Índice

Autor: Alberto J. Molina Cantero
Trabajo basado en el realizado por David Guerrero, Isabel Gómez y Francisco Pérez. Última modificación: 28/03/12

Modelo de programador del CS2010

- 8 Registros de propósito general: R0, R1, R2, R3, R4, R5, R6 y R7 (para suministrar datos a la ALU).
- 3 Registros de propósito específico:
  - **PC (Program counter)**:

    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 (Status Register):**

    Dan información sobre el tipo de resultado de la última operación realizada en la ALU: Z (cero), V (desbordamiento), N (negativo) y C (carry/borrow).
  - **SP (Stack Pointer):** Puntero de pila (memoria LIFO).

    SP apunta hacia la primera posición libre de la pila.

    PUSH (escribir en la pila): MEM(SP) ← REG; SP← SP-1

    POP ó PULL (leer de la pila): SP←SP+1; REG ← [MEM(SP)]

    En CS2010 sólo las instrucciones de subrutinas (CALL, RET) usan SP

Formato de instrucciones del CS2010

- El juego de instrucciones (ISP) (mnemónicos del ensamblador) del CS2010 es un subconjunto del de la arquitectura AVR.

  Los programas en ensamblador del CS2010 pueden reensamblarse y ejecutarse en el AVR excepto los modos indirectos.

  No es compatible a nivel de código máquina

- Instrucciones de 16 bits, con códigos de operación de 5 bits (32 tipos de instrucción diferentes)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
código de operación															

Formato de instrucciones del CS2010

Formatos: Solo hay tres y comparten campos

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							

Códigos de operación CS2010

Bits del código de operación					NEMÓNICO	FORMATO	TIPO	SINTAXIS	EFECTO[1]	VCNZ[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←Rdestino	**----**
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”

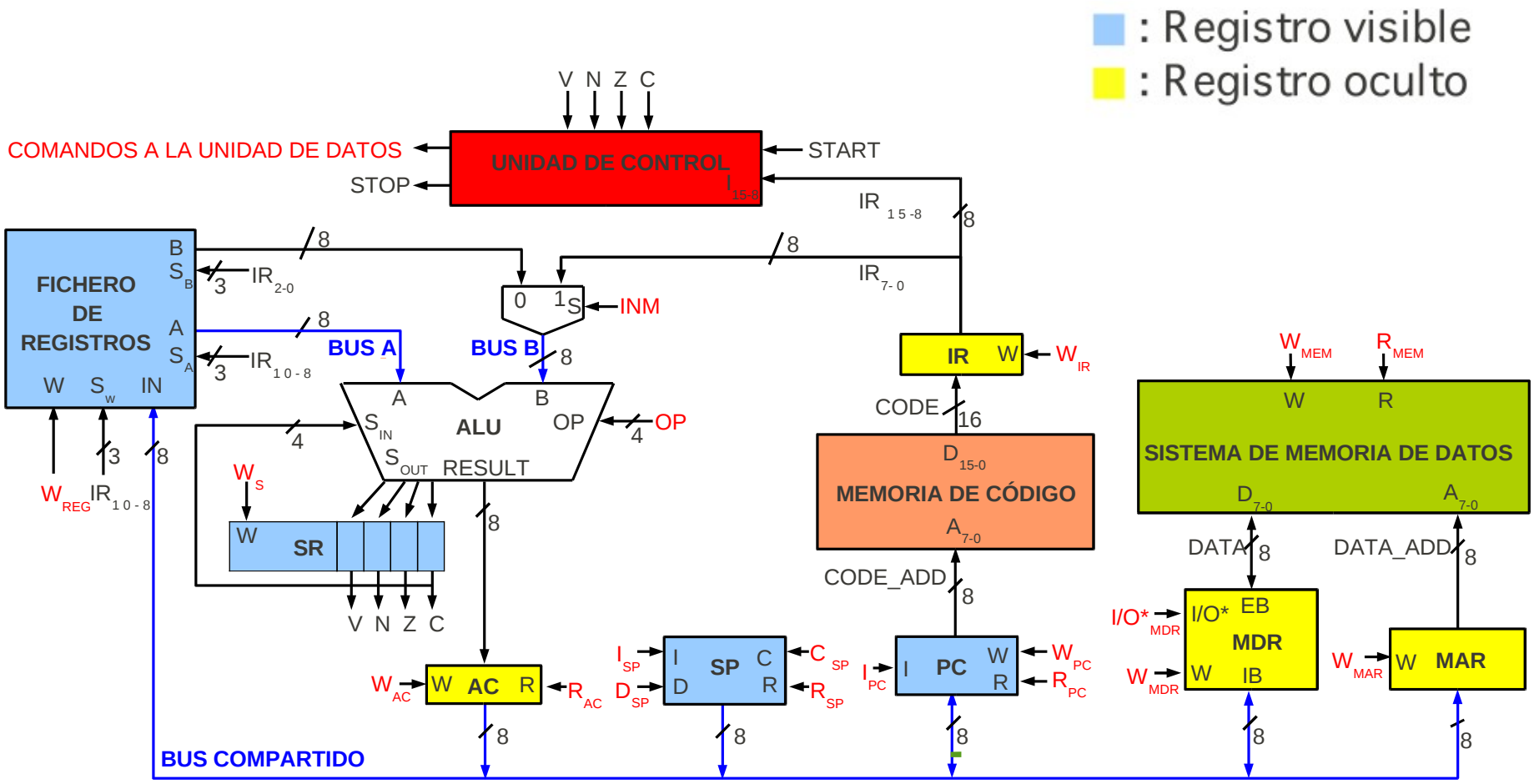
Códigos de operación CS2010

Bits del código de operación					NEMÓNICO	FORMATO	TIPO	SINTAXIS	EFECTO[1]	VCNZ[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←Rdestino	----
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	----

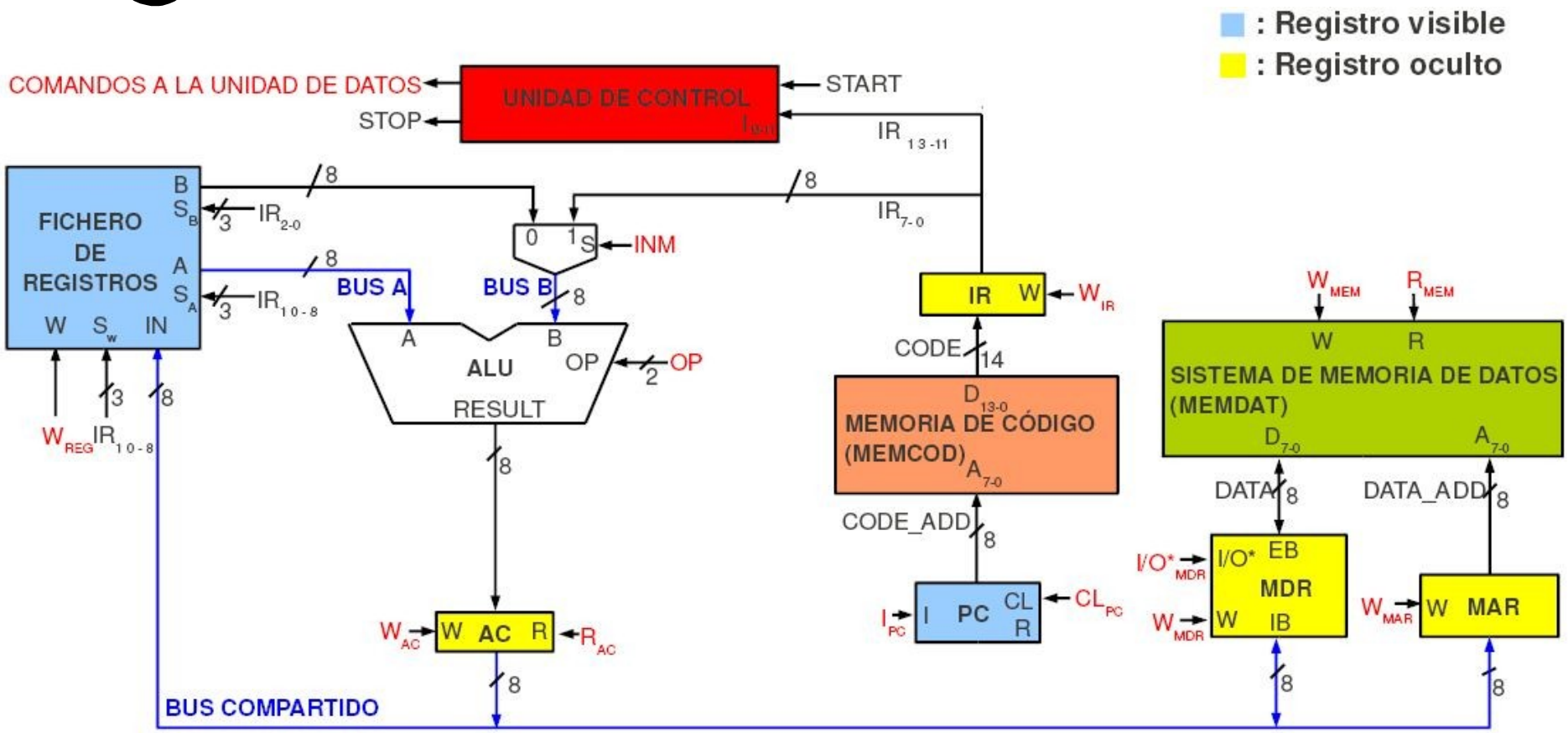
[1] (sin tener en cuenta el registro de estado y el incremento del PC)

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

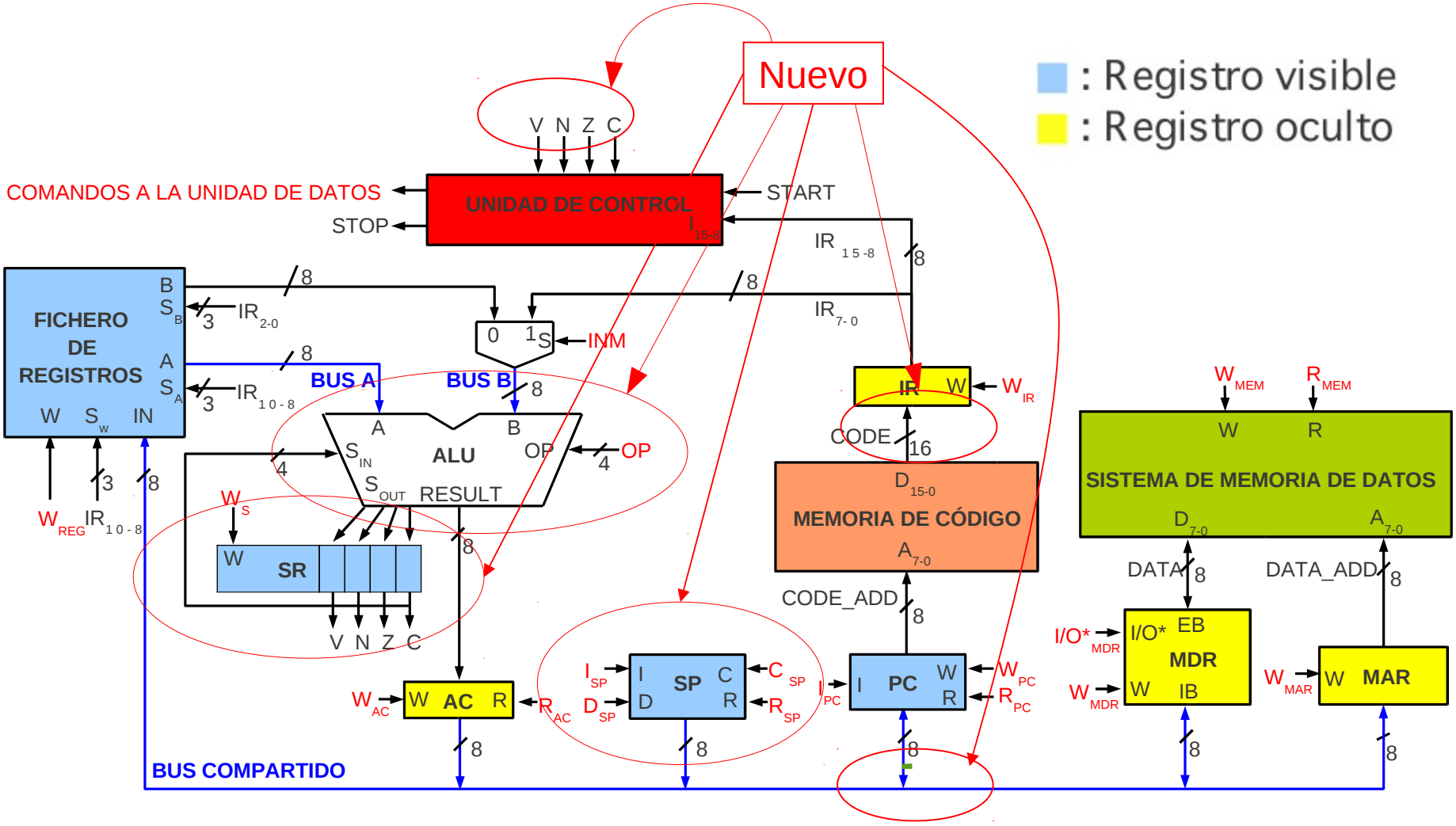
Arquitectura propuesta para el CS2010



Arquitectura del CS2



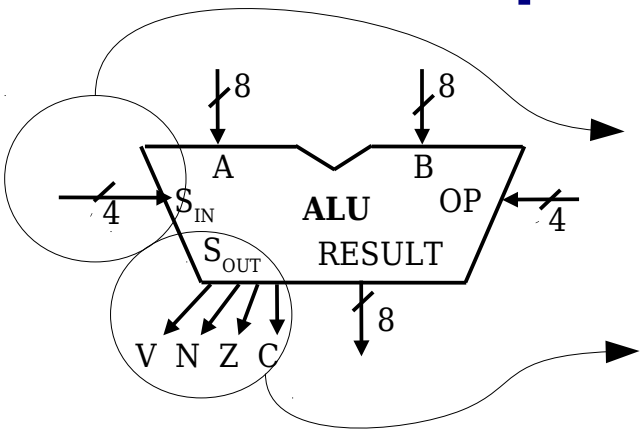
Arquitectura propuesta para el CS2010



Registros “ocultos al programador” de la arquitectura del CS2010

- **IR[16] (Instruction Register):** sirve para almacenar la instrucción que está siendo ejecutada.
- **MDR[8] (Memory Data Register):** Almacenar los datos que se intercambian con la memoria de datos.
- **MAR[8] (Memory Address Register):** Sirve para direccionar la memoria de datos
- **AC[8]:** Almacena el resultado de la operación realizada por la ALU
- **PC[8]**
- **SP[8]**

Descripción de los nuevos componentes del Computador Simple CS2010 (i)



$S_{IN}=\{V_{IN}N_{IN}Z_{IN}C_{IN}\}$ Salida del registro estado que almacena Sout

$S_{OUT}=\{V\ N\ Z\ C\}$

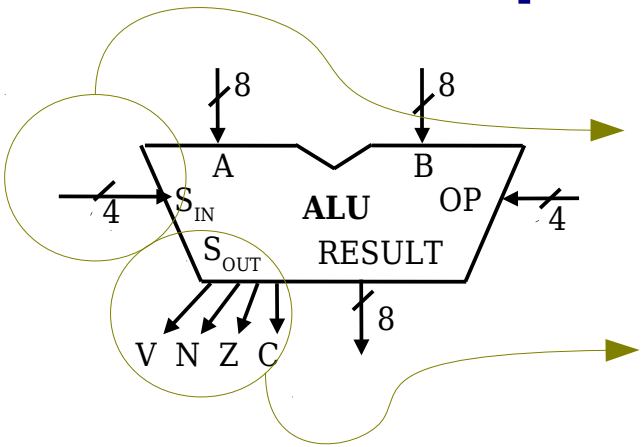
- Bit C: Acarreo de operaciones aritméticas. Se puede poner a 0 o a 1. Se utiliza para operaciones de rotación.

- Bit Z: Se pone a 1 cuando el resultado de una operación es 0

- Bit N: Bit más significativo del resultado.

- Bit V: Informa de overflow en operaciones aritméticas con signo.

Descripción de los nuevos componentes del Computador Simple CS2010 (i)

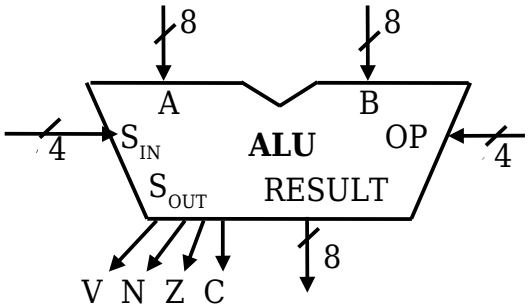


$S_{IN}=\{V_{IN}N_{IN}Z_{IN}C_{IN}\}$ Salida del registro De estado, SR, que almacena Sout.

$S_{OUT}=\{V\ N\ Z\ C\}$

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	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	overflow(A-B)	$RESULT_7$	NOT $OR_{i=0}^{7}(RESULT_i)$	borrow(A-B)
1	1	-	-	B	-	-	-	-

Descripción de los nuevos componentes del Computador Simple CS2010 (i)

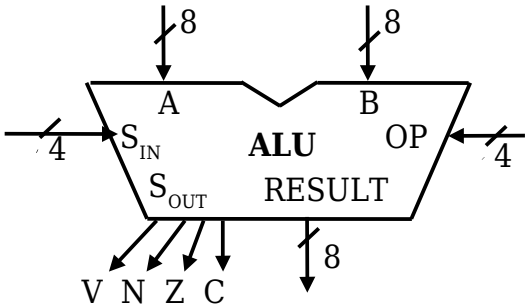


OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
0	0	-	0	-	V_{IN}	N_{IN}	Z_{IN}	0
0	0	1	1	-	V_{IN}	N_{IN}	Z_{IN}	1

- Permite poner a uno o a cero el biestable de acarreo, C, dejando intactos los restantes banderines: V, N,Z

- Instrucciones SEC,CLC

Descripción de los nuevos componentes del Computador Simple CS2010 (i)



OP_3	OP_2	OP_1	OP_0	RESULT=	V_{OUT}=	N_{OUT}=	Z_{OUT}=	C_{OUT}=
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

-Desplaza el contenido de A (derecha o izquierda) con Cin.

- Cout = A0 o A7, Z =1 si resultado es 0, N es el bit 7 del resultado y V si el bit de signo cambia de valor.

- Instrucciones ROR, ROL

Ejemplos de ejecución de algunas instrucciones

Indique el resultado de las siguientes instrucciones y el valor que tomarían los banderines tras su ejecución. Para todas ellas el contenido de los registros R3 y R7 son $80 y $7F y los valores iniciales de los banderines son: CNZV=1000

a) ADD R3,R7 R3=_____ R7=_____ CNZV= _____
b) SUB R3,R7 R3=_____ R7=_____ CNZV= _____
c) SUB R7,R3 R3=_____ R7=_____ CNZV= _____
d) SUBI R8,$80 R3=_____ R7=_____ CNZV= _____
e) CP R3,R7 R3=_____ R7=_____ CNZV= _____
f) CPI R8,$80 R3=_____ R7=_____ CNZV= _____
g)ROR R7 R3=_____ R7=_____ CNZV= _____
h) ROL R3 R3=_____ R7=_____ CNZV= _____
i) LDI R3,1 R3=_____ R7=_____ CNZV= _____
j) MOV R3,R7 R3=_____ R7=_____ CNZV= _____
k) CLC R3=_____ R7=_____ CNZV= _____
l) SEC R3=_____ R7=_____ CNZV= _____

Formato de instrucciones del CS2010

Hay tres y comparten campos

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							

Códigos de condición de la instrucción de salto condicional BRXX *(Branch if…)*

Los bits $IR_{10}IR_9I\ R_8$ codifican la condición de salto *xx*.

I_{10}	I_9	I_8	CONDICIÓN	nmónico(s) de la condición xx =	notas
0	0	0	Z	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 base 2 sin signo
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

Instrucciones de salto condicional: BREQ, BRCS (BRLO), BRVS, BRLT.

En programación las instrucciones de salto condicional viene tras una instrucción de comparación (CP, CPI) las cuales actualizan los banderines del registro de estado sin modificar los operandos.

Ejemplos de uso de las instrucciones de salto condicional

Comparación en igualdad: CP, BREQ

- La instrucción CP R1,R0 resta R0 de R1 y afecta a los flags.
- Si Z=1, implica que el resultado de la instrucción anterior fue 0.
- Por tanto R1==R0.

Pseudocódigo	**Ensamblador**	
		CP R1,R0
Si (A==B)		BREQ R1igualR0
{..cuerpo A=B... }	R1noesigualR0:	{...cuerpo A!=B...}
sino		JMP Fsi
{..cuerpo A !=B }	R1igualR0:	{..cuerpo A=B...}
fsi	Fsi:	

Ejemplos de uso de las instrucciones de salto condicional

Comparación en desigualdad (números sin signo): CP, BRCS (BRLO)

- La instrucción CP R1,R0 resta R0 de R1 y afecta a los flags.
- Si C=1, implica que el resultado de la instrucción anterior fue en desigualdad.
- Por tanto R1<R0.

Pseudocódigo	**Ensamblador**	
		CP R1,R0
Si (A<B)		BRCS R1menorqR0
{..cuerpo A<B... }	R1mayorigualqR0:	{...cuerpo A>=B...}
sino		JMP Fsi
{..cuerpo A >=B }	R1menorqR0:	{..cuerpo A<B...}
fsi	Fsi:	

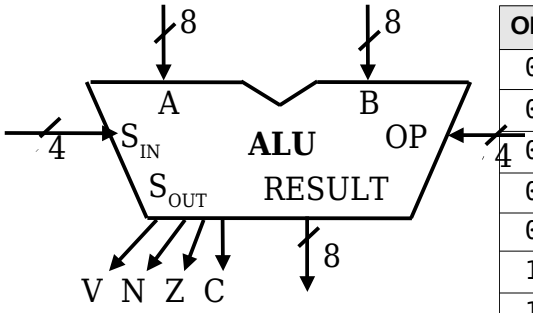
Ejemplos de uso de las instrucciones de salto condicional

Comparación en desigualdad (números con signo): CP, BRLT

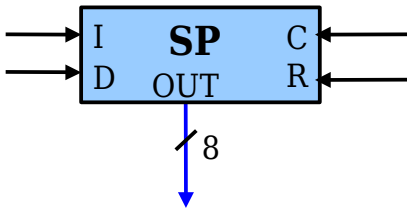
- La instrucción CP R1,R0 resta R0 de R1 y afecta a los flags.
- N xor V representa el signo verdadero del resultado.
- Si V=0, el signo verdadero es N.
- Si V=1, el signo verdadero es el contrario de N.
- Si se restan dos números con signo, R0-R1, si el resultado verdadero es negativo, el número R1 es mayor que R0.

Pseudocódigo	**Ensamblador**	
		CP R1,R0
Si (A<B)		BRLT R1menorqR0
{..cuerpo A<B... }	R1mayorigualqR0:	{...cuerpo R1>=R0...}
sino		JMP Fsi
{..cuerpo A >=B }	R1menorqR0:	{..cuerpo R1<R0...}
fsi	Fsi:	

Descripción de los nuevos componentes del Computador Simple CS2010 (ii)



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	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)

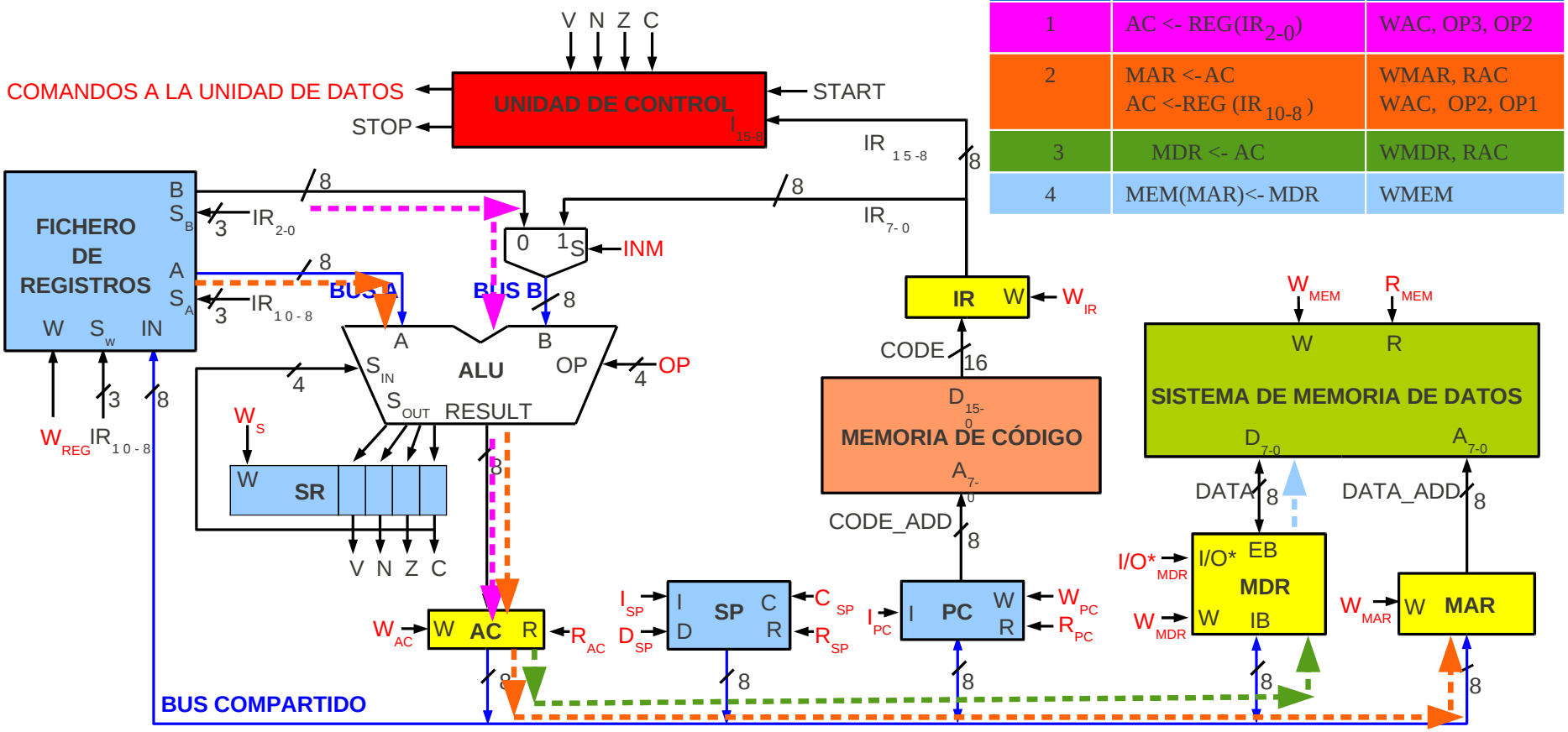


I D C R	SP ←	OUT:=
0 0 0 0	SP	HI
0 0 0 1	SP	SP
0 0 1 0	0	HI
0 1 0 0	SP-1	HI
1 0 0 0	SP+1	HI
OTRAS	PROHIBIDAS	

- La descripción del STATUS REGISTER (SR) es idéntica a la del MAR

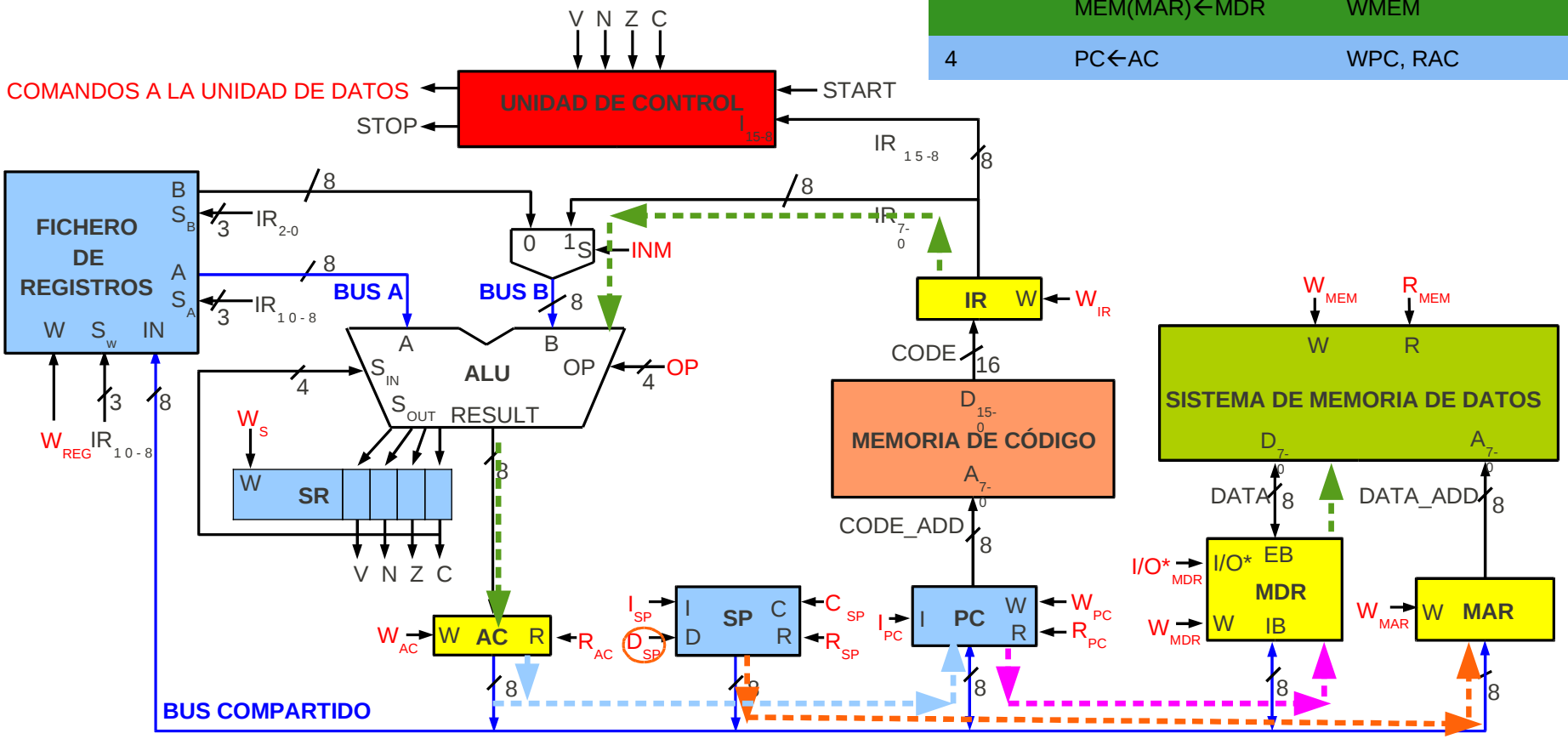
Ejemplo de secuencia de microinstrucciones de la instrucción ST (Rj),Ri *(Store in memory)*

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



Ejemplo de secuencia de microinstrucciones de la instrucción CALL dirección *(Llamada a subrutina)*

CICLO	MEM(SP) ← PC; SP← SP-1; PC← dirección	
1	MDR ←PC	WMDR, RPC
2	MAR ← SP, SP←SP-1	WMAR, RSP, DSP
3	AC←R 7-0 MEM(MAR)←MDR	INM, OP3, OP2, WAC WMEM
4	PC←AC	WPC, RAC



Nota

La descripción RT del resto de las instrucciones se encuentra en un documento anexo a éste.

El alumno debe trabajar con ese documento.

Ejemplo de programación en ensamblador del CS2010

Ejemplo 1. *Escriba una subrutina para el cálculo de la multiplicación mediante el algoritmo de sumas sucesivas.*

```
; Ejemplo 1:
; multiplica los datos de los registros R1 y R2
; y devuelve el resultado en R0 (truncando a 8 bits)
; sumo el dato de R1 tantas veces como diga R2

MULT:       LDI R0,0        ;Inicializa R0 a 0
            CPI R2,0        ;Comprueba si multiplicador es 0
            BREQ RETORNA    ;Salta al final de la rutina si R2=0
BUCLE:      ADD R0,R1       ;A R0 se le añade el multiplicando
            SUBI R2,1       ;Decrementa multiplicador R2 en 1
            BREQ RETORNA    ;Salta al final de la rutina si R2=0
            JMP BUCLE       ;Repite bucle
RETORNA:    RET             ;Retorno de subrutina
```

Ejemplo de programación en ensamblador del CS2010

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

```
; Ejemplo 2:
; devuelve en R0 el mayor de los datos almacenados
; en los registros R1 y R2.

MAX:      CP R1,R2        ;Evalúa R1-R2
          BRLT R2MAYOR    ;Si N xor V = 1 entonces R2>R1
          MOV R0,R1       ;Copia R1 en R0
          RET             ;Salida de la subrutina
R2MAYOR:  MOV R0,R2       ;Copia R2 en R0
          RET             ;Salida de la subrutina
```

Ejemplo de programación en ensamblador del CS2010

Ejemplo 3. *Escriba una subrutina que devuelva el mayor de dos números sin signo*

```
; Ejemplo 3:
; devuelve en R0 el mayor de los datos almacenados
; en los registros R1 y R2.

MAX:      CP R1,R2        ;Evalúa R1-R2
          BRCS R2MAYOR    ;Si C = 1 entonces R2>R1
          MOV R0,R1       ;Copia R1 en R0
          RET             ;Salida de la subrutina
R2MAYOR:  MOV R0,R2       ;Copia R2 en R0
          RET             ;Salida de la subrutina
```

Ejemplo de programación en ensamblador del CS2010

Ejemplo 4. *Escriba una subrutina que escriba, en orden descendente, los números del 100 al 1 en una tabla situada a partir de la posición de memoria 123*

```
; Ejemplo 4:
; Rellena una tabla en orden descendente, del 100 a 1
; la tabla empieza en la dirección 123

CUENTA:     LDI R0,100          ;Cargo en R0 el número 100
            LDI R1,123          ;Cargo en R1 la dirección donde empieza la tabla.

BUCLE:      ST (R1),R0          ;Muevo a la dirección de memoria que está en R1
                                ;el dato almacenado en R0
            ADDI R1,1           ;Incremento R1 para que éste contenga la
                                ;dirección ;siguiente
            SUBI R0,1           ;Decremento el valor de R0
            BREQ RETORNA        ;Si R0=0, hemos terminado. Saltamos a Retorna
            JMP BUCLE           ;Salta a la línea con la etiquete bucle
RETORNA:    RET                 ;Finalizamos el bucle.
```

Índice