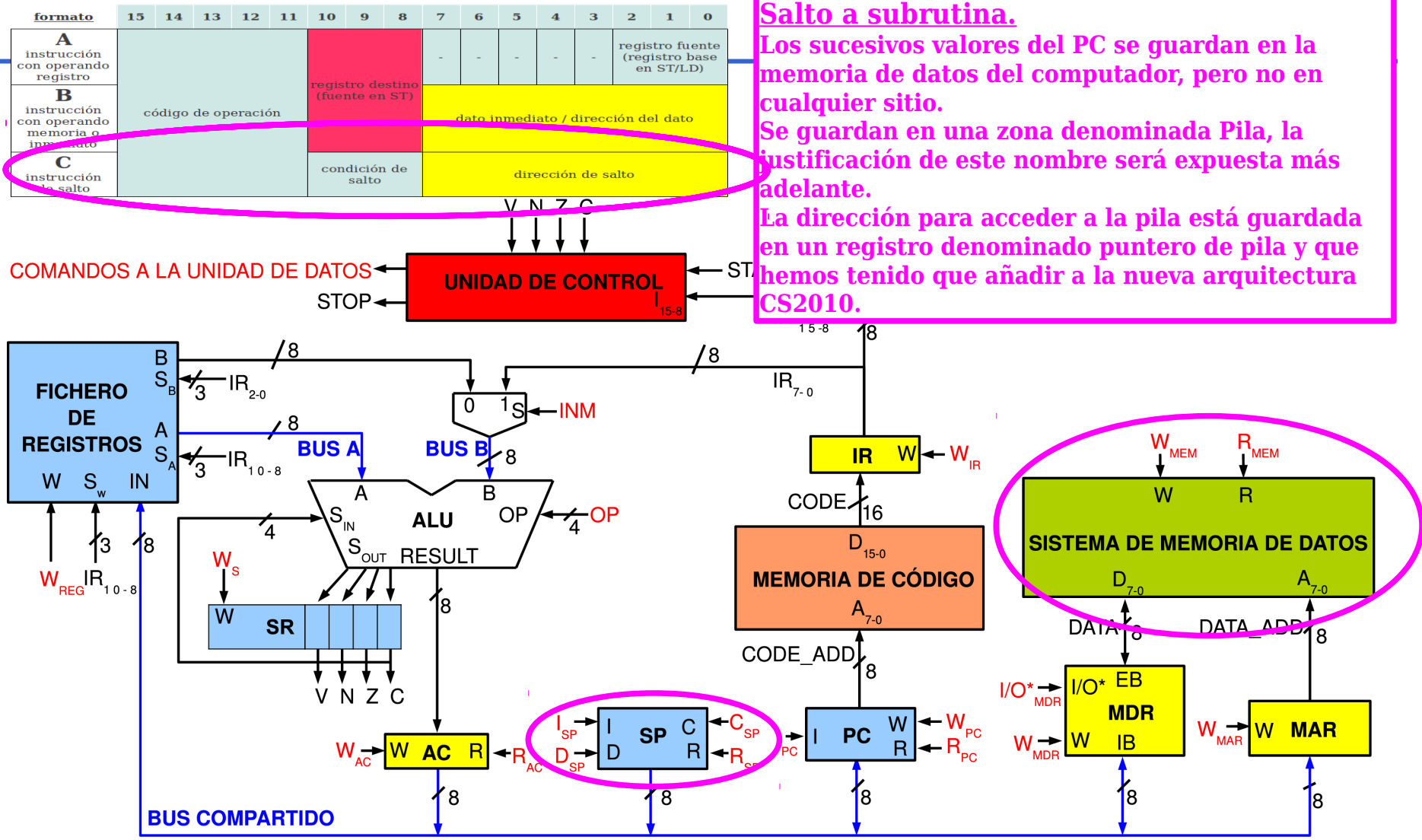


Salto a subrutina.

Los sucesivos valores del PC se guardan en la memoria de datos del computador, pero no en cualquier sitio.

Se guardan en una zona denominada Pila, la justificación de este nombre será expuesta más adelante.

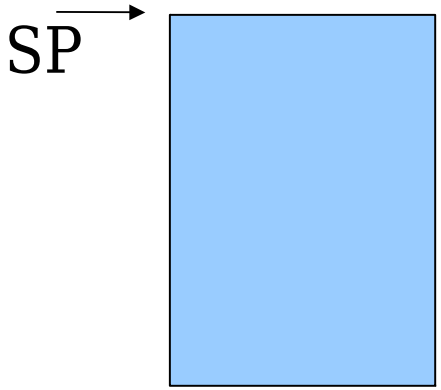
La dirección para acceder a la pila está guardada en un registro denominado puntero de pila y que hemos tenido que añadir a la nueva arquitectura CS2010.



Estructura de la Pila

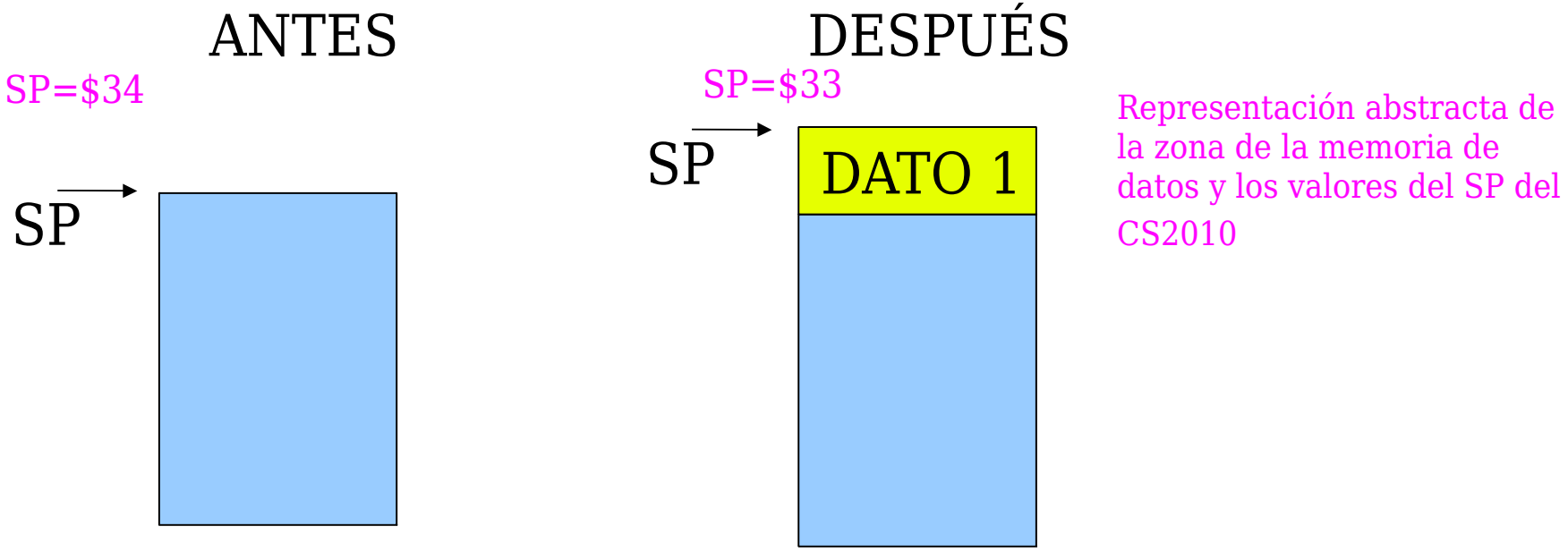
Antes de meter un nuevo dato el SP tiene guardada la dirección de la pila donde el dato se guardará.
Es como si se hiciera una pila de libros, el primero que quito es el último que he puesto.
El primer dato se guarda en una dirección determinada y el siguiente, en la dirección inmediatamente inferior. La pila crece hacia direcciones decrecientes de la memoria.

Supongamos SP=$34



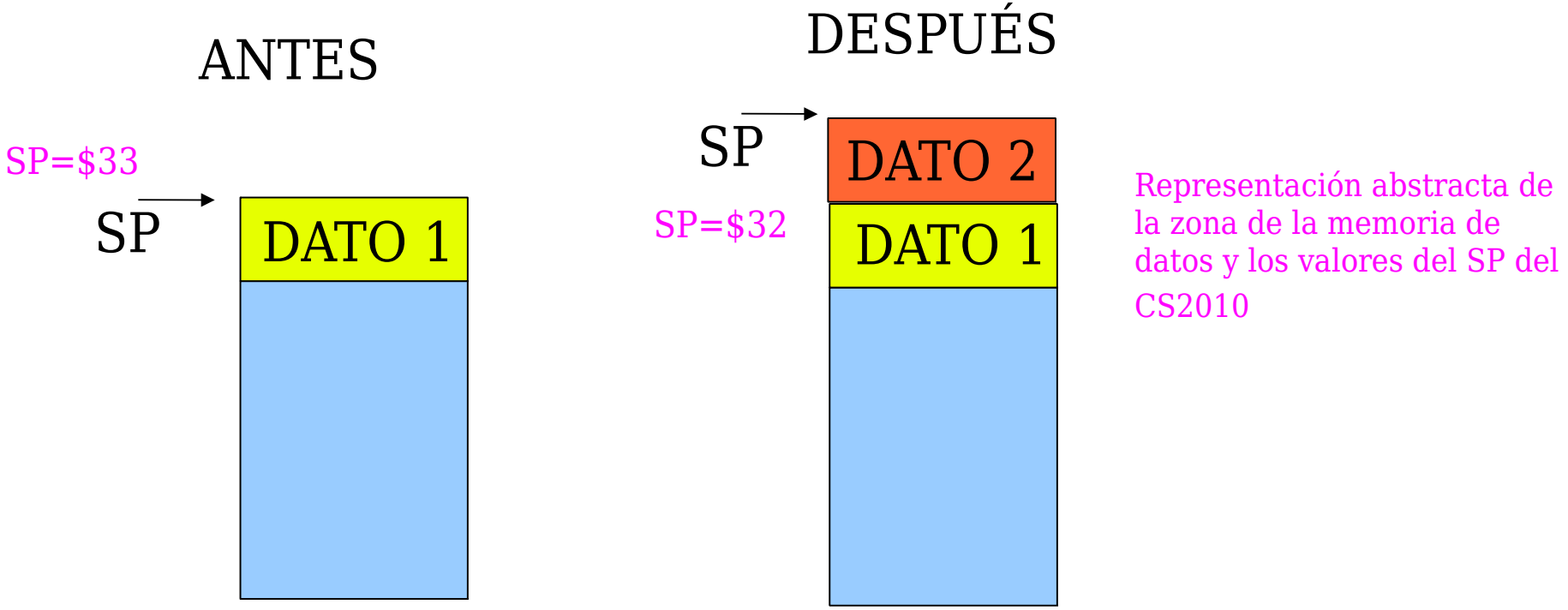
Representación abstracta de la zona de la memoria de datos y los valores del SP del CS2010

Estructura de la Pila



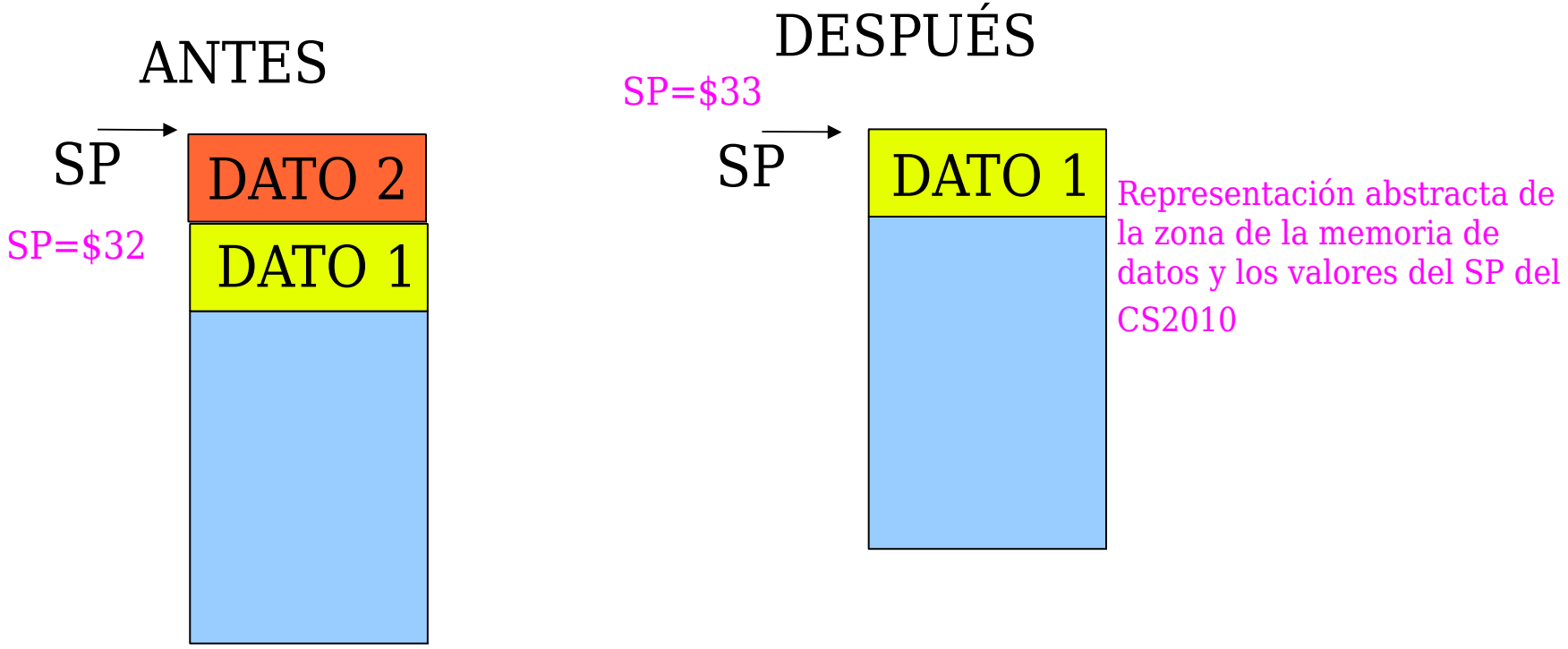
Proceso de guardar datos en la pila. Operación de PUSH

Estructura de la Pila



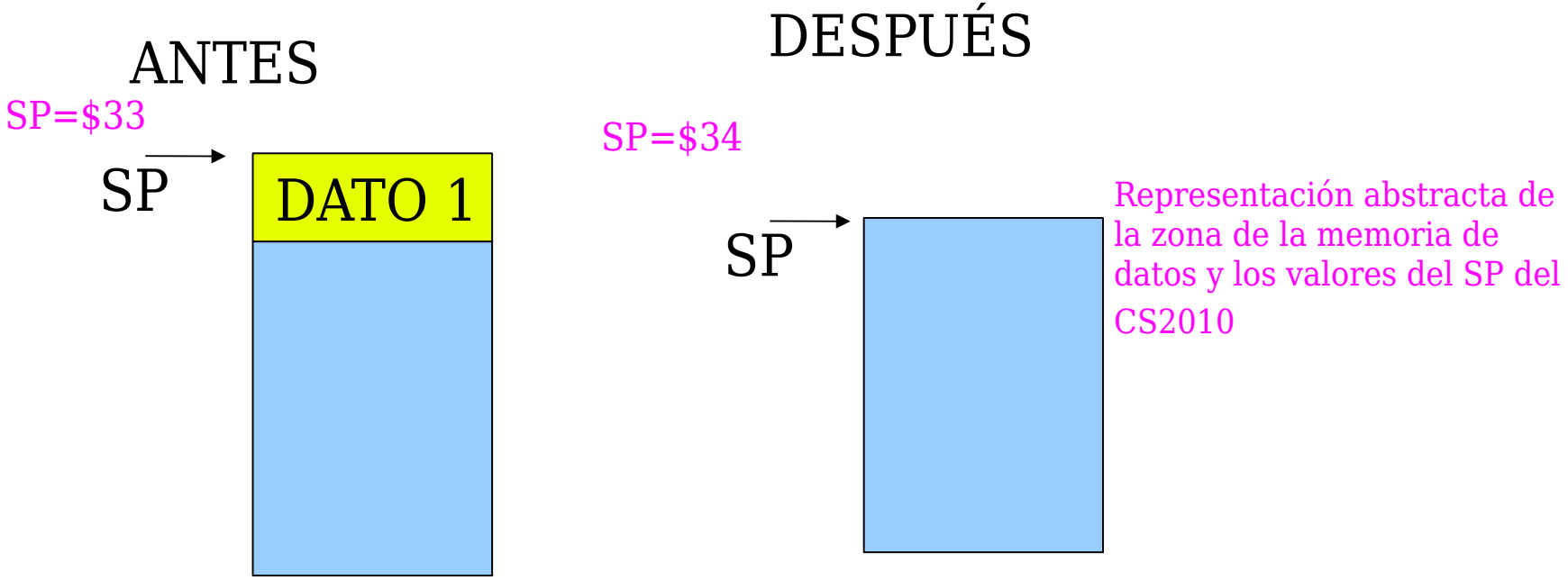
Proceso de guardar datos en la pila. Operación de PUSH

Estructura de la Pila



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

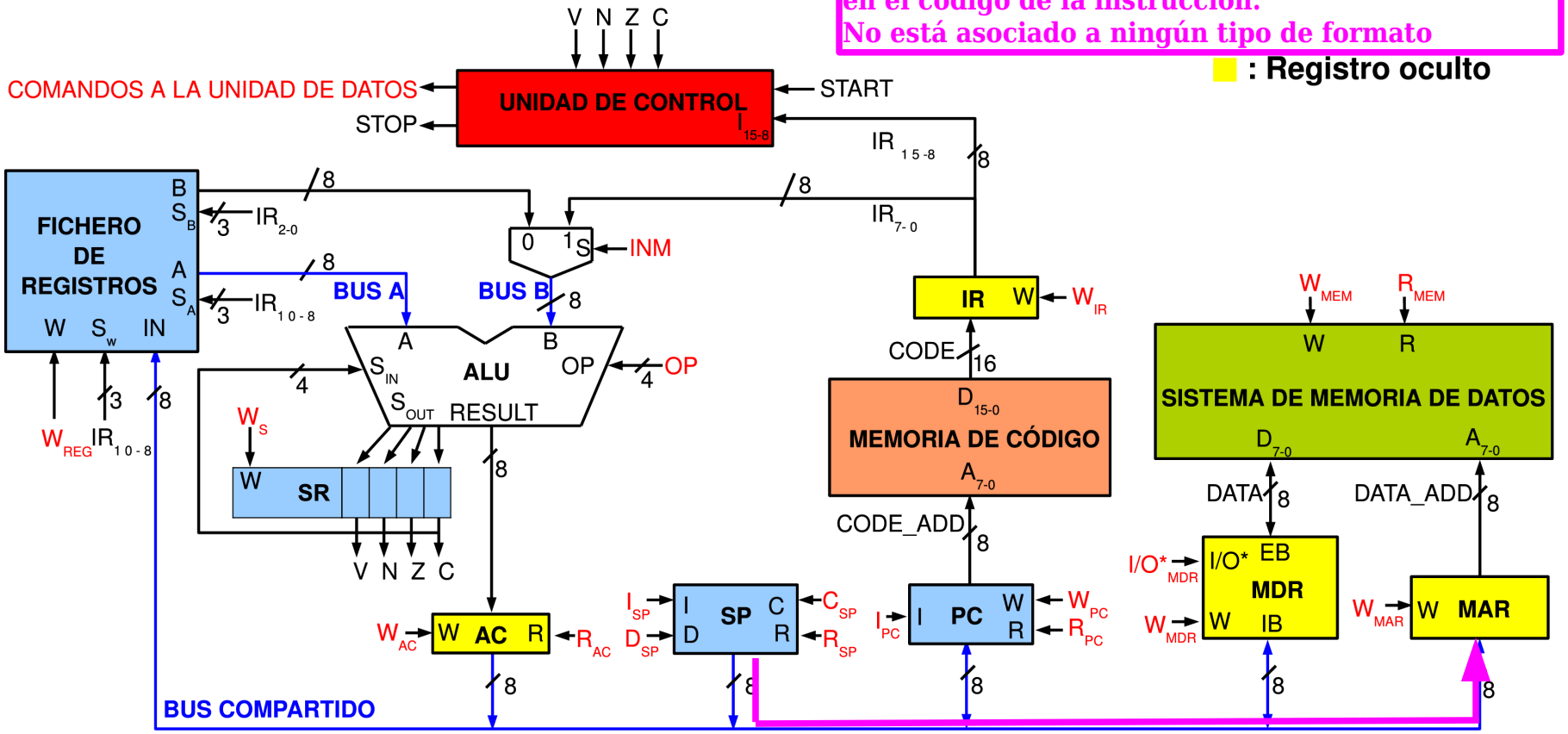
Estructura de la Pila



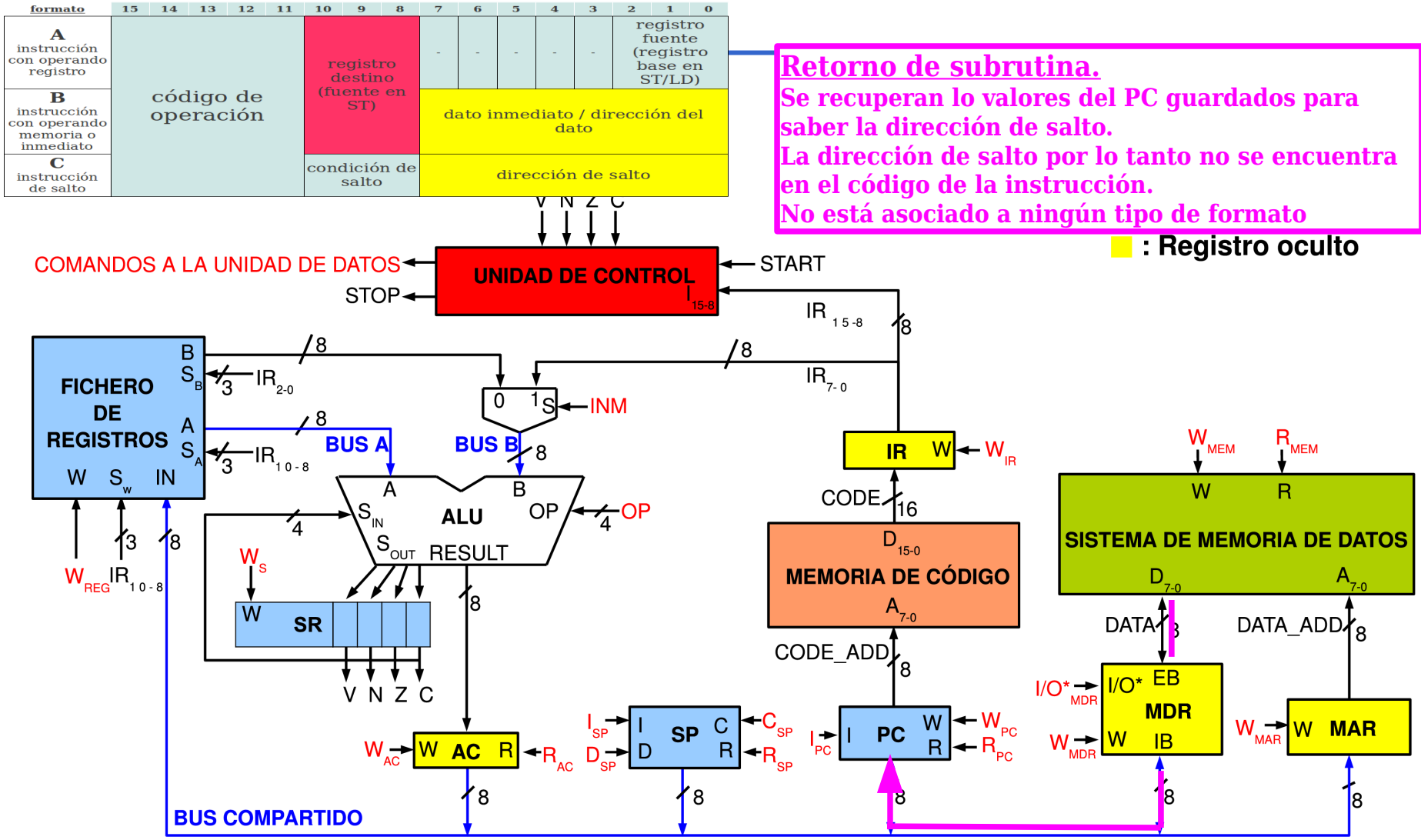
Proceso de extracción de datos de la pila. Operación de PULL. El primero que Lee es el último que se escribió.

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

Retorno de subrutina.
Se recuperan lo valores del PC guardados para saber la dirección de salto.
La dirección de salto por lo tanto no se encuentra en el código de la instrucción.
No está asociado a ningún tipo de formato



Decripción general:
Incrementar SP y guardarlo en MAR
Esto permite conocer la dirección del salto



Retorno de subrutina.
Se recuperan lo valores del PC guardados para saber la dirección de salto.
La dirección de salto por lo tanto no se encuentra en el código de la instrucción.
No está asociado a ningún tipo de formato

: Registro oculto

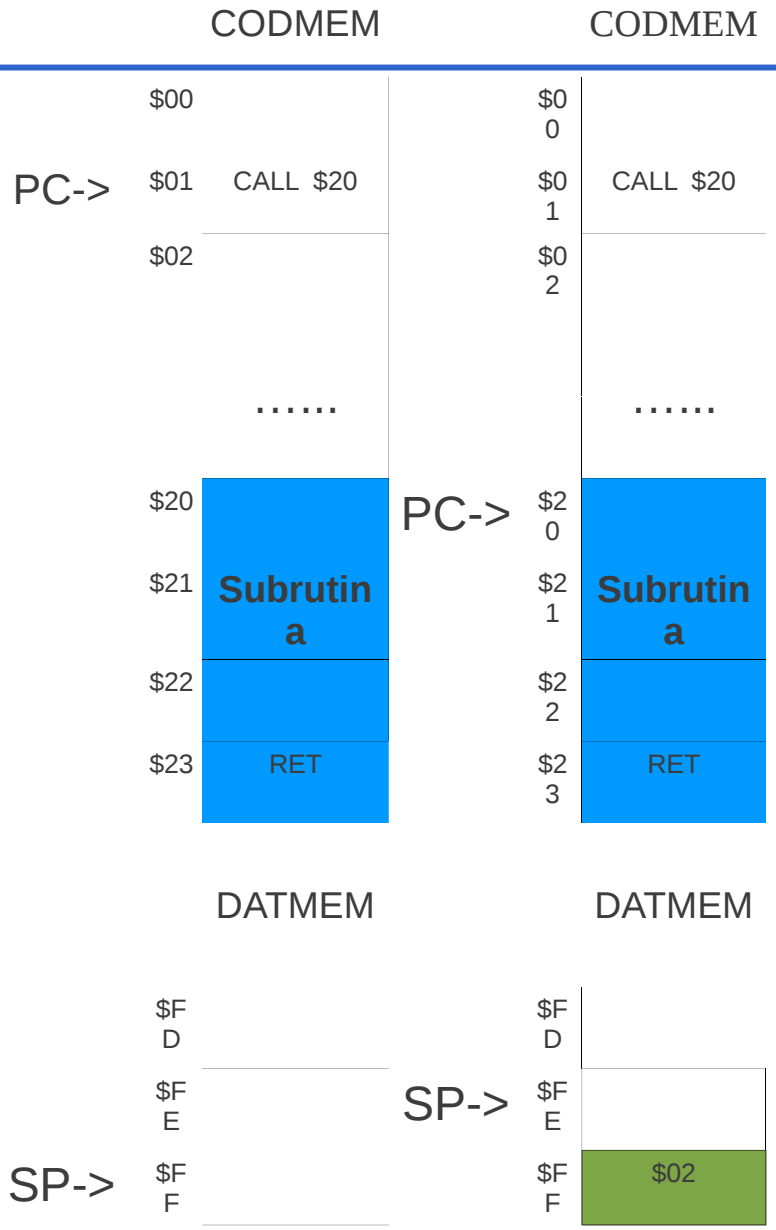
Decripción general:
Acceso a la pila recuperación del valor Deseado del PC, vuelta a la instrucción siguiente a La de salto a subrutina

Instrucciones de llamada y retorno de subrutinas

- Una subrutina es una secuencia de instrucciones que realizan una cierta tarea a partir de unos datos de entrada, generando unos datos de salida
- Cuando un programa necesita realizar la tarea, hace una llamada a la subrutina, ésta se ejecuta y cuando termina, el programa se reanuda por donde se había quedado (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

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).

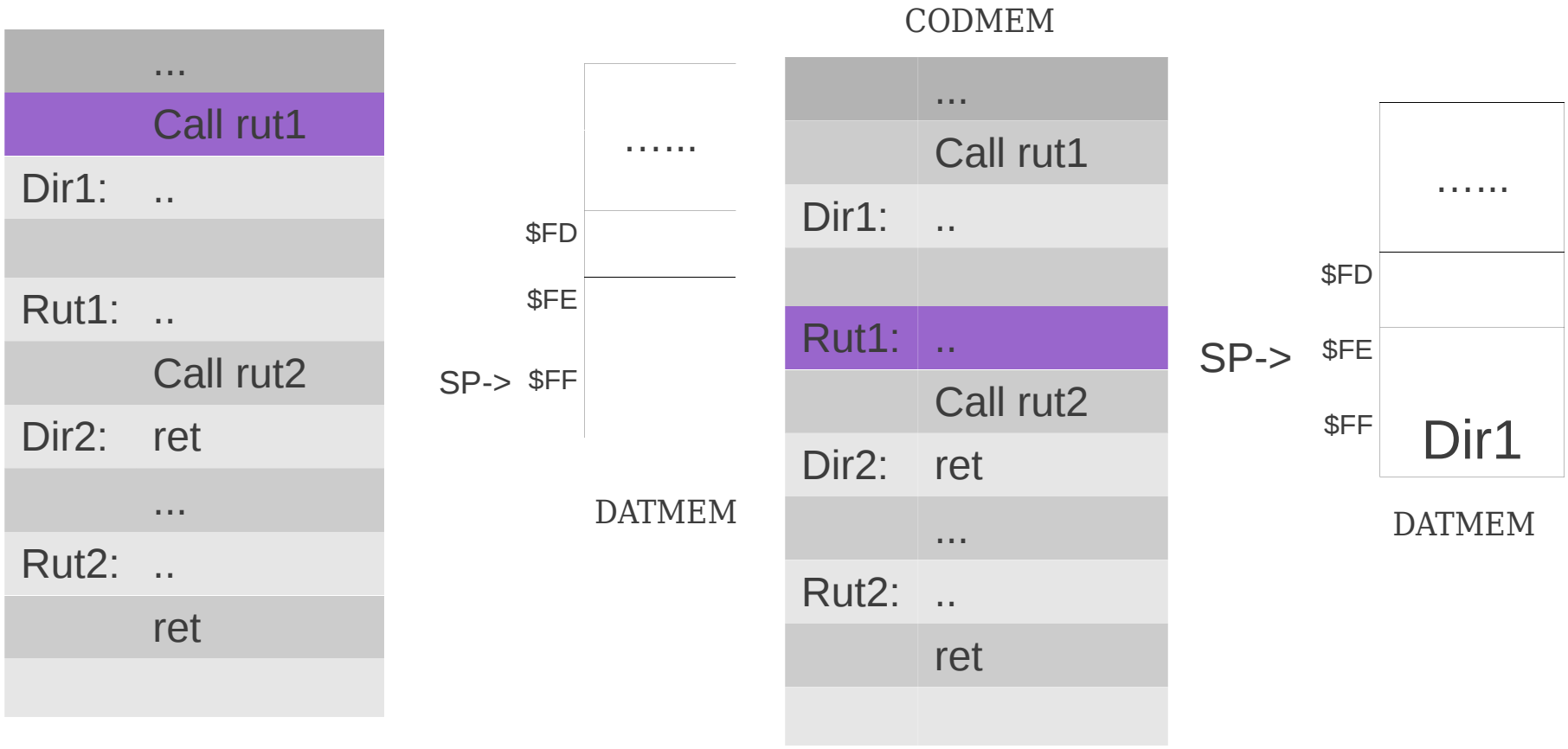


Instrucciones de llamada y retorno de subrutinas

- 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 sobreescrito.

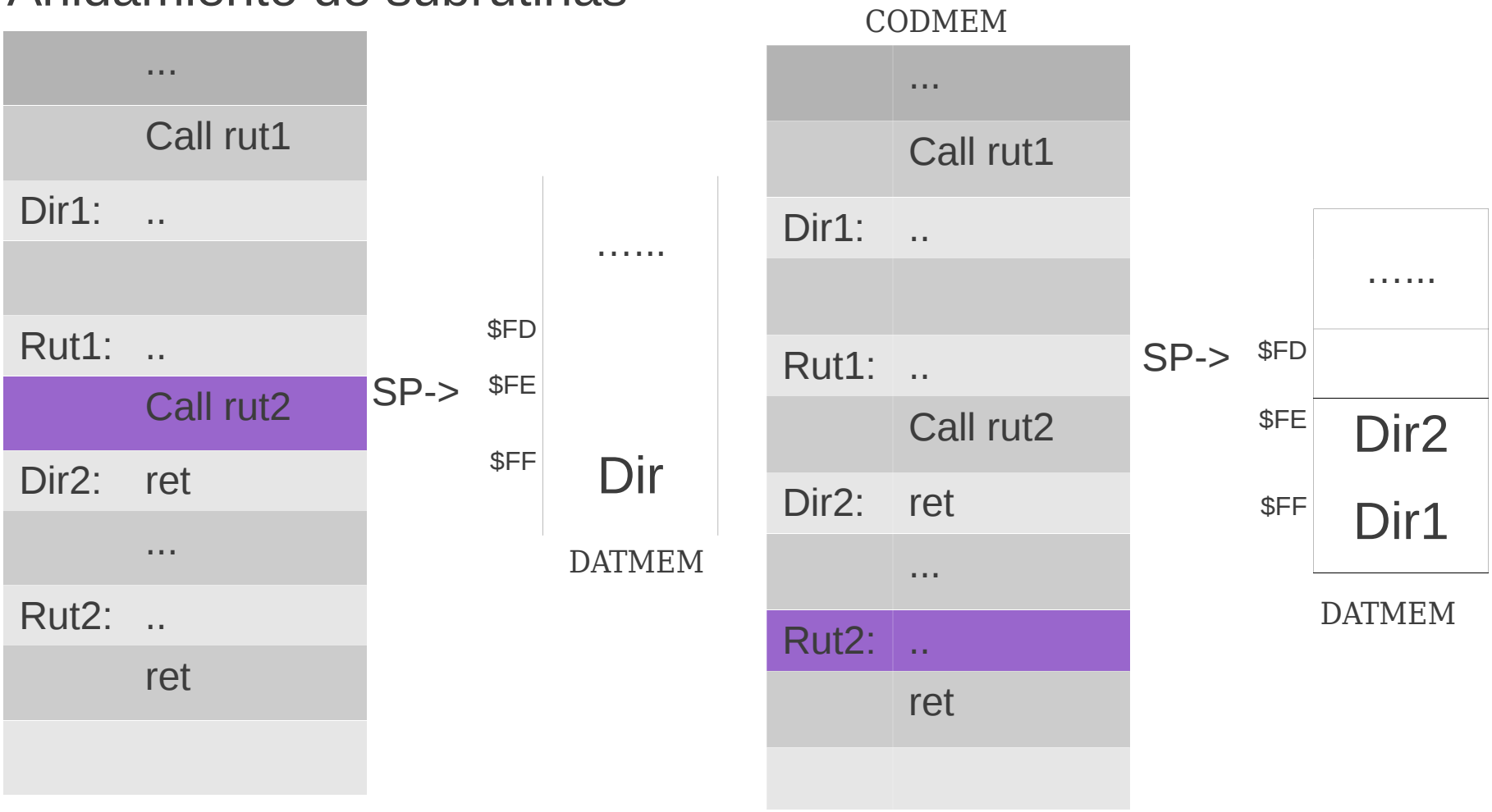
Instrucciones de llamada y retorno de subrutinas

- Anidamiento de subrutinas



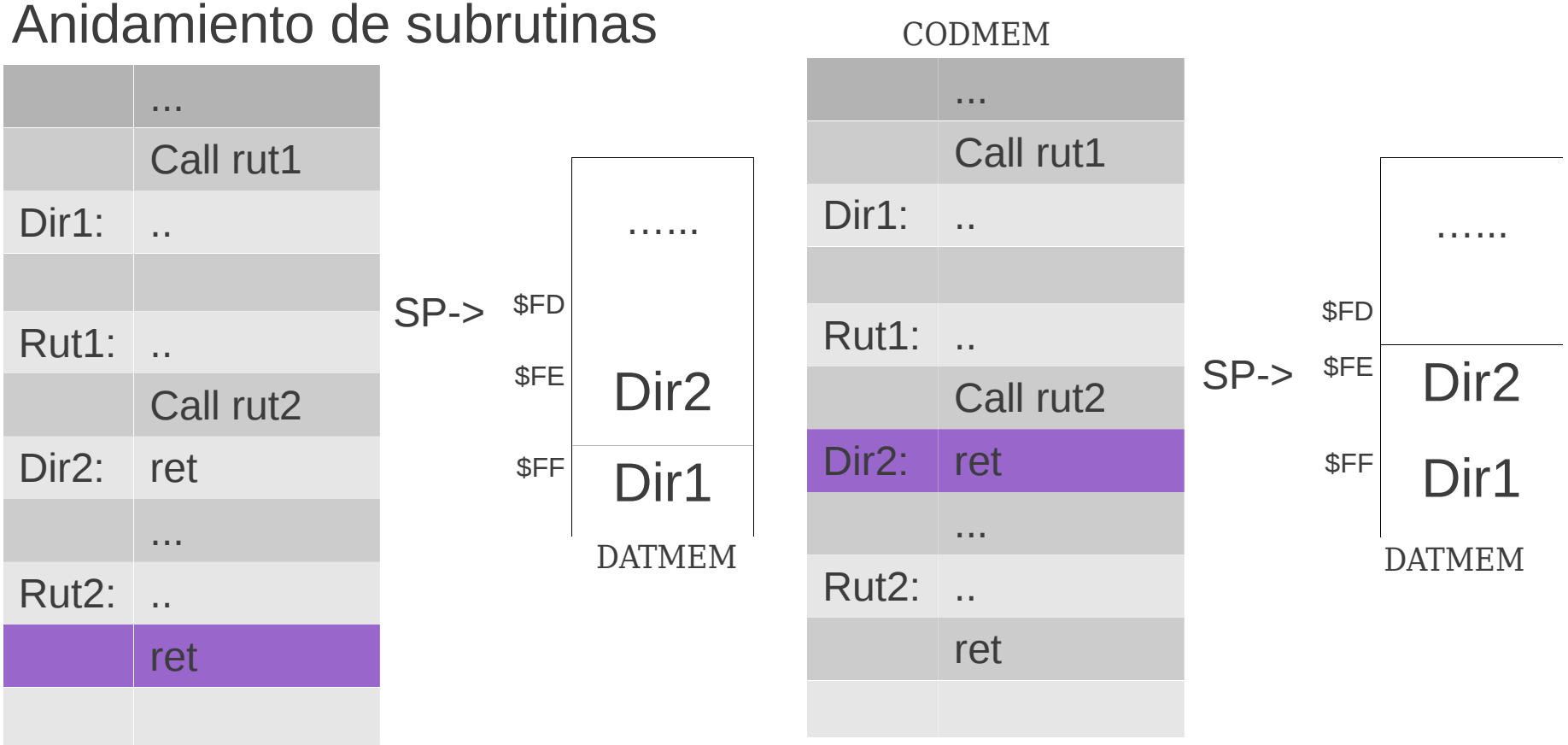
Instrucciones de llamada y retorno de subrutinas

Anidamiento de subrutinas



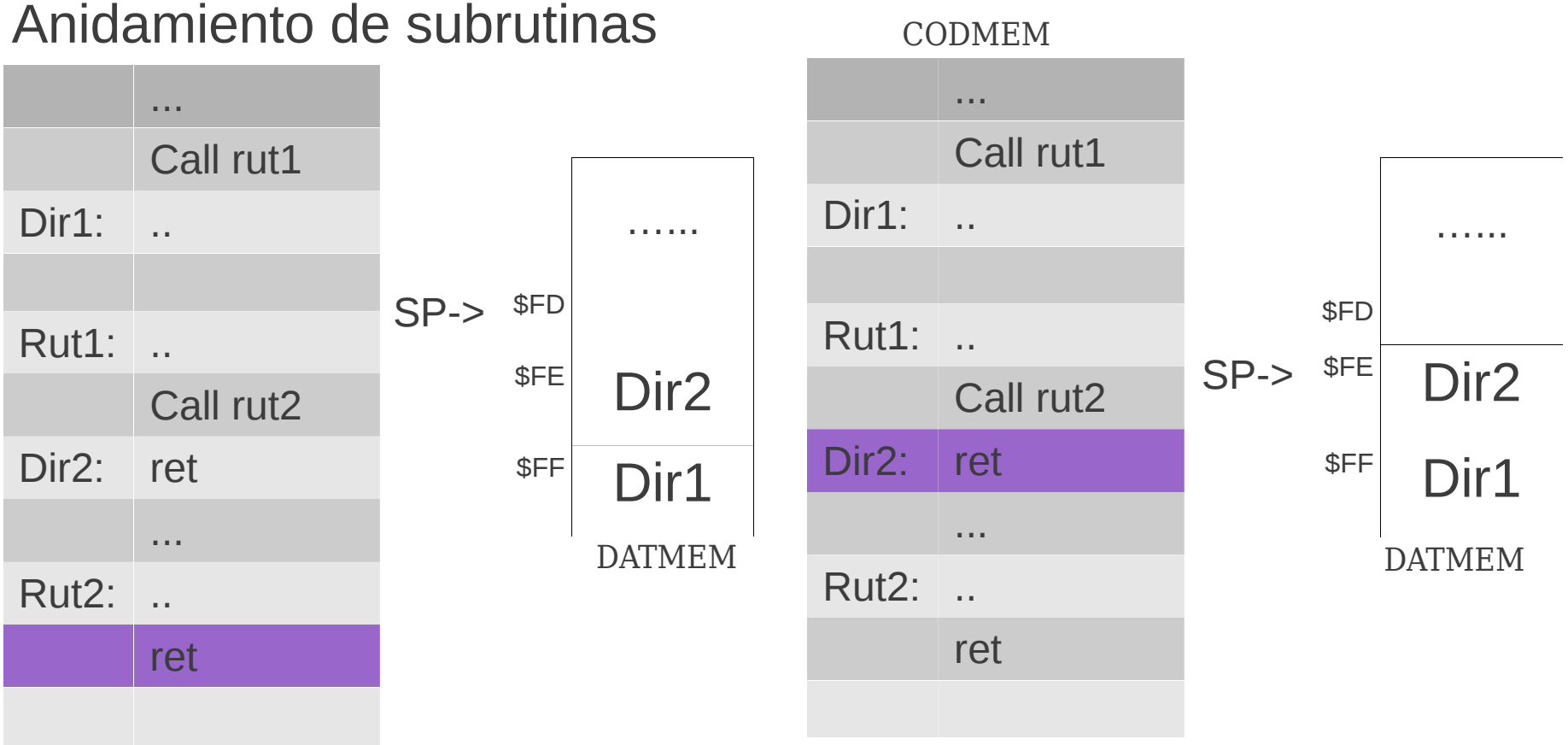
Instrucciones de llamada y retorno de subrutinas

Anidamiento de subrutinas



Instrucciones de llamada y retorno de subrutinas

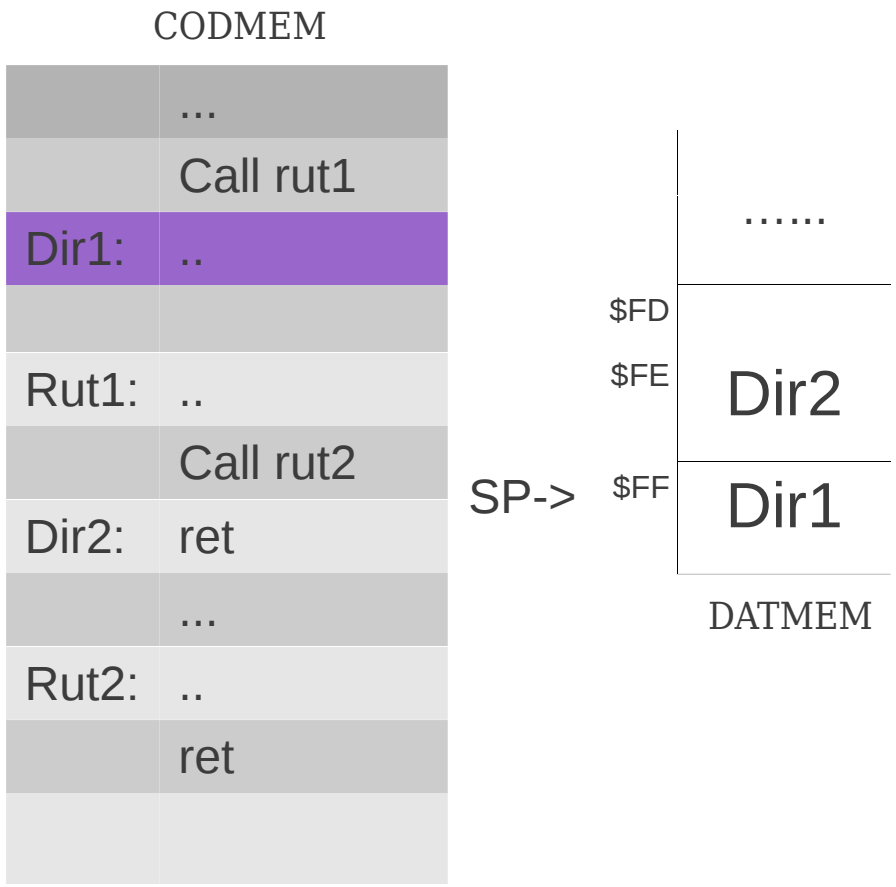
Anidamiento de subrutinas



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)

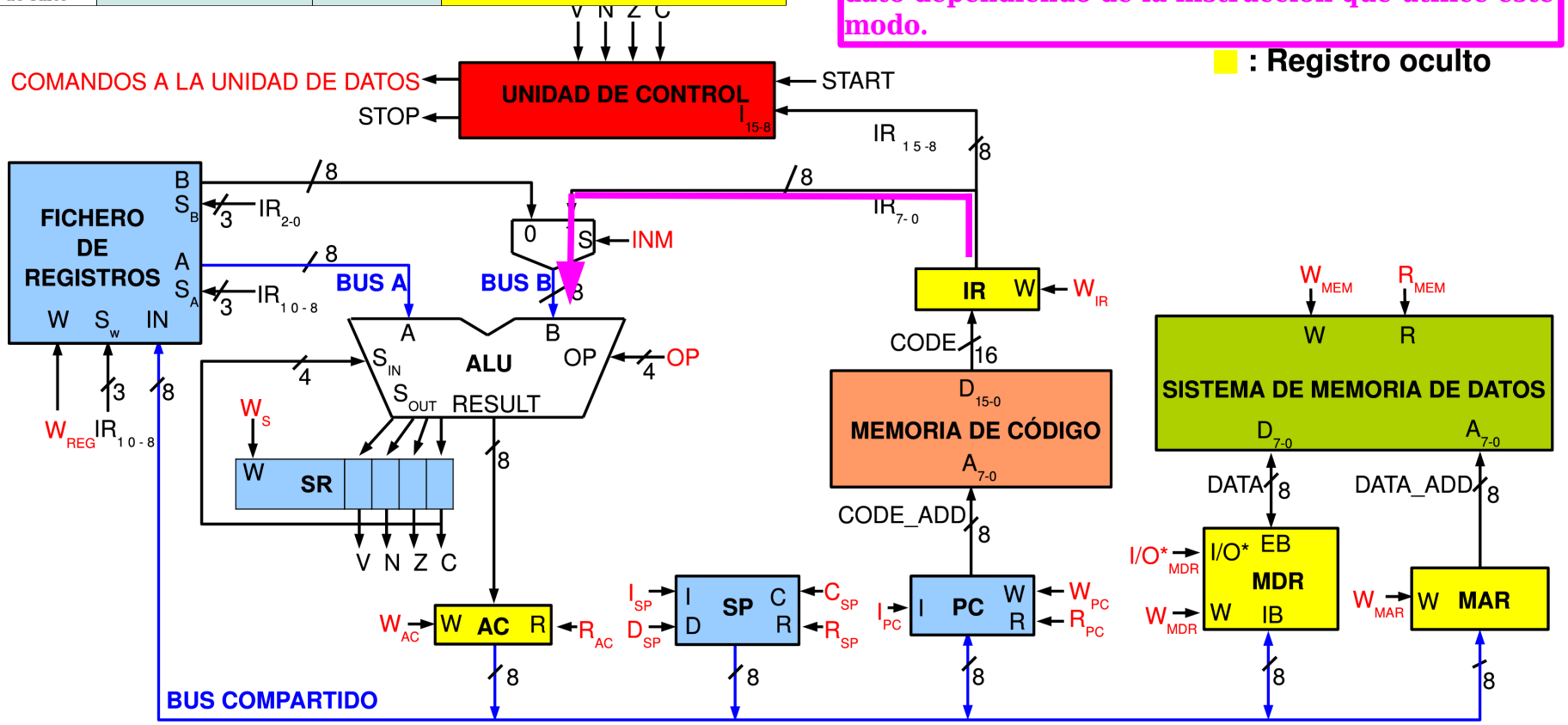


Modo inmediato

- Consiste en que el dato esta almacenado en el 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.



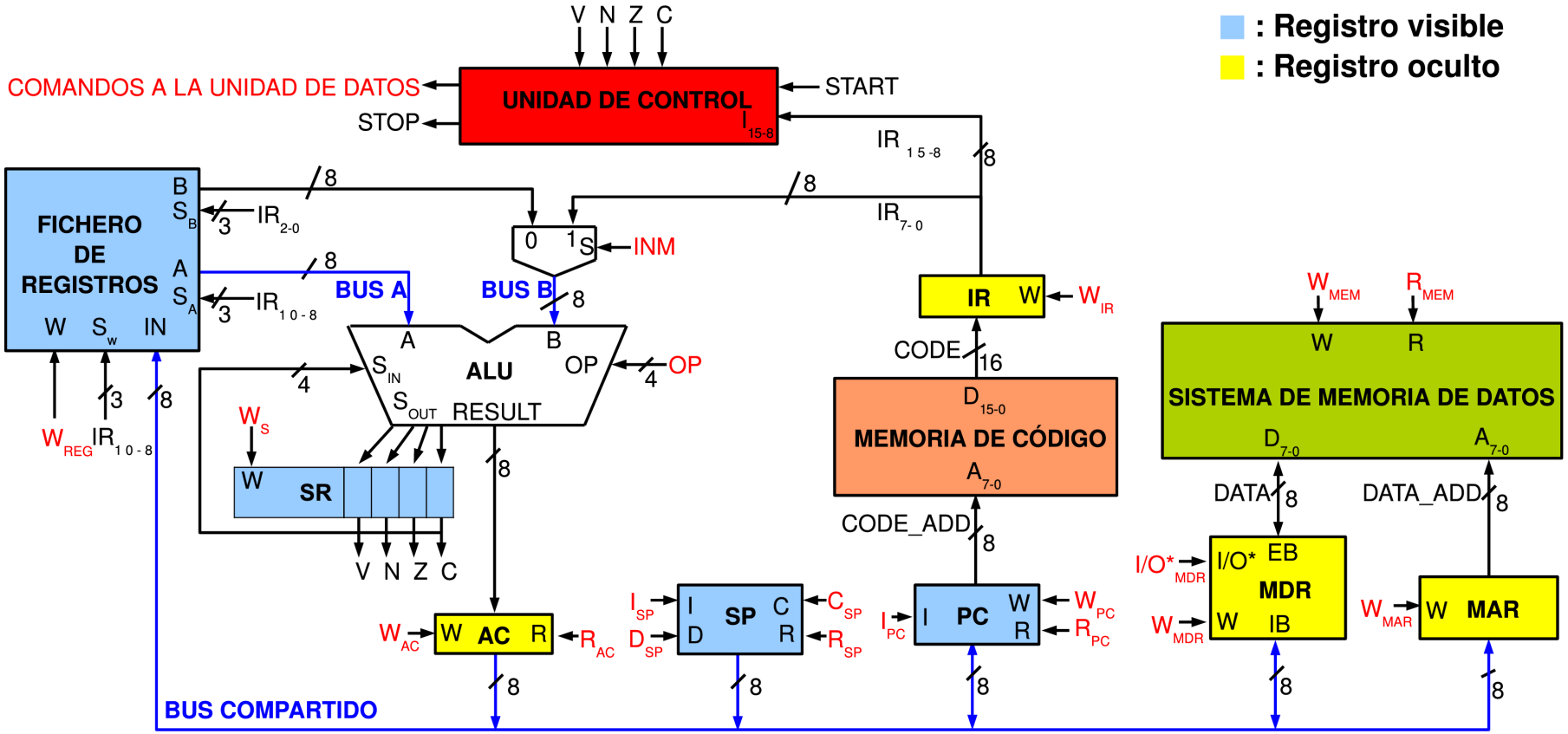
Registros visibles de la nueva arquitectura

- **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 (acarreo). 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. Codifíca 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.

Registros ocultos

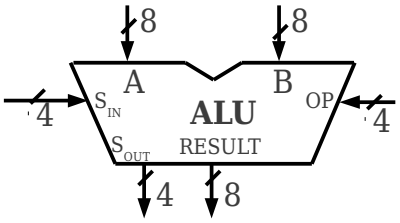
- **IR:** registro de 16 bits que sirve para almacenar la instrucción que está siendo ejecutada.
- **MDR.** Registro de 8 bits que sirve para almacenar los datos que se van a intercambiar con la memoria de datos.
- **MAR.** Registro de 8 bits que direcciona la memoria de datos.
- **AC.** Registro de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

Una posible implementación



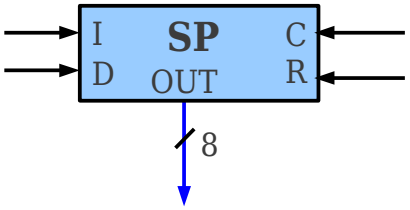
Una posible implementación

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_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	C_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-



I D C R	SP ←	OUT:=
0 0 0 0	SP	HI
0 0 0 1	SP	SP
0 0 1 0	0	HI
0 1 0 0	SP-1	HI
1 0 0 0	SP+1	HI
1 0 0 1	SP+1	SP
0 1 0 1	SP-1	SP
0 0 1 1	0	SP
otras	PROHIBIDAS	

La descripción de SR es idéntica a la del MAR

Formatos: Solo hay tres y comparten campos

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

Tabla de instrucciones

Asignación de códigos de operación: Facilita la decodificación

Bits del código de operación					NEMÓNICO	FORMATO	TIPO	SINTAXIS	EFECTO[1]	VNZC[2]
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)	Rfuente←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	Rfuente←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	BR*xx*	C	salto	BR*xx* 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	----

[1] (sin tener en cuenta el registro de estado y el incremento del PC)

[2] El caracter '-' denota “no modificado”, '*' denota “modificado de forma definida”, 'U' denota “no documentado”

Tabla de instrucciones

Asignación de códigos de operación: Facilita la decodificación

Bits del código de operación					NEMÓNICO	FORMATO	TIPO	SINTAXIS	EFECTO[1]	VNZC[2]
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)	Rfuente←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	Rfuente←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	BR*xx*	C	salto	BR*xx* 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	----

[1] (sin tener en cuenta el registro de estado y el incremento del PC)

[2] El caracter '-' denota “no modificado”, '*' denota “modificado de forma definida”, 'U' denota “no documentado”

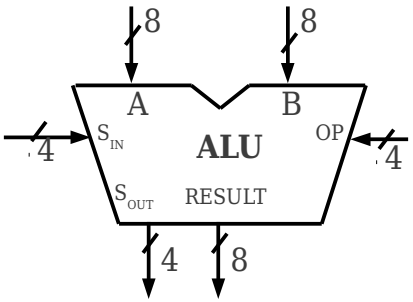
Códigos de condición de la instrucción de bifurcación condicional: BR*xx*

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

BRZS, BREQ, BRCS,BRLO, BRVS,BRLT

Relación de la ALU con las instrucciones



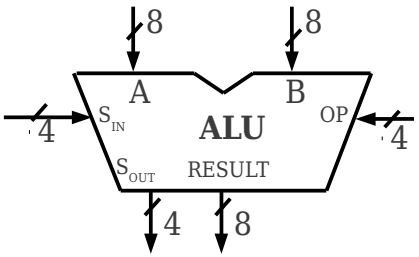
$S_{IN}=\{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$

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

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	C_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	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



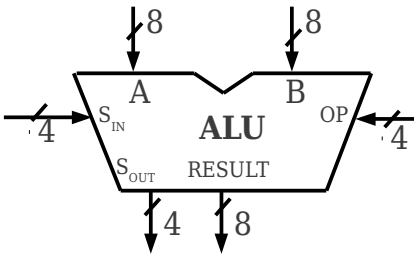
$S_{IN}=\{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$
$S_{OUT}=\{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	c_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	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



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

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	c_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

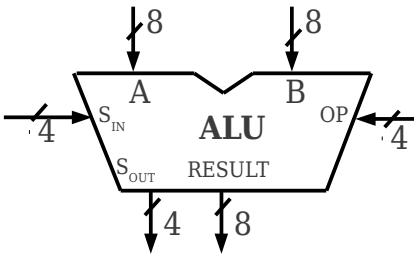
¿Porqué necesitamos estas operaciones? Instrucciones: ***ROR Rd*** **y** ***ROL Rd***

Como vemos la ALU rota el dato que entra por su entrada A con la entrada de acarreo, ya que la tiene disponible como hemos dicho anteriormente. El dato rotado es lo que nos proporciona en RESULT.

¿Qué información nos proporcionan los flags?

V: La operación EXOR de C_{in} con A_0 en el caso de desplazamiento a la dcha, y de A_7 con A_6 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

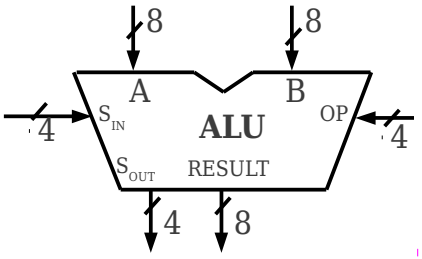


$S_{IN}=\{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$
$S_{OUT}=\{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	c_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos estas operaciones? Instrucciones: ***ROR Rd* y *ROL Rd***
¿Qué nos proporciona en 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



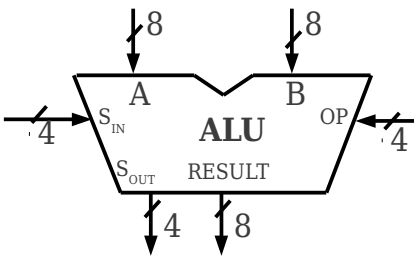
$S_{IN}=\{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$
$S_{OUT}=\{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	C_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

Lo que entra por A lo direccionan los bits $IR_{10-8:}$ fuente en ST (mirar arquitectura)
¿Porqué necesitamos esta operación? Instrucciones: ***ST (Rb),Rf; STS dirección,Rf;***

Los flags no importan.

Relación de la ALU con las instrucciones



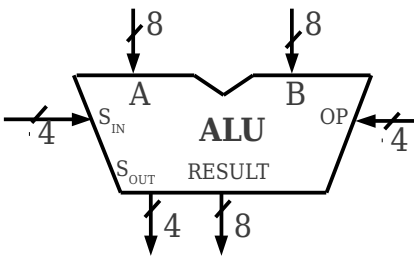
$S_{IN}=\{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$
$S_{OUT}=\{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	c_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

Instrucciones que necesitan que un dato pase por la entrada B sin alterarse.
El dato proviene de la pila de registros en el caso del registro base, lo direcciónan los bits $IR_{2\text{-}0:}$
o bien del registro IR si es una dirección o un dato inmediato (mirar arquitectura).

¿Porqué necesitamos esta operación? Instrucciones:***ST (Rb),Rf; STS dirección,Rf; LD Rd, (Rb); LDS Rd,dirección; CALL dirección, BRxx dirección; JMP dirección, LDI Rd,dato, MOV Rd,Rf***
Los flags no se ven afectados.

Relación de la ALU con las instrucciones



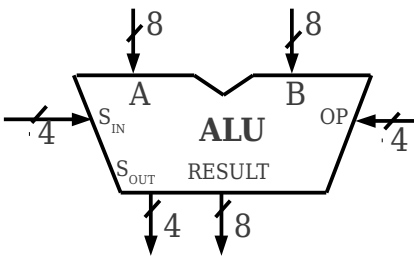
$S_{IN}=\{V_{IN}, N_{IN}, Z_{IN}, C_{IN}\}$
$S_{OUT}=\{V_{OUT}, N_{OUT}, Z_{OUT}, C_{OUT}\}$

OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	c_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos esta operación? Instrucciones que necesitan suma.
Instrucciones: ***ADD Rd,Rf; ADDI Rd,dato;***

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_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	0	1	-	-	-	-	-
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1
0	1	0	0	SHR(A,C_{IN})	c_{IN} EXOR A_0	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_0
0	1	0	1	SHL(A,C_{IN})	A_7 EXOR A_6	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	A_7
0	1	1	-	A	-	-	-	-
1	0	0	-	(A + B) mod 2^8	overflow(A+B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	carry(A+B)
1	0	1	-	(A - B) mod 2^8	underflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

¿Porqué necesitamos esta operación? Instrucciones que necesitan resta.
Instrucciones: ***SUB Rd,Rf; CP Rd,Rf; CPI Rd,dato; SUBI Rd,dato;***

V: desbordamiento en suma de números con signo.
N: MSB del resultado
Z: NOR del resultado indica si es cero
C: acarreo

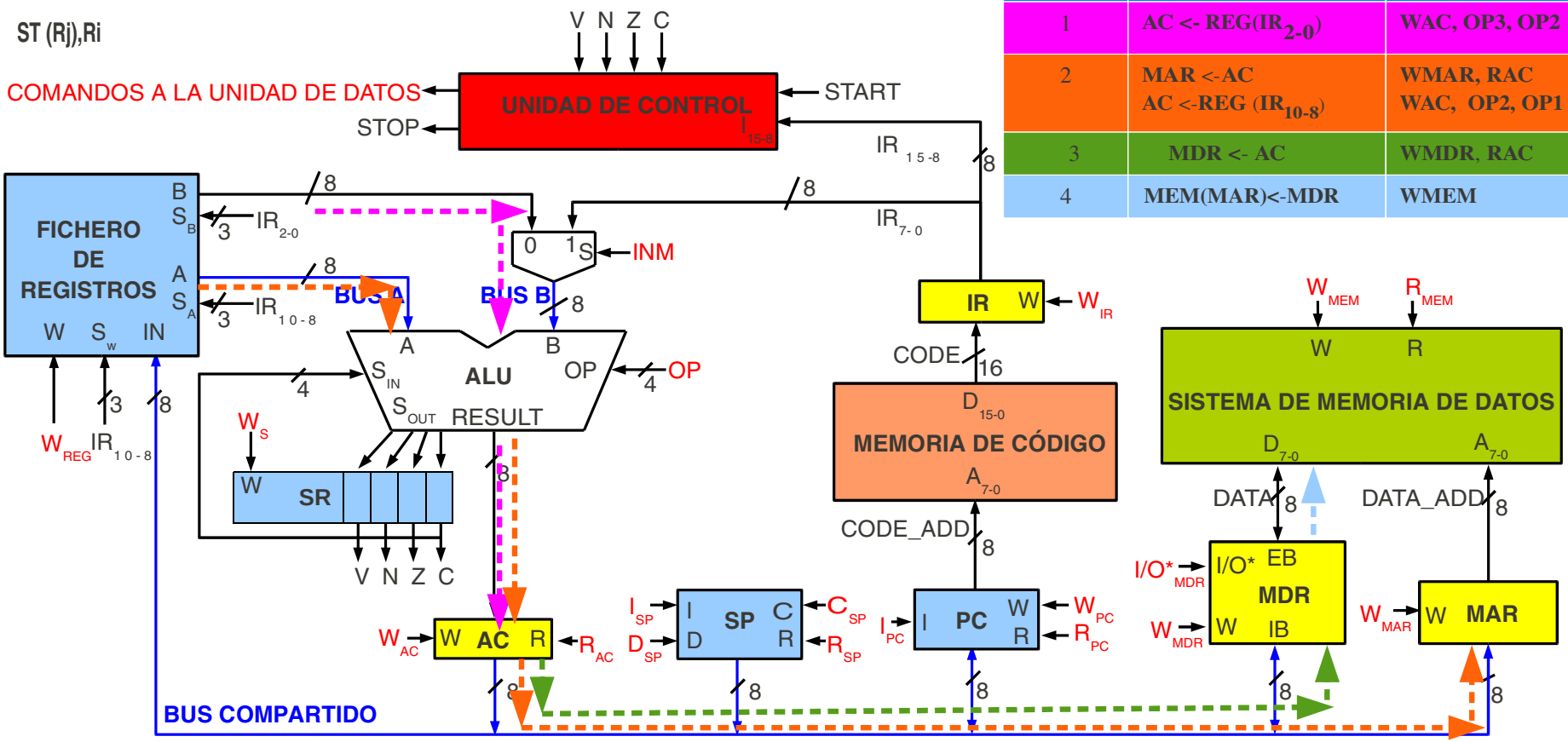
Descomposición de las instrucciones en microoperaciones para algunos casos

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

Ejemplo de instrucción de acceso a memoria
Direccionamiento indirecto de registro

ST (Rj),Ri

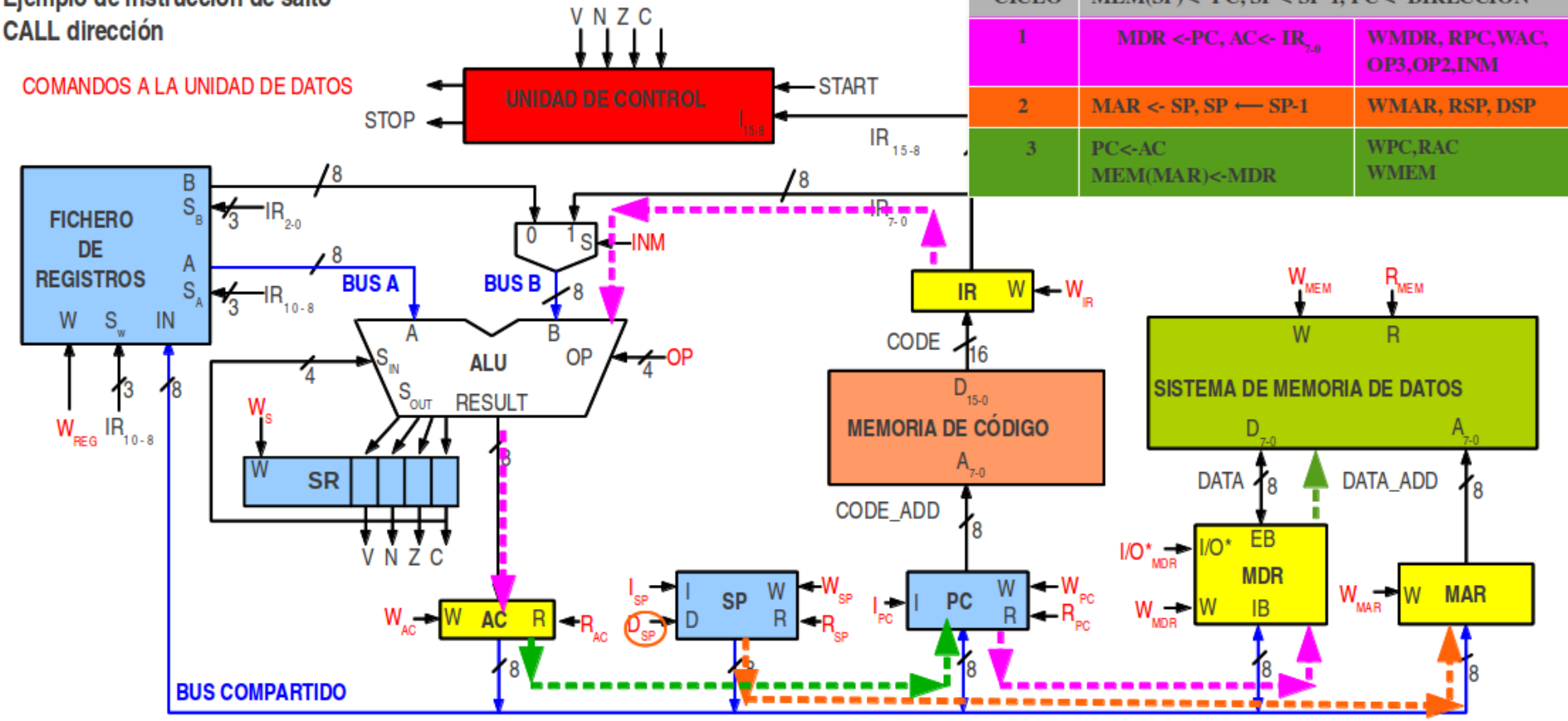
CICLO	MEM(Rj) <- Ri	
1	AC <- REG(IR_{2-0})	WAC, OP3, OP2
2	MAR <-AC AC <-REG (IR_{10-8})	WMAR, RAC WAC, OP2, OP1
3	MDR <- AC	WMDR, RAC
4	MEM(MAR)<-MDR	WMEM



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

Ejemplo de instrucción de salto
CALL dirección

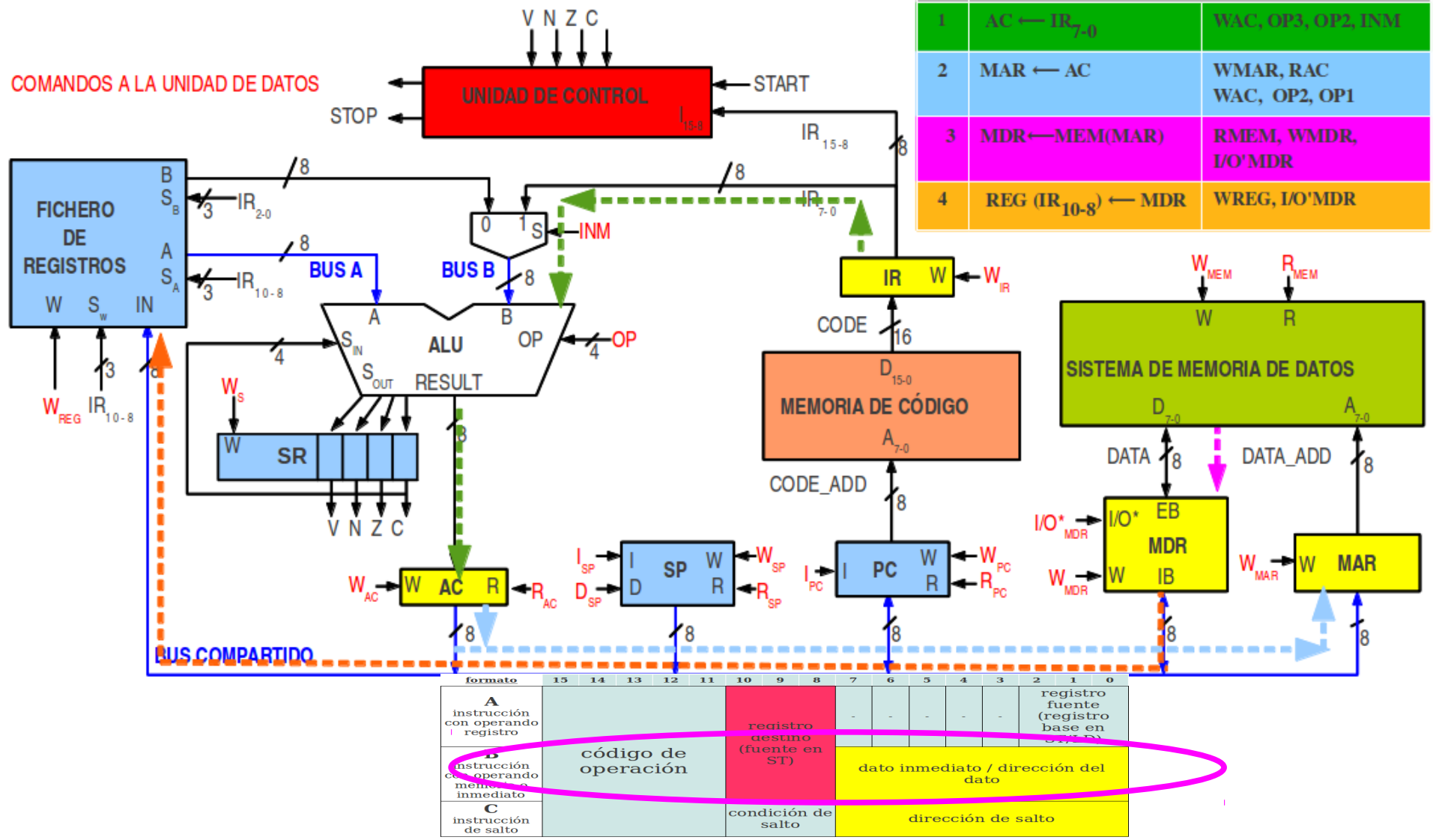
CICLO	MEM(SP) <- PC, SP <-SP-1, PC < -DIRECCION	
1	MDR <-PC, AC<- IR_{7-0}	WMDR, RPC,WAC, OP3,OP2,INM
2	MAR <- SP, SP ⟵ SP-1	WMAR, RSP, DSP
3	PC<-AC MEM(MAR)<-MDR	WPC,RAC WMEM



Ejemplo de instrucción de transferencia desde la memoria con modo de Direccionamiento absoluto

LDS Ri, dirección

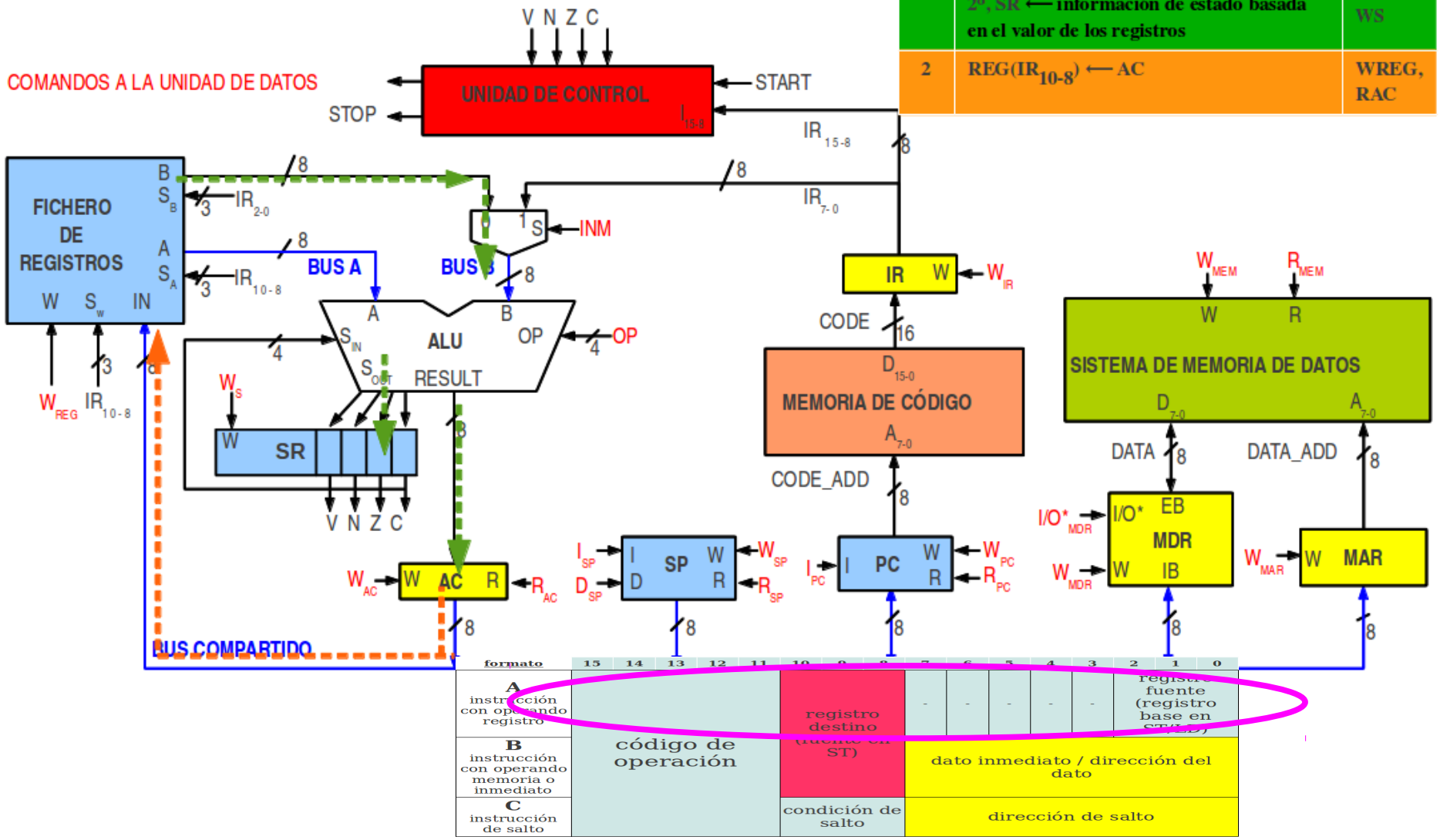
	Ri ⟵ MEM(dirección)	
1	AC ⟵ IR_{7-0}	WAC, OP3, OP2, INM
2	MAR ⟵ AC	WMAR, RAC WAC, OP2, OP1
3	MDR⟵MEM(MAR)	RMEM, WMDR, I/O'MDR
4	REG (IR_{10-8}) ⟵ MDR	WREG, I/O'MDR



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

Ejemplo de instrucción aritmética sobre la pila de Registros
Direccionamiento directo de registro
ADD Ri, Rj

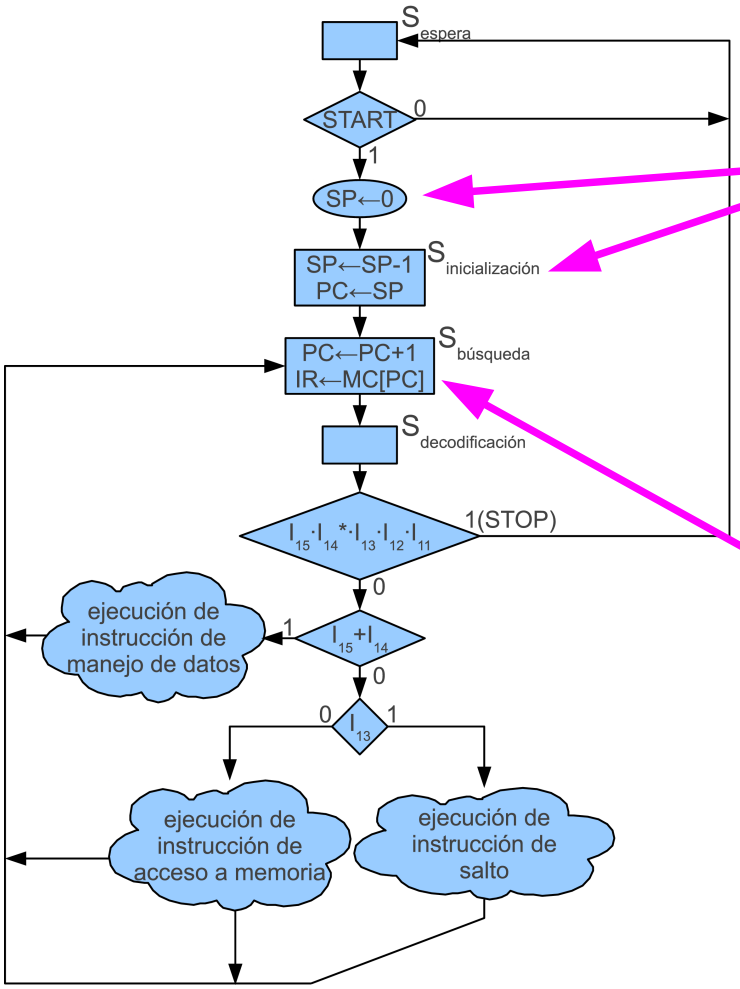
	Ri ⟵ MEM(dirección)	
1	AC ⟵ $REG(IR_{10-8}) + REG(IR_{2-0}) \bmod 2^8$, SR ⟵ información de estado basada en el valor de los registros	WAC, OP3 WS
2	$REG(IR_{10-8})$ ⟵ AC	WREG, RAC



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

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

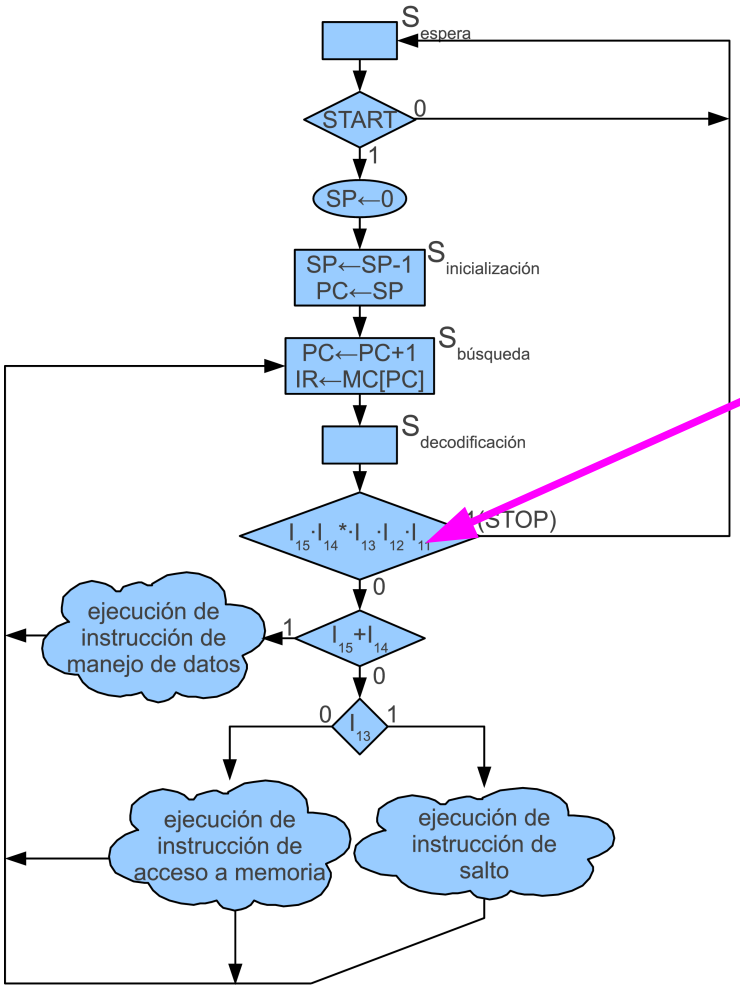
Cartas ASM



Inicialización pone el PC a 0 para que apunte a la primera instrucción guardada en memoria y el SP a la dirección más alta de la memoria a partir de la cual comenzará a rellenarse la pila.

La fase de búsqueda accede a memoria para lectura de la instrucción y guardarla en el registro IR.
Así mismo se incrementa el contador del programa para que apunte a la siguiente instrucción.

Cartas ASM



La fase de ejecución es muy compleja al tratarse de muchas instrucciones. En la carta se establecerán distintos caminos que vendrán dados por el valor del código de operación de la instrucción. Cada camino tendrá una serie de cajas de estados correspondientes a las distintas microoperaciones de las que consta cada instrucción.

Programación

Instrucciones del CS2010

CODOP	TIPO	Sintaxis	Efecto	VNZC
00000	mov. datos	ST (Rb),Rf	MEM(Rb) ffffl←Rf	- - - -
00001	mov. datos	LD Rd,(Rb)	Rd ← MEM(Rb)	- - - -
00010	mov. datos	STS dir, Rf	MEM(dir) ← Rf	- - - -
00011	mov. datos	LDS Rd, dir	Rd ← MEM(dir)	- - - -
00100	salto	CALL dir	push PC, PC←dir	- - - -
00101	salto	RET	pull PC	- - - -
00110	salto	BRxx dir	xx: PC←dir	- - - -
01000	aritmética	ADD Rd,Rf	Rd← Rd + Rf	* * * *
01010	aritmética	SUB Rd,Rf	Rd←Rd - Rf	* * * *
01011	estado	CP Rd,Rf	Rd - Rf	* * * *
01111	mov. datos	MOV Rd,Rf	Rd←Rf	- - - -

Instrucciones del CS2010

CODOP	TIPO	Sintaxis	Efecto	VNZC
10010	estado	CLC	C←0	- - - *
10011	estado	SEC	C←1	- - - *
10100	despl.	ROR Rd	Rd ← SHR (Rd,C)	* * * *
10101	despl.	ROL Rd	Rd ← SHL (Rd,C)	* * * *
10111	parada	STOP	proc. a espera	- - - -
11000	aritmética	ADDI Rd,d8	Rd← Rd + d8	* * * *
11010	aritmética	SUBI Rd,d8	Rd← Rd - d8	* * * *
11011	estado	CPI Rd,d8	Rd – d8	* * * *
11111	mov. datos	LDI Rd,d8	Rd←d8	- - - -
otros códigos	-	-	-	UUUU

Lenguaje ensamblador CS2010

El **lenguaje ensamblador** consta de:

- Instrucciones ejecutables: Son las desarrolladas para el CS2010. Su sintaxis es la del mnemónico que se ha ido presentando.
- Directivas de ensamblado o pseudoinstrucciones: Son instrucciones que ayudan a escribir el programa y a documentarlo, pero que no causan código ejecutable.
- Conjunto de normas (léxicas, semánticas y sintácticas) que lo hacen operativo.

Lenguaje ensamblador CS2010

El **lenguaje ensamblador** consta de:

- Instrucciones ejecutables: Son las desarrolladas para el CS2010. Su sintaxis es la del mnemónico que se ha ido presentando.
- Directivas de ensamblado o pseudoinstrucciones: Son instrucciones que ayudan a escribir el programa y a documentarlo, pero que no causan código ejecutable.
- Conjunto de normas (léxicas, semánticas y sintácticas) que lo hacen operativo.

Directivas del ensamblador CS2010

Las principales **normas y directivas de ensamblado del CS2010** son:

- No distingue mayúsculas de minúsculas
- Comentario hasta final de línea ";" :

```
; Esto es un comentario: elimino ST (R2),R1
```

- Numerales: Formas de dar valores numéricos. Son: decimal con/sin signo (-27, 58); binario de 8 bits precedidos de 0b (0b11001111); hexadecimal de uno o dos dígitos precedidos de 0x (0xB, 0xF4).
- Etiquetas: Cadena alfanumérica que comienza por letra: p.ej. eti3. Pueden definirse con EQU o mediante identificador de línea de instrucción:

```
guarda: ST…        ; asociará a "guarda" el número de instrucción correspondiente a ST…
```

- .EQU <etiqueta>=<numeral>. Asocia un valor a una etiqueta:

```
.EQU n8=25                    ;asocia el 25 a la etiqueta n8
```

- .OPCODE <ByteMS>, <ByteLS>. Escribe directamente un código máquina. El <ByteMS> debe ser un numeral; <ByteLS> puede ser numeral o etiqueta

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

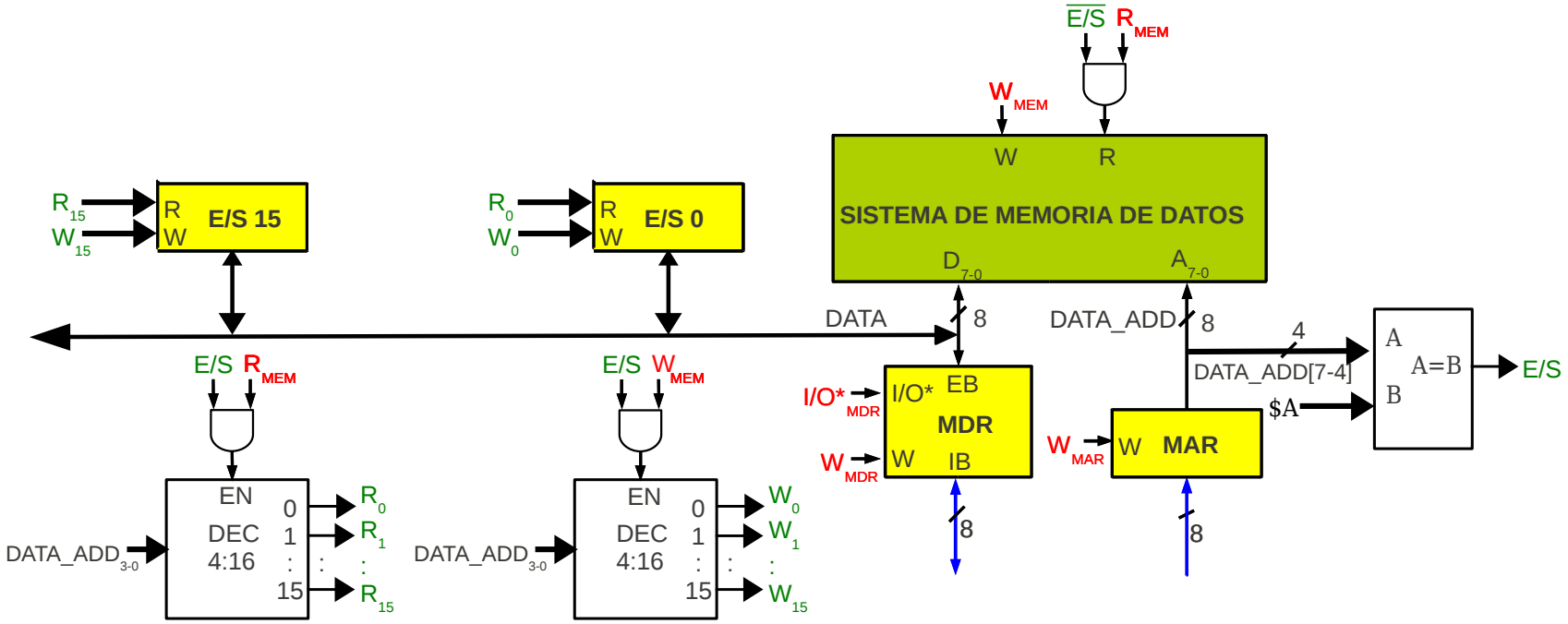
- El CS 2010 no ha permitido solucionar unas de las deficiencias del CS2 planteadas al principio de este tema: la incapacidad de comunicación con el exterior.
- Es decir, no posee capacidad de entrada salida.
- A continuación se expone un procedimiento que permitiría dotarlo de dicha capacidad.

Entrada/Salida

- Desde el punto de vista del programador la memoria es vista como un conjunto de bytes almacenados en direcciones concretas.
- Parte de ese rango de direcciones puede reservarse para que el procesador pueda acceder al exterior (entrada/salida)

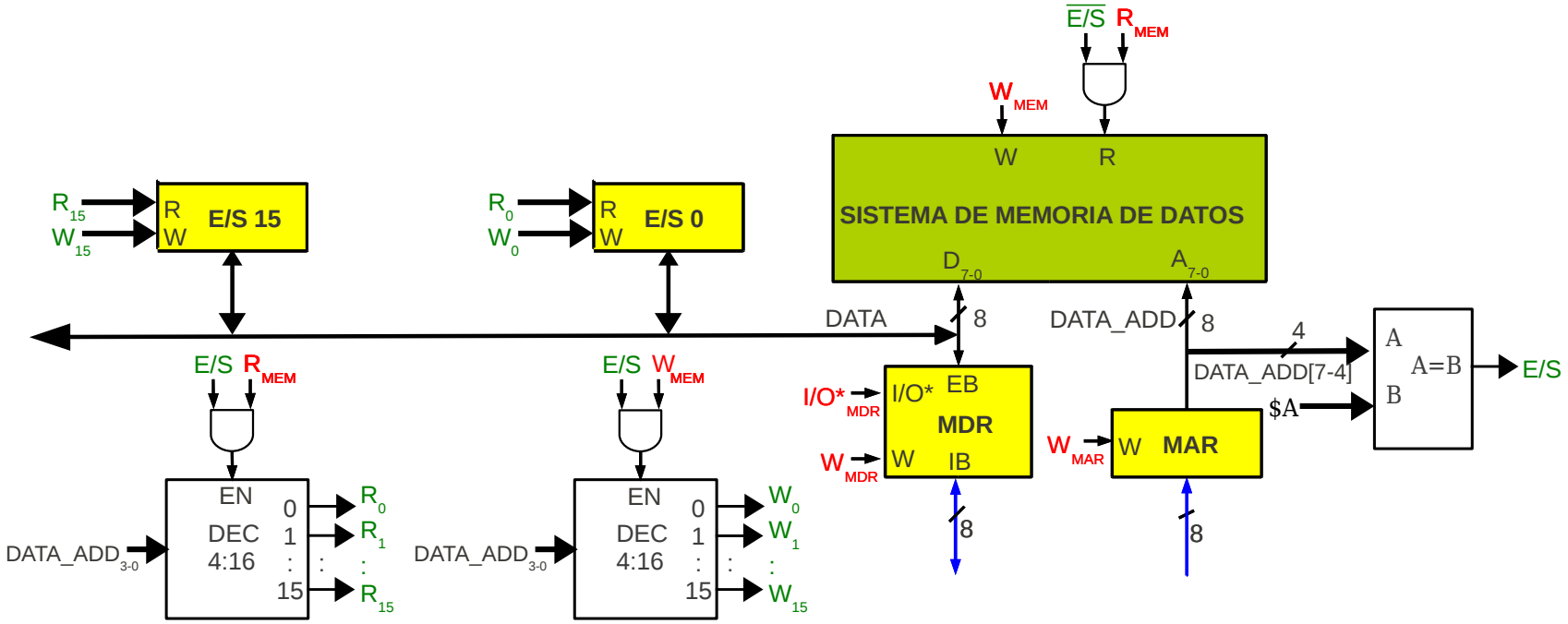
Entrada/Salida

- 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.



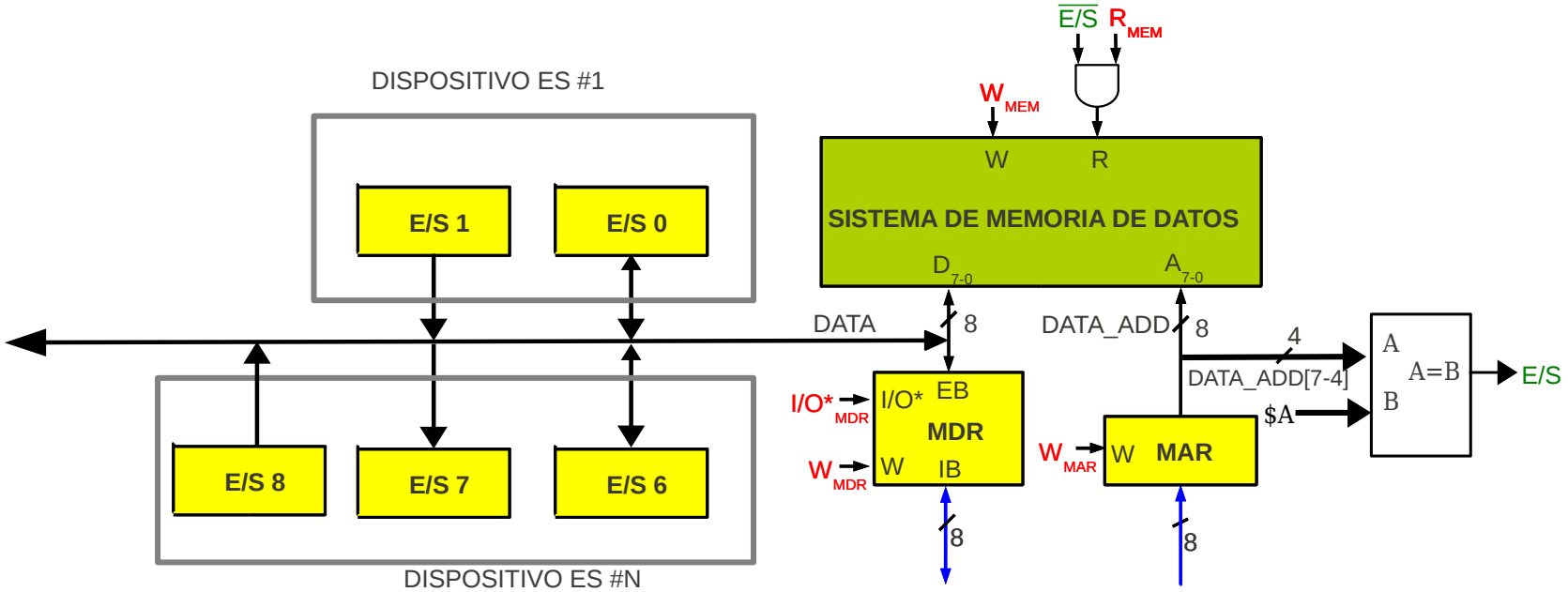
Entrada/Salida

- 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.



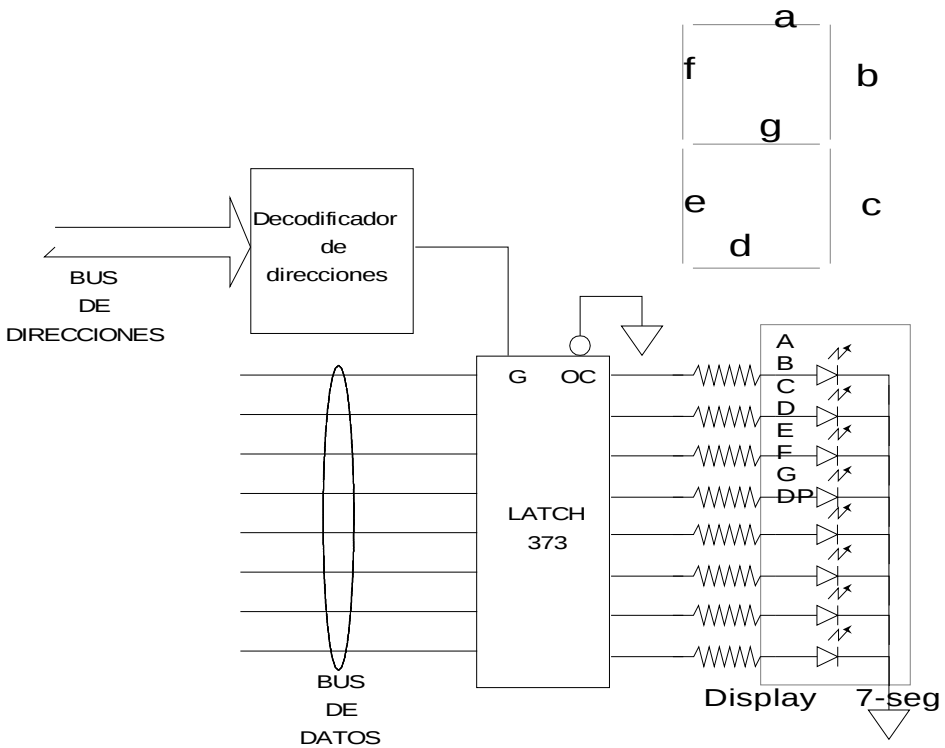
Entrada/Salida

- 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

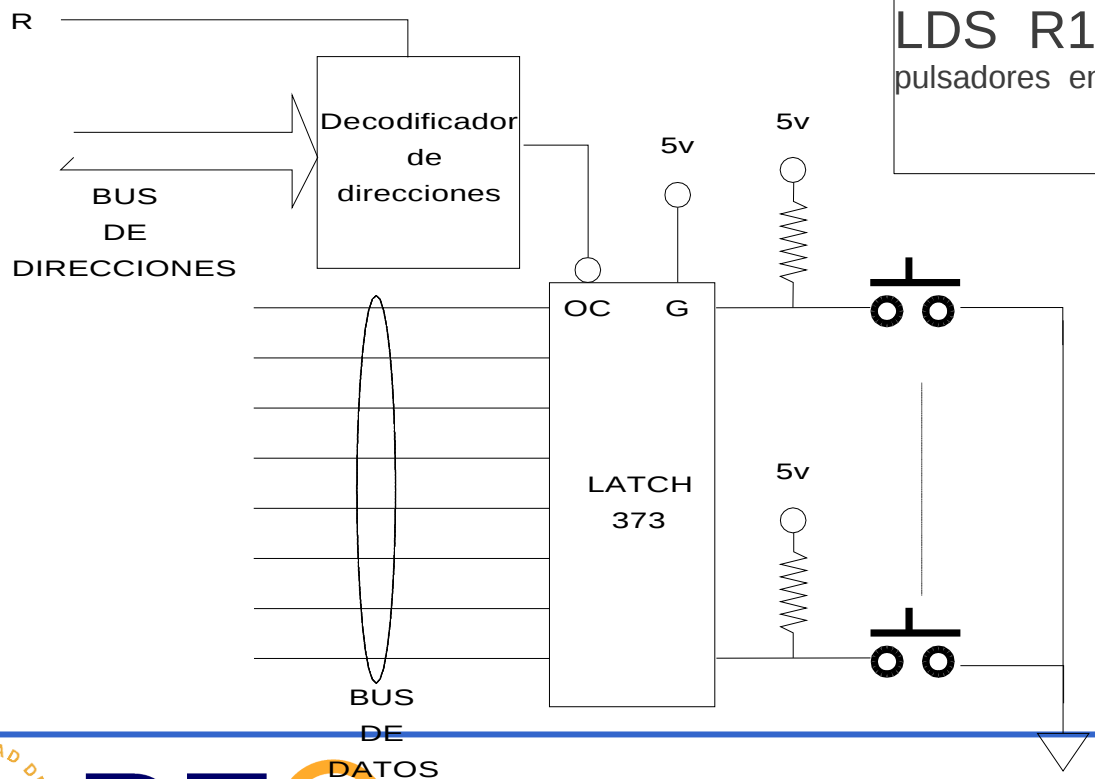
- Dispositivo de salida (display de 7 segmentos)




```
LDI  R1,$FF /*Código 7seg del número 8*/
STS $A0,R1 /*Muestra el número 8
en la dirección  del puerto de E/S*/
```

Entrada/Salida

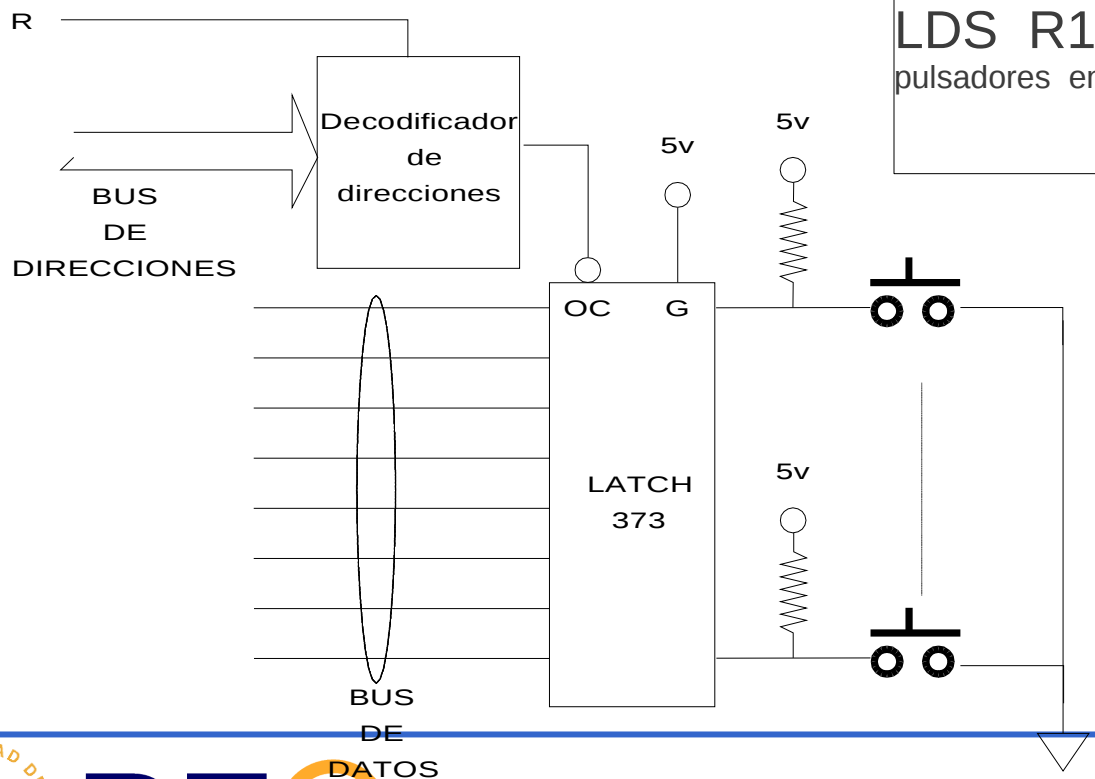
- Dispositivo de entrada (pulsadores)




```
LDS R1, $A1 /*Almacena el estado de los pulsadores en la dirección del puerto de E/S */
```

Entrada/Salida

- Dispositivo de entrada (pulsadores)




```
LDS R1, $A1 /*Almacena el estado de los pulsadores en la dirección del puerto de E/S */
```