Universidad Europea
LAUREATE INTERNATIONAL UNIVERSITIES

OPERACIONES BÁSICAS Y DIRECCIONAMIENTO

Índice

Presentación	4
Flags. El SREG: registro de estado	5
Suma/resta	7
Instrucciones de suma: ADD/ADC	9
Instrucciones de resta: SUB/SBC	11
Instrucciones de comparación: CP/CPC/IPC	14
Instrucciones lógicas: AND/ANDI/EOR/OR/ORI/TST	16
Otras instrucciones de importancia	19
Instrucciones para manipular el registro SREG	21
Desplazamiento y rotación de bits	23
Instrucciones de multiplicación y división	25
Resumen	27
Referencias bibliográficas	28

Presentación

El **movimiento de datos**, es la **operación más asidua** en cualquier programa, seguido muy de cerca por la de **manipulación de datos**. De ahí la importancia de que estas instrucciones sean muy optimas en tiempo de ejecución y consumo de memoria.

La manipulación de datos se realiza en la **ALU** (Arithmetic and Logic Unit) de la CPU:

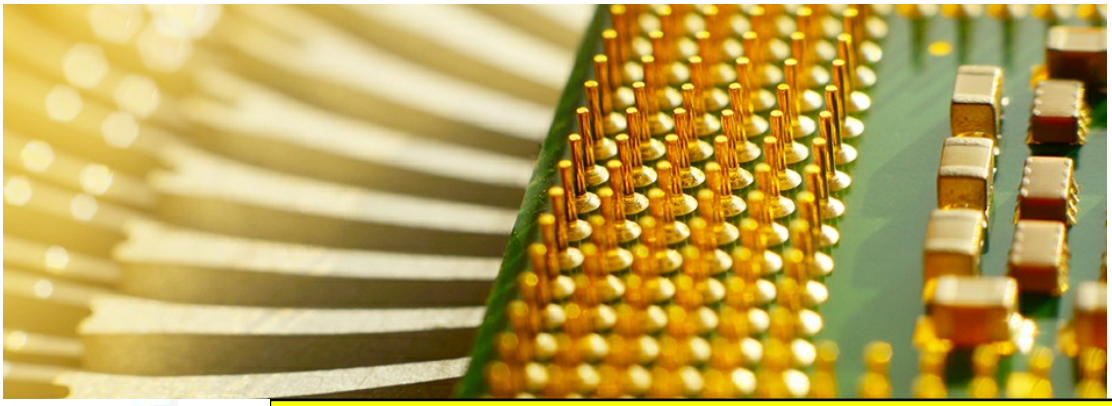
- La ALU, es, principalmente, un montón de **circuitos electrónicos combinacionales** que realizan todas las operaciones que necesita el procesador.
- Los **datos de entrada** con los que procesa la ALU, suelen estar almacenados, por lo general, en registros.

Cada vez que se realiza una operación en la ALU, esta **registra el resultado del proceso** (no el valor resultado de la operación, sino el resultado de cómo ha terminado y cuáles han sido las características de la operación) en un registro especial por medio de los bits (flags=banderas) en la **PSW** (Process Status Word):

- La información que se puede guardar en este registro de Status, son cosas del estilo "el resultado ha sido **cero**", "**positivo**", "**negativo**", "ha habido **desbordamiento**", "**carry**", etc.

Los **objetivos** que se pretenden alcanzar con este recurso son los siguientes:

- Diferenciar las **operaciones aritméticas** en ensamblador.
- Aprender a **alterar la ejecución secuencial** de un programa mediante las instrucciones de comparación.



Flags. El SREG: registro de estado

El registro de estado, “SREG” para Atmega328 incluye esas señales de:

Bit 0x3F (0x5F)	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG

Por defecto, todas las señales **se inicializan a 0** (=cero). Por lo que un “0” corresponde a que el bit no está activo o la condición no se cumple.

El **significado** de cada bit de bandera es el siguiente:

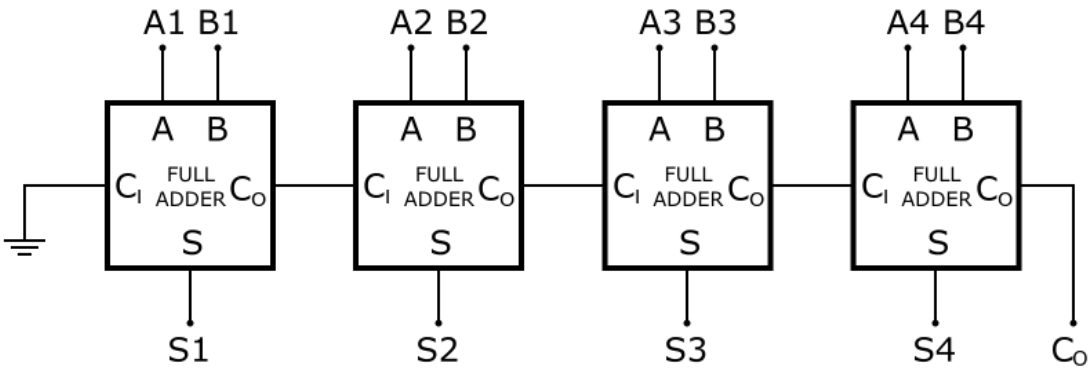
- C: carry. Indica que se ha producido acarreo en una operación aritmética o lógica.
- Z: cero. Indica un resultado de cero en una operación aritmética o lógica.
- N: negativo. Indica un resultado negativo en una operación aritmética o lógica.
- V: complemento a 2 de desbordamiento. Indica un desbordamiento en las operaciones de complemento a 2.
- S: bit de signo. Indica señal resultado de la operación. Siempre ha calculado como N $\oplus$ V.
- H: medio carry. Útil para las operaciones BCD (decimal codificado en binario).
- T: bit de almacenamiento de copia. Es una memoria de 1 bit utilizado por las instrucciones a nivel de bits.
- I: activación de interrupciones globales. No ocurren interrupciones si es 0, en caso de valor 1, las interrupciones están activas.

Dos instrucciones importantes, de manejo de registros, que veremos a continuación, son las instrucciones encargadas de **inicializar registros**. Estas instrucciones son las siguientes:

CLR (CLear Register)	CLR (CLear Register) **inicializa a ceros** el registro. $0010 \mid 01d_4d_4 \mid d_3d_2d_1d_0 \mid d_3d_2d_1d_0$ Su **formato** es el siguiente: CLR Rd→ ;Rd←0 • Donde Rd debe ser un registro de R0 .... R31. • Como información curiosa, decir que esta instrucción, **realmente no existe**. Ensamblador lo permite, pero la reemplaza por la instrucción EOR (Exclusive OR) con el formato EOR Rd, Rd (→ Rd=Rd⊕Rd). • Flags que actualiza: **S = 0, N = 0, V = 0, Z = 1.**
SER (SEt Register)	Carga un 0xFF en el registro Rd, es decir, lo **inicializa todo a unos**. $1110 \mid 1111 \mid d_3d_2d_1d_0 \mid 1111$ Su **formato** es el siguiente: SER Rd ;Rd←0xFF • Donde Rd debe ser un registro de R16 .... R31 (no utilizable para R0 ... R15). • Flags que actualiza: **no afectados**.

Suma/resta

La operación aritmética más básica es la **suma**. Para implementar esta operación, se utiliza el **sumador completo de un bit**, unidos en cascada para conseguir sumadores (/restadores) de N bits.

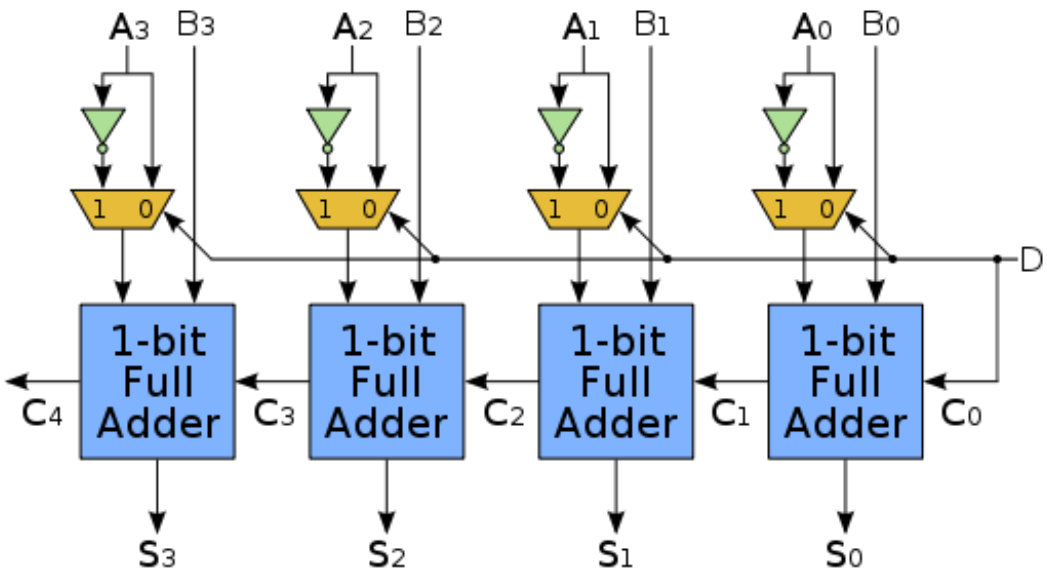


Nota: a menudo, las operaciones de comparación se realizan mediante una resta, teniendo en cuenta que el resultado de a-b es:

- 0 si a = b.
- Positivo si a > b.
- Negativo si a < b.

Por lo general, la resta se implementa como una **adición del número negativo correspondiente**. Así "a – b" se implementa como "a + (-b)".

Nótese que lo que se hace es el Ca2 de "b" y luego se suma.



Instrucciones de suma: ADD/ADC

La suma estándar se realiza usando las instrucciones **ADD** o **ADC**:

ADD (ADDition)	Suma **sin entrada de acarreo**. $0000 \mid 11r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su **formato** es el siguiente: ADD Rd, Rr ;Rd ← Rd + Rr • Suma los registros Rd y Rr y pone el resultado en el registro Rd.
ADC (ADdition with Carry)	Suma **con carry de entrada**. Considera el carry almacenado en el registro SREG del resultado de la operación anterior: $0001 \mid 11r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su **formato** es el que se muestra a continuación: ADC Rd, Rr ;Rd ← Rd + Rr + C • Suma dos registros y el flag C anterior, guarda resultado en el registro Rd.

Para **ambas instrucciones**:

- Flags que actualiza: afectar a las señales **H, S, V, N, Z, C.**
- Rd y Rr pueden ser un registro de **R0 .... R31**.
- Se puede **sumar un registro consigo mismo**.

Ejemplo

Código ensamblador para la expresión “b = a + 7”

Código ensamblador para la expresión “b = a + 7”

```
.CSEG
start:
        LDI R16, 7          ;put the 7 into a register
        LDS R0, a           ;copy ‘a’ to a register
        ADD R0, R16         ;adds ‘a’+7, result in R0
        STS b, R0           ;copy result to ‘b’
```

Ejemplo

Sumar el número 0x3AF2 y 0x1C7D

Sumar el número 0x3AF2 y 0x1C7D

Sumar el número 0x3AF2 y 0x1C7D (ambos 16 bits), y almacenar el resultado en la variable var1:

```
.DSEG                       ;indicate we are going to use data memory addresses
        var1: .BYTE 2       ;memory address and reserve 2 bytes

.CSEG
start:
        LDI R16, 0xF2
        LDI R17, 0x3A       ;1st number in R17:R16
        LDI R18,0x7D
        LDI R19,0x1C        ;2nd number in R19:R18
        ADD R16, R18        ;adds low byte, result in R16
        ADC R17, R19        ;adds high byte plus low byte carry, result in R17
                            ;now the result is in R17:R16. Then copy it into var_1
        STS var_1, R16      ;copy low byte to var_1
        STS var_1+1, R17    ;copy high byte to var_1
```

/

Instrucciones de resta: SUB/SBC

La resta estándar se realiza utilizando las instrucciones **SUB/SBC**:

SUB (SUBstraction)	Resta **sin entrada de acarreo**. $0001 \mid 10r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su **formato** es el siguiente: SUB Rd, Rr ;Rd ← Rd – Rr • Resta dos registros y pone el resultado en el registro Rd.
SBC (SuBtraction whith Carry)	Resta con acarreo "C" de la operación anterior. $0000 \mid 10r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su **formato** es el que se muestra a continuación: SBC Rd, Rr ;Rd ← Rd - Rr – C • Resta dos registros Rd y Rr y al resultado le resta Carry, pone el resultado en el registro Rd.

Para ambas instrucciones:

- Flags que actualiza: a las señales **H, S, V, N, Z, C**.
- Rd y Rr pueden ser un registro de **R0 .... R31**.
- Se puede **sumar un registro consigo mismo**.

Instrucciones de resta con Inmediatos. SUBI/SBCI

Las instrucciones de resta también pueden funcionar con **datos constantes**:

SUBI (SUBstract Immediate)	Resta un **valor constante sin carry**. Su **formato** es el siguiente: SUBI Rd, k ;Rd ← Rd – k • Resta la constante k (valor inmediato) al registro Rd, pone el resultado en Rd.
SBCI (SuBtract Immediate with Carry)	Resta un valor constante, considerando el carry anterior. Su **formato** es el que se muestra a continuación: SBCI Rd, k ;Rd ← Rd - k – C: • Resta la constante k a Rd, y después el flag Carry "C", pone el resultado en Rd.

Para ambas instrucciones:

- Flags que actualiza: a las señales H, S, V, N, Z, C.
- Rd pueden ser un registro de R16 .... R31.
- K debe ser: 0 < k < 255.

Ejemplo Código ensamblador para la expresión b = a – 7

Código ensamblador para la expresión “b = a + 7”

```
DSEG  ;indicate we are going to use data memory addresses
 a:  .BYTE 1     ; 1 byte var
 b:  .BYTE 1     ; 1 byte var

.CSEG
start:
 LDI R16, 7     ;put the 7 into a register
 LDS R0, a     ;copy ‘a’ to a register
 SUB R0, R16     ;calculates ‘a’-7, result in R0
 STS b, R0     ;copy result to ‘b’

```

Calcular la resta 0x3AF2 - 0x1C7D (de 16 bits) y almacenar el resultado en la variable var1:

```
DSEG              ;indicate we are going to use data memory addresses
 var1:  .BYTE 2    ; memory address and reserve 2 bytes

.CSEG
start:
 LDI R16, 0xF2
 LDI R17, 0x3A ;1st number in R17:R16
 LDI R18,0x7D
 LDI R19,0x1C ;2nd number in R19:R18
 SUB R16, R18 ;subtracts low byte, result in R16
 SBC R17, R19 ;subtracts high byte plus low byte carry
        ;now the result is in R17:R16. Then copy it into var1
 STS var1, R16 ;copy low byte to var1
 STS var1+1, R17 ;copy high byte to var1
```

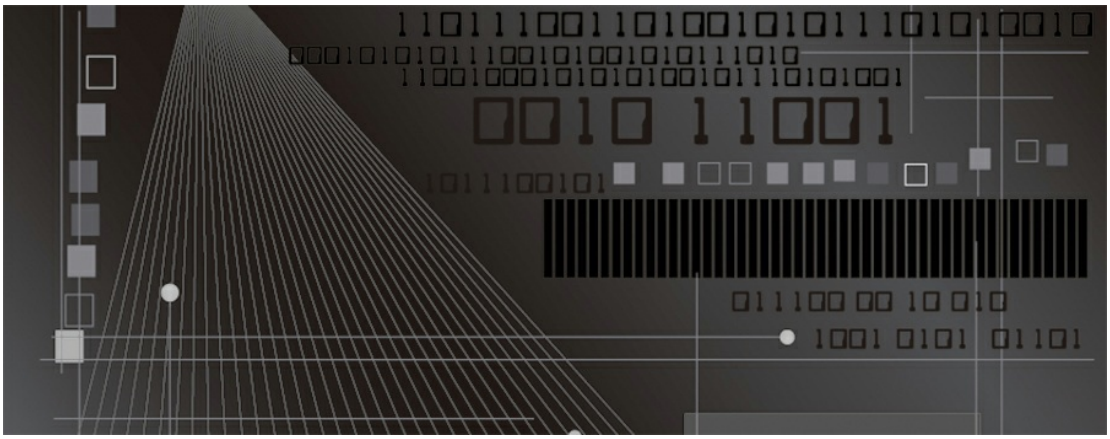

Instrucciones de comparación: CP/CPC/IPC

Las **comparaciones** son una instrucción fundamental para **organizar la secuencia de ejecución de un programa**. Estas instrucciones permitirán **alterar la ejecución secuencial de un programa**. Las condiciones son instrucciones primitivas para implementar instrucciones de alto nivel como bucles o condiciones (for, while-do, do-while, if, switch, etc.).

CP (ComPare)	Compara dos registros mediante resta. $0001 \mid 01r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su formato es el siguiente: CP Rd, Rr ;Rd = Rr • Donde Rd, Rr puede ser R0 .... R31.
CPC (ComPare with Carry)	Compara dos registros con Carry mediante resta. $0000 \mid 01r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su formato es: CPC Rd, Rr→ ;Rd = Rr – C • Compara Rd con Rd y carry. • Donde Rd, Rr puede ser R0 .... R31.
IPC (ComPare with Immed)	Compara registro con constante. $0011 \mid K_7k_6k_5k_4 \mid d_3d_2d_1d_0 \mid k_3k_2k_1k_0$ Su formato es: IPC Rd, k ;Rd = k

Para las tres instrucciones:

- Se modifican los flags: **H, S, V, N, Z, C** .
- Los contenidos de registros que interviene en la comparación **no se ven afectados**, solo modifica los flags del registro SREG.
- Es típico utilizar estas instrucciones justo **antes de instrucciones de salto condicional**, como, por ejemplo: BREQ o BRLT.



Instrucciones lógicas: AND/ANDI/EOR/OR/ORI/TST

Las **operaciones lógicas** más comunes son:

AND (Y lógico)	$0010 \mid 00r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su formato es el siguiente: AND Rd, Rr ;Rd ← Rd · Rr • Y lógico a nivel de bits. • Rd, Rr puede ser cualquiera de R0 .... R31.
ANDI (Y lógico con Inmediato)	$0111 \mid K_7k_6k_5k_4 \mid d_3d_2d_1d_0 \mid k_3k_2k_1k_0$ Su formato es: ANDI Rd, k ;Rd ← Rd · k • AND lógico a nivel de bits entre Rd y k. • Rd, Rr debe ser de R16 R31.
EOR (OR exclusivo)	$0010 \mid 01r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su formato es: EOR Rd, Rr ;Rd ← Rd ⊕ Rr • XOR bit a bit entre Rd y Rr. • Rr y Rd debe ser un registro entre R0 .... R31. Nota: también se utiliza para almacenar un 0 en el registro Rd (CLR Rd ≡ EOR Rd, Rd).

O (OR lógico)	$0010 \mid 10r_4d_4 \mid d_3d_2d_1d_0 \mid r_3r_2r_1r_0$ Su formato es: O Rd, Rr ;Rd ← Rd + Rr • OR bit a bit entre Rd y Rr. • Rd, Rr puede ser cualquiera registro entre R0 .... R31.
ORI (OR con Inmediata)	$0110 \mid K_7k_6k_5k_4 \mid d_3d_2d_1d_0 \mid k_3k_2k_1k_0$ Su formato es: ORI Rd, k ;Rd ← Rd + k1 • OR bit a bit entre Rd y k. • Rd debe ser un registro entre R16 R31. • K debe ser de 8 bits: 0 < k < 256.
TST (TeST por 0 o <0)	Su formato es el que se muestra a continuación: TST Rd ;Rd ← Rd · Rd • Calcula Rd y Rd (Rd mantiene su valor) y calcula si Rd es cero o menos de cero. • Rd debe ser uno de R0 .... R31. Nota: TST y AND son similares. TST Rd ≡ AND Rd, Rd.

Para todas las instrucciones: solo modifican los flags **Z, N, V**.

Ejemplo

Establecer en 1 bits 0, 2 y 5, dejar el resto no afectado

Ejemplo

Borrar los bits 0 a 3 y 7, deje el resto no afectado

Borrar los bits 0 a 3 y 7, deje el resto no afectado

```
1. ANDI Rd, 0B01110110
```

Ejemplo

Invertir los bits 2 y 5, dejan el resto no afectado

Invertir los bits 2 y 5, dejan el resto no afectado

```
1. MOV R16, Rd          ;Hacemos una copia del valor inicial
2. ANDI R16, 0B11011011  ;Nos quedamos con todos los bits excepto el 2 y 5 del
   valor original
3. MOV R17, Rd          ;Hacemos una copia del valor inicial
4. COM R17              ;Invertimos el valor original en R17
5. ANDI R17, 0B00100100  ;Borramos todos los valores del numero invertido,
   excepto lo bits 2 y 5.
6. OR R16, R17          ;Juntamos los dos números anteriores.
7. MOV Rd, R16          ;Volvemos a cargar el valor invertido en la posición original.
```


Otras instrucciones de importancia

En la siguiente tabla se muestran **otras instrucciones** importantes:

	Nemotécnico	Operandos	Descripción	Operación	Flags modificados
Aritméticas	COM	Rd	Hacer el complemento a 1 del registro	Rd←0xFF - Rd	Z,C,N,V
	NEG	Rd	Hacer el complemento a 2 del registro	Rd←0x00 - Rd	Z,C,N,V,H
	INC	Rd	Incrementar registro	Rd←Rd + 1	Z,N,V
	DEC	Rd	Decrementar registro	Rd←Rd - 1	Z,N,V
	SBR	Rd,K	Establecer Bit(s) en Registro a 1	Rp←Rp v k	Z,N,V
	CBR	Rd,K	Resetear Bit(s) en Registro a 0	Rp←Rp · (0xFF - K)	Z,N,V
Bit	SBI	P,b	Establecer Bit “b” en Registro “P”de E/S a 1	I/O(P,b)←1	None
	CBI	P,b	Resetear Bit “b” en	I/O(P,b)←0	None

Nomenclatura:

Rd: Registro entre R0 R31.

Rp: Registro entre R16 R31.

P: uno de los primeros 32 registros de Entrada/Salida (0 .... 31).

b: indica un numero de bit en un byte: b = 0 ... 7.

Ejemplos:

- SBR R20, 00111100b ;establece los bits 2,3,4 y 5 a 1 en el registro R20.
- CBR R20, 10000001b establece los bits 0 y 7 a 0 en el registro R20.
- SBI 0x05, 7 ;establece a 1 el bit 7 del PORTB (0x05 = PORTB).
- CBI 0x05, 7 ;establece a 0 el bit 7 del PORTB (0x05 = PORTB).

Instrucciones para manipular el registro SREG

Las siguientes **instrucciones** permiten establecer (Set) o desactivar (Clear) los flags del registro de estado SREG:

flag	Para establecer	Para resetear	Uso
C	SEC	CLC	Bit de Carry
Z	SEZ	CLZ	Bit de Cero
N	SEN	CLN	Bit de negativo
V	SEV	CLV	Desbordamiento en conversión Ca2
S	SES	CLS	Bit de signo
H	SEH	CLH	Bit de Carry intermedio: Usado en operaciones aritméticas
T	SET	CLT	Bit sin uso. Utilizado para almacenar un bit: Bit de memoria temporal
I	SEI	CLI	Bit de interrupciones activadas

Establecer bit o resetear bit en el registro de estado SREG manualmente

- BSET:
  - BSET B ;establece a 1 el bit “b” en SREG.

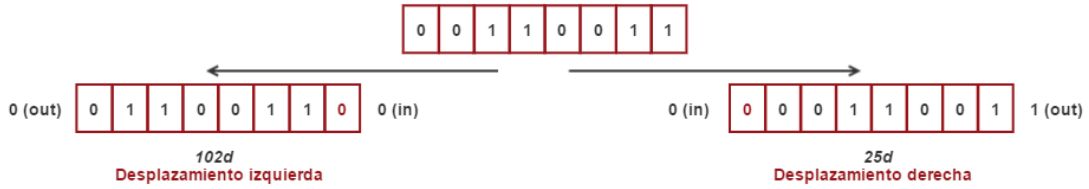
ninguna operación. Por el contrario, este registro se puede **leer y escribir como una memoria**. Además, hay instrucciones que permiten copiar el contenido desde el bit hacia otro bit del SREG y viceversa.

Así, las instrucciones siguientes permiten manejar este bit:

SET (Set T)	Almacena un 1 en el bit T.
CLT (Clear T)	Almacena un 0 en el bit T.
BST (Bit STore)	BST Rr, b ;Almacena el bit "b" del registro Rr en el bit T.
BLD (Bit LoaD)	BLD Rd, b ;Carga el bit "b" del registro Rd desde el bit T.

Desplazamiento y rotación de bits

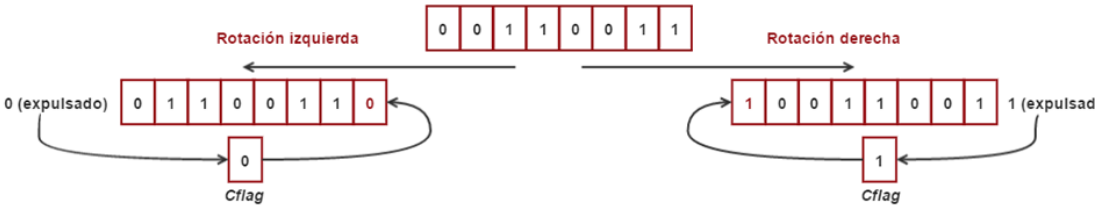
El desplazamiento de bits, consiste en el **movimiento de los bits en un registro** de una posición a la izquierda (o hacia la derecha):



Las **instrucciones** que realizan estas funciones son:

LSL (Logical Shift Left)	• Su formarto es LSL Rd. • Desplaza los bits del registro Rd a la izquierda. • Siempre entra un **0 como nuevo bit por la derecha**, y el bit 7 se pierde.
LSR (Logical Shift Right)	• Su formato es LSR Rd. • Desplaza los bits del registro Rd a la **derecha**. • Siempre entra un **0 como nuevo bit por la izquierda**, y el bit 0 se pierde.

La rotación consiste en un desplazamiento circular de los bits. La rotación, consiste en un desplazamiento, con la salvedad de que no se pierde ningún bit, el bit que sale por un extremo consecuencia del desplazamiento, es el mismo que entra por el otro extremo.



Las **instrucciones** que realizan estas funciones son:

ROL (ROtate Left)	• Su formato es ROL Rd. • Gira los bits del registro a la **izquierda**. El bit 7 que es expulsado por la izquierda, se inserta por la derecha.
ROR (ROtate Right)	• Su formato es ROR Rd. • Gira los bits del registro a la **derecha**. El bit 0 que es expulsado por la derecha, se inserta por la izquierda.

Nota: ambas instrucciones utilizan el **flag C con almacenamiento intermedio**, por lo que modifica dicho flag.

Instrucciones de multiplicación y división

Para realizar la multiplicación de dos números, hay **varias opciones**, dependiendo del tipo de dato que interviene.

La instrucción de multiplicación **más simple** e s **MUL (MULtiply),** la cual calcula el producto de dos números de 8 bits sin signo almacenados en los registros Rd y Rr.. Su formato es el siguiente:

MUL Rd, Rr ;R1:R0 ← Rd * Rr

Donde:

- El resultado se guarda en el par de registros R0:R1, para dar un registro de 16bits. R1 tiene el byte alto, R0 el byte bajo.
- Rd, Rr puede ser cualquiera de R0 .... R31.

Otras versiones de la multiplicación	• **MULS (MULtiply Signed)**: multiplica dos enteros con signo:
	  ○ Formato: MULS Rd, Rr ;R1:R0 ← Rd * Rr.
	• **MULSU (MULtiply Signed Unsigned)**: multiplica un entero con signo y otro sin signo:
	  ○ Formato: MULSU Rd, Rr ;R1:R0 ← Rd * Rr.
	  ○ Rr se interpreta sin signo y Rd con signo.

Multiplicación para números reales (punto flotante)	• **FMUL (Fractional MULtiply):** ○ Formato: FMUL Rd, Rr ;R1:R0 ← Rd * Rr << 1 ○ Multiplicación de dos números en punto flotante de simple precisión (8 btis) sin signo. Donde: ■ Rd, Rr puede ser cualquiera de R0 .... R31. ■ El resultado es un número fraccionario de 16 bits. ○ **FMULS (Fractional MULtiply Signed**): multiplica dos números en punto flotante con signo: ■ Formato: FMULS Rd, Rr ;R1:R0 ← Rd * Rr << 1 ○ **FMULSU (Fractional MULtiply Signed Unsigned**): multiplica un número en punto flotante con signo y otro sin signo: ■ Formato: FMULSU Rd, Rr ;R1:R0 ← Rd * Rr << 1 ■ Rr se interpreta sin signo y Rd con signo. Si deseas más información relativa al punto flotante, consulte las referencias bibliográficas que aparecen al final del contenido.

División

Los microprocesadores de Atmel* **no incluyen** la instrucción de división, pero **esta se puede implementar con desplazamientos**. Para más información ver el documento "División en

Resumen

En este recurso, hemos visto las **instrucciones de manipulación de datos** más importantes que implementa los microprocesadores Atmel. Pero estas instrucciones son una **simplificación** del conjunto de instrucciones que soporta el microprocesador.

Para tener una visión más exacta de estas instrucciones, es importante tener a mano la **documentación oficial del microprocesador**.

Si, necesitamos más información de cada una de las instrucciones soportadas por el microprocesador, la documentación anterior, es solo un resumen de **consulta rápida**.

En esta documentación oficial, además de tener la información de todas las instrucciones soportadas, también tendremos **información detallada** de cada una de ellas: registros que usa, flags que modifica, ciclos de reloj que consume en su ejecución, ejemplos de uso, etc.

Estos documentos son los elementos de consulta principal, cuando necesitemos **programar en ensamblador**.

Para más información, puedes consultar las **referencias bibliográficas** que aparecen al final del contenido.

¡Enhorabuena! Has finalizado con éxito.

Referencias bibliográficas

- IEEE coma flotante. Disponible en: <https://es.wikipedia.org/wiki/IEEE_coma_flotante>. [Consultado el 6 de junio de 2016].

- Instrucciones soportadas por el microprocesador Arduino ATmega328p. Disponible en: <http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf>. [Consultado el 6 de junio de 2016].

- Microprocesador Arduino ATmega328p. Disponible en: <http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Summary.pdf>. [Consultado el 6 de junio 2016].