
# 3. PROGRAMACIÓN

## Microcontroladores

M. C. Felipe Santiago Espinosa

Marzo de 2017

# Lenguaje ensamblador

**Un programa en Ensamblador puede incluir:**

- **Instrucciones:** Elementos del lenguaje que se traducen a código máquina. Cada instrucción tiene su opcode y sus operandos. El procesador ejecuta las instrucciones para determinar el comportamiento de un sistema.

- **Directivas:** Elementos del lenguaje que ayudan en la organización de un programa, indicando diferentes aspectos, como la ubicación del código, definiciones, etc. Las directicas no generan código máquina, son elementos propios de la herramienta que se emplee para ensamblar un programa.

# Conjunto de Instrucciones

- El repertorio del ATMega328 cuenta con 131 instrucciones, las cuales están organizadas en 5 grupos:
  - Instrucciones Aritméticas y lógicas (28)
  - Instrucciones de control de flujo (Bifurcaciones) (36)
  - Instrucciones de transferencia de datos (35)
  - Instrucciones para el manejo y evaluación de bits (28)
  - Instrucciones especiales (4)
- La mayoría de las instrucciones son de 16 bits.

# Instrucciones Aritméticas y Lógicas

- Sumas y restas

| Instrucción | | Descripción | Operación | Banderas |
|---|---|---|---|---|
| ADD | Rd, Rs | Suma sin acarreo | Rd = Rd + Rs | Z,C,N,V,H,S |
| ADC | Rd, Rs | Suma con acarreo | Rd = Rd + Rs + C | Z,C,N,V,H,S |
| ADIW | Rd, k | Suma constante a palabra | [Rd + 1:Rd] = [Rd + 1:Rd] + k | Z,C,N,V,S |
| SUB | Rd, Rs | Resta sin acarreo | Rd = Rd - Rs | Z,C,N,V,H,S |
| SUBI | Rd, k | Resta constante | Rd = Rd – k | Z,C,N,V,H,S |
| SBC | Rd, Rs | Resta con acarreo | Rd = Rd – Rs – C | Z,C,N,V,H,S |
| SBCI | Rd, k | Resta constante con acarreo | Rd = Rd – K – C | Z,C,N,V,H,S |
| SBIW | Rd, k | Resta constante a palabra | [Rd + 1:Rd] = [Rd + 1:Rd] - k | Z,C,N,V,S |

Las que trabajan con palabras se ejecutan en 2 ciclos de reloj, las demás únicamente en 1 ciclo.

- Multiplicaciones enteras y fraccionales.

| Instrucción | Descripción | Operación | Banderas |
|---|---|---|---|
| MUL Rd, Rs | Multiplicación sin signo | R1:R0 = Rd * Rs | Z,C |
| MULS Rd, Rs | Multiplicación con signo | R1:R0 = Rd * Rs | Z,C |
| MULSU Rd, Rs | Multiplicación de un número con signo y otro sin signo | R1:R0 = Rd * Rs | Z,C |
| FMUL Rd, Rs | Multiplicación fraccional sin signo | R1:R0 = ( Rd * Rs ) << 1 | Z,C |
| FMULS Rd, Rs | Multiplicación fraccional con signo | R1:R0 = ( Rd * Rs ) << 1 | Z,C |
| FMULSU Rd, Rs | Multiplicación fraccional de un número con signo y otro sin signo | R1:R0 = ( Rd * Rs ) << 1 | Z,C |

Todas se ejecutan en 2 ciclos de reloj.

- Instrucciones Lógicas Binarias

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| AND | Rd, Rs | Operación lógica AND | Rd = Rd AND Rs |
| ANDI | Rd, k | Operación lógica AND con una constante | Rd = Rd AND k |
| OR | Rd, Rs | Operación lógica OR | Rd = Rd OR Rs |
| ORI | Rd, k | Operación lógica OR con una constante | Rd = Rd OR k |
| EOR | Rd, Rs | Operación lógica OR Exclusiva | Rd = Rd XOR Rs |

Todas se ejecutan en 2 ciclos de reloj.

- Instrucciones para enmascarar información

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| SBR | Rd, k | Pone en alto los bits indicados en la constante | Rd = Rd OR k |
| CBR | Rd, k | Pone en bajo los bits indicados en la constante | Rd = Rd AND (0xFF – k) |

Se ejecutan en 1 ciclo de reloj y modifican a las banderas Z, C, N, V y S.

- Instrucciones Aritméticas o Lógicas Unarias

| Instrucción | | Descripción | Operación | Banderas |
|---|---|---|---|---|
| COM | Rd | Complemento a 1 | Rd = 0xFF – Rd | Z,C,N,V,S |
| NEG | Rd | Negado o complemento a 2 | Rd = 0x00 – Rd | Z,C,N,V,H,S |
| INC | Rd | Incrementa un registro | Rd = Rd + 1 | Z,N,V,S |
| DEC | Rd | Decrementa un registro | Rd = Rd – 1 | Z,N,V,S |
| TST | Rd | Evalúa un registro | Rd = Rd AND Rd | Z,C,N,V,S |
| CLR | Rd | Limpia un registro (pone en bajo) | Rd = 0x00 | Z,C,N,V,S |
| SER | Rd | Ajusta un registro (pone en alto) | Rd = 0xFF | Ninguna |

- Se ejecutan en 1 ciclo de reloj.

# Instrucciones de Control de Flujo

- Saltos Incondicionales

| Instrucción | Descripción | Operación |
|---|---|---|
| RJMP k | Salto relativo | PC = PC + k + 1 |
| IJMP | Salto indirecto | PC = Z |
| JMP k | Salto absoluto | PC = k |

Los saltos relativo e indirecto se ejecutan en 2 ciclos de reloj, el salto absoluto en 3.

- Instrucciones para el manejo de Rutinas

| Instrucción | Descripción | Operación |
|---|---|---|
| RCALL k | Llamada relativa a una rutina | PC = PC + k + 1, PILA ← PC + 1 |
| ICALL | Llamada indirecta a una rutina | PC = Z, PILA ← PC + 1 |
| CALL k | Llamada absoluta a una rutina | PC = k, PILA ← PC + 1 |
| RET | Retorno de una rutina | PC ← PILA |
| RETI | Retorno de rutina de interrupción | PC ← PILA, I = 1 |

Las llamadas relativa e indirecta a rutinas se ejecutan en 3 ciclos de reloj, la llamada absoluta y los retornos se ejecutan en 4.

- Instrucciones para la comparación de datos.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| CP | Rd, Rs | Compara dos registros | Rd – Rs |
| CPC | Rd, Rs | Compara registros con acarreo | Rd – Rs – C |
| CPI | Rd, k | Compara un registro con una constante | Rd – k |

Modifican las banderas Z, C, N, V, H y S.

- Brincos Condicionales.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| BRBS | s, k | Brinca si el bit s del registro estado está en alto | si(SREG(s) == 1) PC = PC + k + 1 |
| BRBC | s, k | Brinca si el bit s del registro estado está en bajo | si (SREG(s) == 0) PC = PC + k + 1 |
| BRIE | k | Brinca si las interrupciones están habilitadas | si (I == 1) PC = PC + k + 1 |
| BRID | k | Brinca si las interrupciones están deshabilitadas | si (I == 0) PC = PC + k + 1 |
| BRTS | k | Brinca si el bit T está en alto | si (T == 1) PC = PC + k + 1 |
| BRTC | k | Brinca si el bit T está en bajo | si (T == 0) PC = PC + k + 1 |
| BRHS | k | Brinca si la bandera H está en alto | si (H == 1) PC = PC + k + 1 |
| BRHC | k | Brinca si la bandera H está en bajo | si (H == 0) PC = PC + k + 1 |
| BRGE | k | Brinca si es mayor o igual que (con signo) | si (S == 0) PC = PC + k + 1 |
| BRLT | k | Brinca si es menor que (con signo) | si (S == 1) PC = PC + k + 1 |
| BRVS | k | Brinca si la bandera V está en alto | si (V == 1) PC = PC + k + 1 |
| BRVC | k | Brinca si la bandera V está en bajo | si (V == 0) PC = PC + k + 1 |
| BRMI | k | Brinca si es negativo | si (N == 1) PC = PC + k + 1 |
| BRPL | k | Brinca si no es negativo | si (N == 0) PC = PC + k + 1 |
| BREQ | k | Brinca si los datos son iguales | si (Z == 1) PC = PC + k + 1 |
| BRNE | k | Brinca si los datos no son iguales | si (Z == 0) PC = PC + k + 1 |
| BRSH | k | Brinca si es mayor o igual | si (C == 0) PC = PC + k + 1 |
| BRLO | k | Brinca si es menor | si (C == 1) PC = PC + k + 1 |
| BRCS | k | Brinca si la bandera C está en alto | si (C == 1) PC = PC + k + 1 |
| BRCC | k | Brinca si la bandera C está en bajo | si (C == 0) PC = PC + k + 1 |

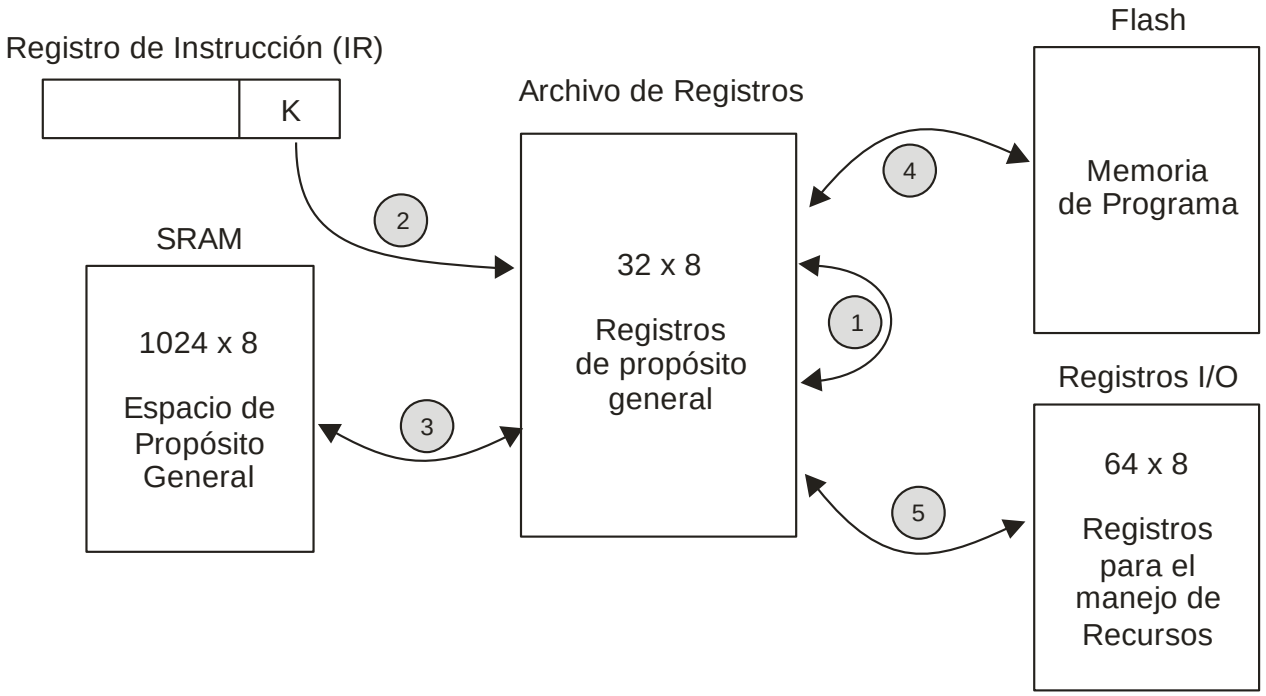
Pueden tardar 1 o 2 ciclos de reloj.

- “Saltitos” Condicionales.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| CPSE | Rd, Rs | Un saltito si los registros son iguales | si(Rd == Rs) PC = PC + 2 o 3 |
| SBRS | Rs, b | Un saltito si el bit b del registro Rs está en alto | si(Rs(b) == 1) PC = PC + 2 o 3 |
| SBRC | Rs, b | Un saltito si el bit b del registro Rs está en bajo | si(Rs(b) == 0) PC = PC + 2 o 3 |
| SBIS | P, b | Un saltito si el bit b del registro P está en alto, P es un registro I/O | si(I/O(P, b) == 1) PC = PC + 2 o 3 |
| SBIC | P, b | Un saltito si el bit b del registro P está en bajo, P es un registro I/O | si(I/O(P, b) == 0) PC = PC + 2 o 3 |

Pueden tardar 1, 2 o 3 ciclos de reloj.

# Instrucciones de Transferencia de Datos




1. Transferencias entre registros
2. Transferencia de una constante a registro
3. Transferencias entre memoria de datos y registros
4. Transferencias entre memoria de código y registros
5. Transferencias entre registros I/O y de propósito general

- Trasferencia entre registros.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| MOV | Rd, Rs | Copia un registro | Rd = Rs |
| MOVW | Rd, Rs | Copia un par de registros | Rd + 1: Rd = Rs + 1: Rs, Rd y Rs registros pares |

- Trasferencia de una constante a un registro.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| LDI | Rd, k | Copia la constante en el registro | Rd = k |

- Transferencias entre memoria de datos y registros (cargas)

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| LDS | Rd, k | Carga directa de memoria | Rd = Mem[k] |
| LD | Rd, X | Carga indirecta de memoria | Rd = Mem[X] |
| LD | Rd, X+ | Carga indirecta con post-incremento | Rd = Mem[X], X = X + 1 |
| LD | Rd, -X | Carga indirecta con pre-decremento | X = X - 1, Rd = Mem[X] |
| LD | Rd, Y | Carga indirecta de memoria | Rd = Mem[Y] |
| LD | Rd, Y+ | Carga indirecta con post-incremento | Rd = Mem[Y], Y = Y + 1 |
| LD | Rd, -Y | Carga indirecta con pre-decremento | Y = Y - 1, Rd = Mem[Y] |
| LDD | Rd, Y + q | Carga indirecta con desplazamiento | Rd = Mem[Y + q] |
| LD | Rd, Z | Carga indirecta de memoria | Rd = Mem[Z] |
| LD | Rd, Z+ | Carga indirecta con post-incremento | Rd = Mem[Z], Z = Z + 1 |
| LD | Rd, -Z | Carga indirecta con pre-decremento | Z = Z - 1, Rd = Mem[Z] |
| LDD | Rd, Z + q | Carga indirecta con desplazamiento | Rd = Mem[Z + q] |

- Transferencias entre registros y memoria de datos (Almacenamientos)

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| STS | k, Rs | Almacenamiento directo en memoria | Mem[k] = Rs |
| ST | X, Rs | Almacenamiento indirecto en memoria | Mem[X] = Rs |
| ST | X+, Rs | Almacenamiento indirecto con post-incremento | Mem[X] = Rs, X = X + 1 |
| ST | -X, Rs | Almacenamiento indirecto con pre-decremento | X = X - 1, Mem[X] = Rs |
| ST | Y, Rs | Almacenamiento indirecto en memoria | Mem[Y] = Rs |
| ST | Y+, Rs | Almacenamiento indirecto con post-incremento | Mem[Y] = Rs, Y = Y + 1 |
| ST | -Y, Rs | Almacenamiento indirecto con pre-decremento | Y = Y - 1, Mem[Y] = Rs |
| STD | Y + q, Rs | Almacenamiento indirecto con desplazamiento | Mem[Y + q] = Rs |
| ST | Z, Rs | Almacenamiento indirecto en memoria | Mem[Z] = Rs |
| ST | Z+, Rs | Almacenamiento indirecto con post-incremento | Mem[Z] = Rs, Z = Z + 1 |
| ST | -Z, Rs | Almacenamiento indirecto con pre-decremento | Z = Z - 1, Mem[Z] = Rs |
| STD | Z + q, Rs | Almacenamiento indirecto con desplazamiento | Mem[Z + q] = Rs |

- Transferencias entre memoria de datos y registros (Acceso a la Pila)

| Instrucción | Descripción | Operación |
|---|---|---|
| PUSH Rs | Inserta a Rs en la pila | Mem[SP] = Rs, SP = SP – 1 |
| POP Rd | Extrae de la pila y coloca en Rd | SP = SP + 1, Rd = Mem[SP] |

- Transferencias entre memoria de código y registros

| Instrucción | Descripción | Operación |
|---|---|---|
| LPM | Carga indirecta de memoria de programa en R0 | R0 = Flash[Z] |
| LPM Rd, Z | Carga indirecta de memoria de programa en Rd | Rd = Flash[Z] |
| LPM Rd, Z+ | Carga indirecta con post-incremento | Rd = Flash[Z], Z = Z + 1 |
| SPM | Almacenamiento indirecto en memoria de programa | Flash[Z] = R1:R0 |

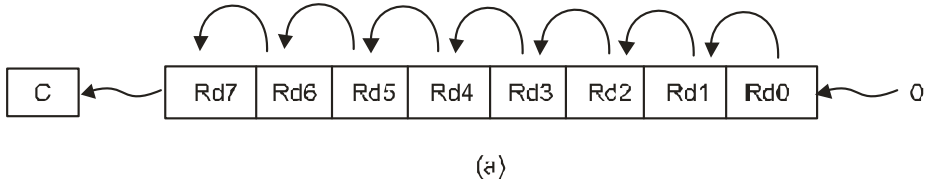
- Transferencias entre Registros I/O y registros de propósito general

| Instrucción | Descripción | Operación |
|---|---|---|
| IN Rd, P | Lee de un registro I/O, deja el resultado en Rd | Rd = P |
| OUT P, Rs | Escribe en un registro I/O, el valor de Rs | P = Rs |

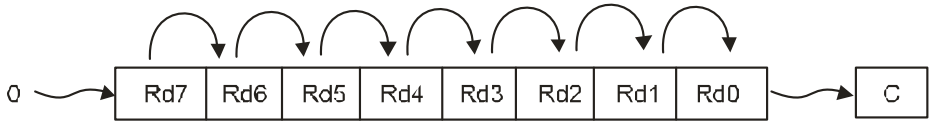
# Instrucciones para el manejo y evaluación de bits

- Desplazamientos y rotaciones

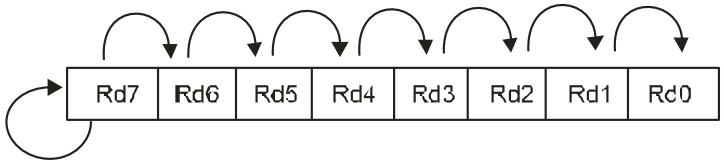
| Instrucción | | Descripción | Operación |
|---|---|---|---|
| LSL | Rd | Desplazamiento lógico a la izquierda | C = Rd(7), Rd(n+1) = Rd(n), Rd(0) = 0 |
| LSR | Rd | Desplazamiento lógico a la derecha | C = Rd(0), Rd(n) = Rd(n + 1), Rd(7) = 0 |
| ROL | Rd | Rotación a la izquierda | C = Rd(7), Rd(n + 1) = Rd(n), Rd(0) = C |
| ROR | Rd | Rotación a la derecha | C = Rd(0), Rd(n) = Rd(n + 1), Rd(7) = C |
| ASR | Rd | Desplazamiento aritmético a la derecha | Rd(n) = Rd(n + 1), n = 0 .. 6 |



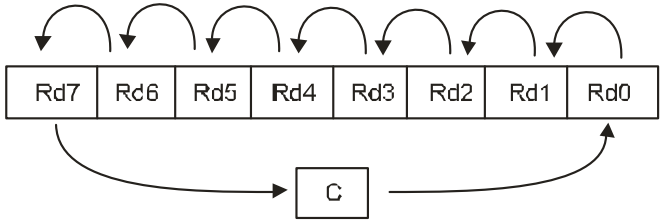

(a)



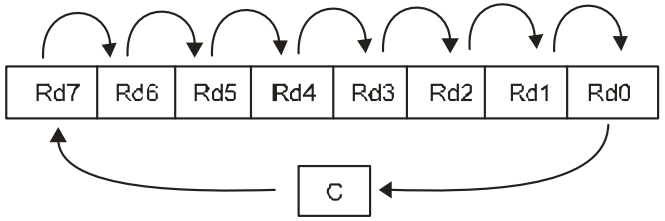

(b)




(c)




(c)




(d)

- Instrucción para el intercambio de nibbles.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| SWAP | Rd | Intercambia nibbles en Rd | Rd(7..4) = Rd(3..0), Rd(3..0) = Rd(7..4) |

Se ejecuta en 1 ciclo de reloj y no modifica banderas.

- Instrucciones para modificar bits en los Registros I/O.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| SBI | P, b | Pone en alto al bit b del Registro P | P(b) = 1 |
| CBI | P, b | Pone en bajo al bit b del Registro P | P(b) = 0 |

Registros I/O que están en el rango de 0x00 a 0x1F. Se ejecutan en 2 ciclos de reloj y tampoco modifican banderas.

- Instrucciones para transferencia de bits.

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| BTS | Rs, b | Almacena al bit b de un registro en T | T = Rs(b) |
| BTL | Rd, b | Carga en el bit b de un registro desde T | Rd(b) = T |

Se ejecutan en 1 ciclo de reloj.

- Instrucción para manipular los bits del registro Estado

| Instrucción | | Descripción | Operación |
|---|---|---|---|
| BSET | s | Pone en alto al bit s del registro Estado | SREG(s) = 1 |
| BCLR | s | Pone en bajo al bit s del registro Estado | SREG(s) = 0 |
| SEI | | Pone en alto al habilitador de interrupciones | I = 1 |
| CLI | | Pone en bajo al habilitador de interrupciones | I = 0 |
| SET | | Pone en alto al bit de transferencias | T = 1 |
| CLT | | Pone en bajo al bit de transferencias | T = 0 |
| SEH | | Pone en alto a la bandera de acarreo del nibble bajo | H = 1 |
| CLH | | Pone en bajo a la bandera de acarreo del nibble bajo | H = 0 |
| SES | | Pone en alto a la bandera de signo | S = 1 |
| CLS | | Pone en bajo a la bandera de signo | S = 0 |
| SEV | | Pone en alto a la bandera de sobreflujo | V = 1 |
| CLV | | Pone en bajo a la bandera de sobreflujo | V = 0 |
| SEN | | Pone en alto a la bandera de negativo | N = 1 |
| CLN | | Pone en bajo a la bandera de negativo | N = 0 |
| SEZ | | Pone en alto a la bandera de cero | Z = 1 |
| CLZ | | Pone en bajo a la bandera de cero | Z = 0 |
| SEC | | Pone en alto a la bandera de acarreo | C = 1 |
| CLC | | Pone en bajo a la bandera de acarreo | C = 0 |

# Instrucciones Especiales

| Instrucción | Descripción |
|---|---|
| NOP | No operación, empleada para forzar una espera de 1 ciclo de reloj |
| SLEEP | Introduce al MCU al modo de bajo consumo previamente seleccionado |
| WDR | Reinicia al Watchdog Timer, para evitar reinicios por su desbordamiento |
| BREAK | Utilizada para depuración desde la interfaz DebugWire |

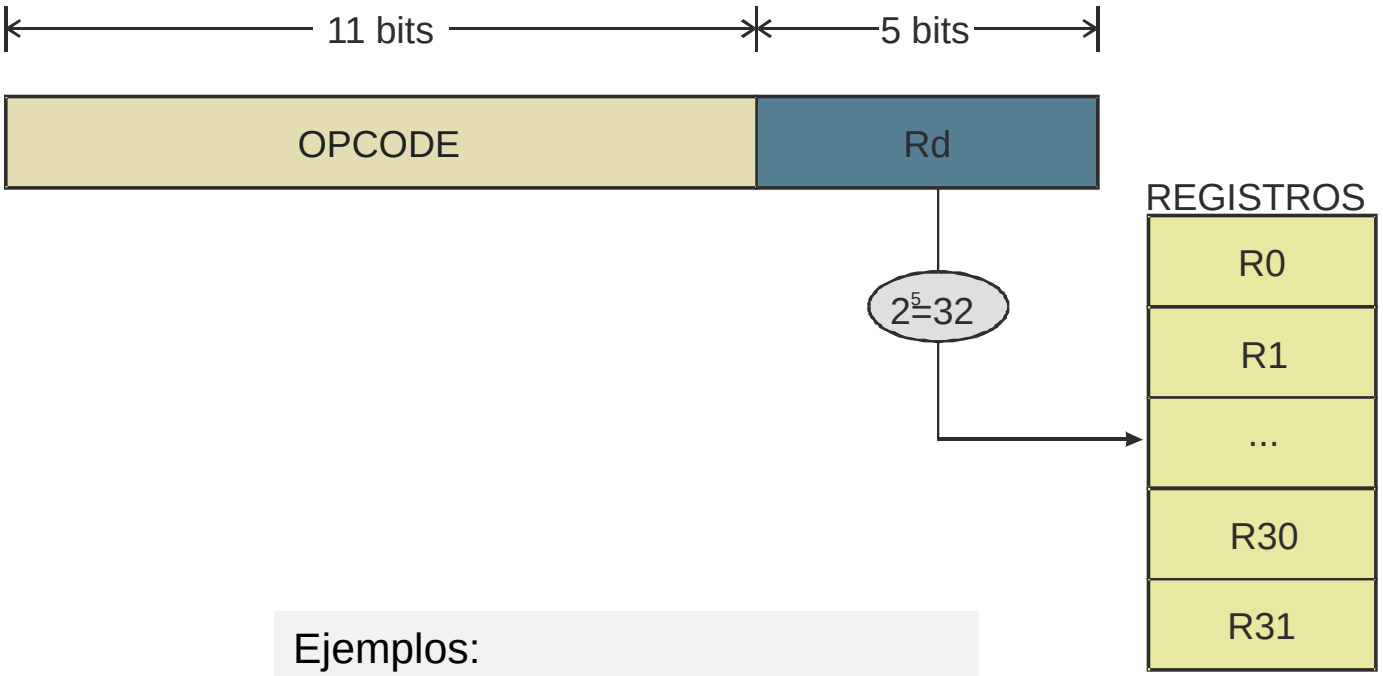
# Modos de direccionamiento

Los Modos de Direccionamiento indican la ubicación de los datos sobre los que operan las instrucciones. Los Microcontroladores AVR manejan los siguientes modos de direccionamiento:

1. Directo por registro
2. Directo a registros I/O
3. Directo a memoria de datos
4. Indirecto a memoria de datos
5. Indirecto a memoria de código
6. Inmediato
7. Direccionamientos en bifurcaciones

# Direccionamiento directo por registro

## Un registro:

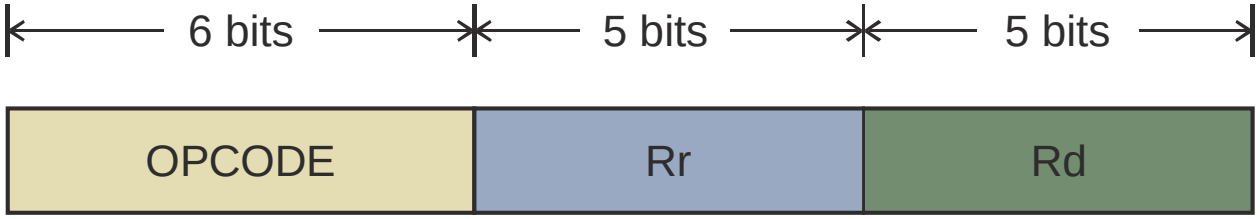



Ejemplos:

```
COM   R3
INC   R5
SER   R7
```

# Direccionamiento directo por registro

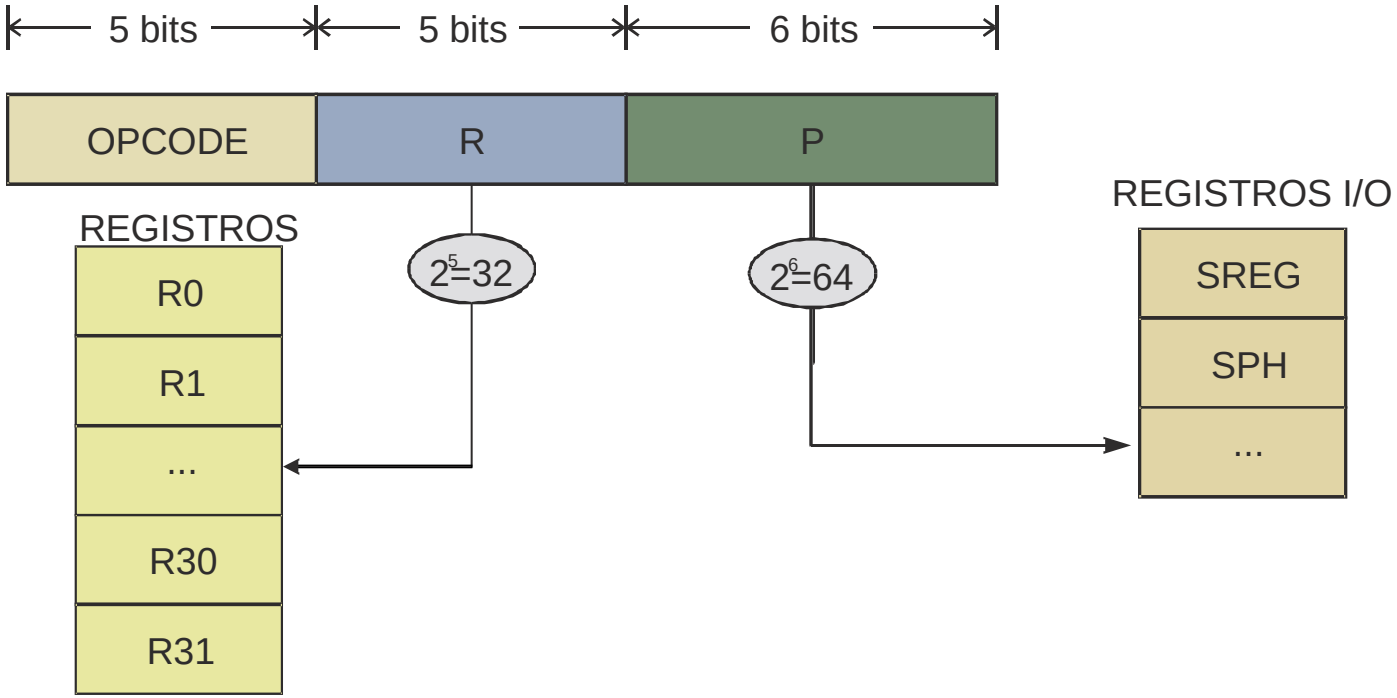
## Dos registros:




Ejemplos:

```
ADD   R1, R2
SUB   R1, R2
AND   R1, R2
MOV   R1, R2
```

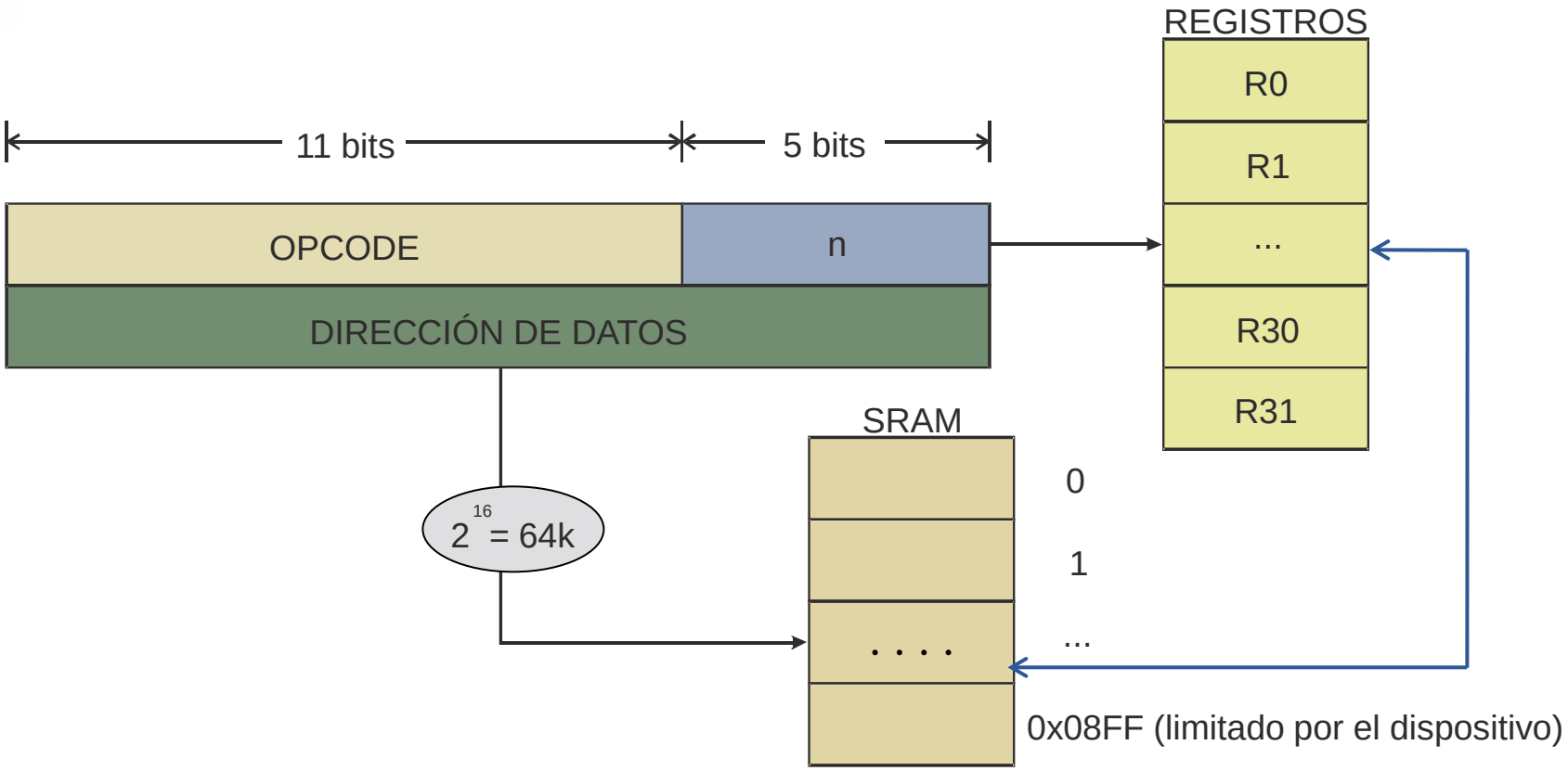
# Direccionamiento directo a I/O




Ejemplos:

```
OUT   PORTB, R13
IN    R15, PINA
```

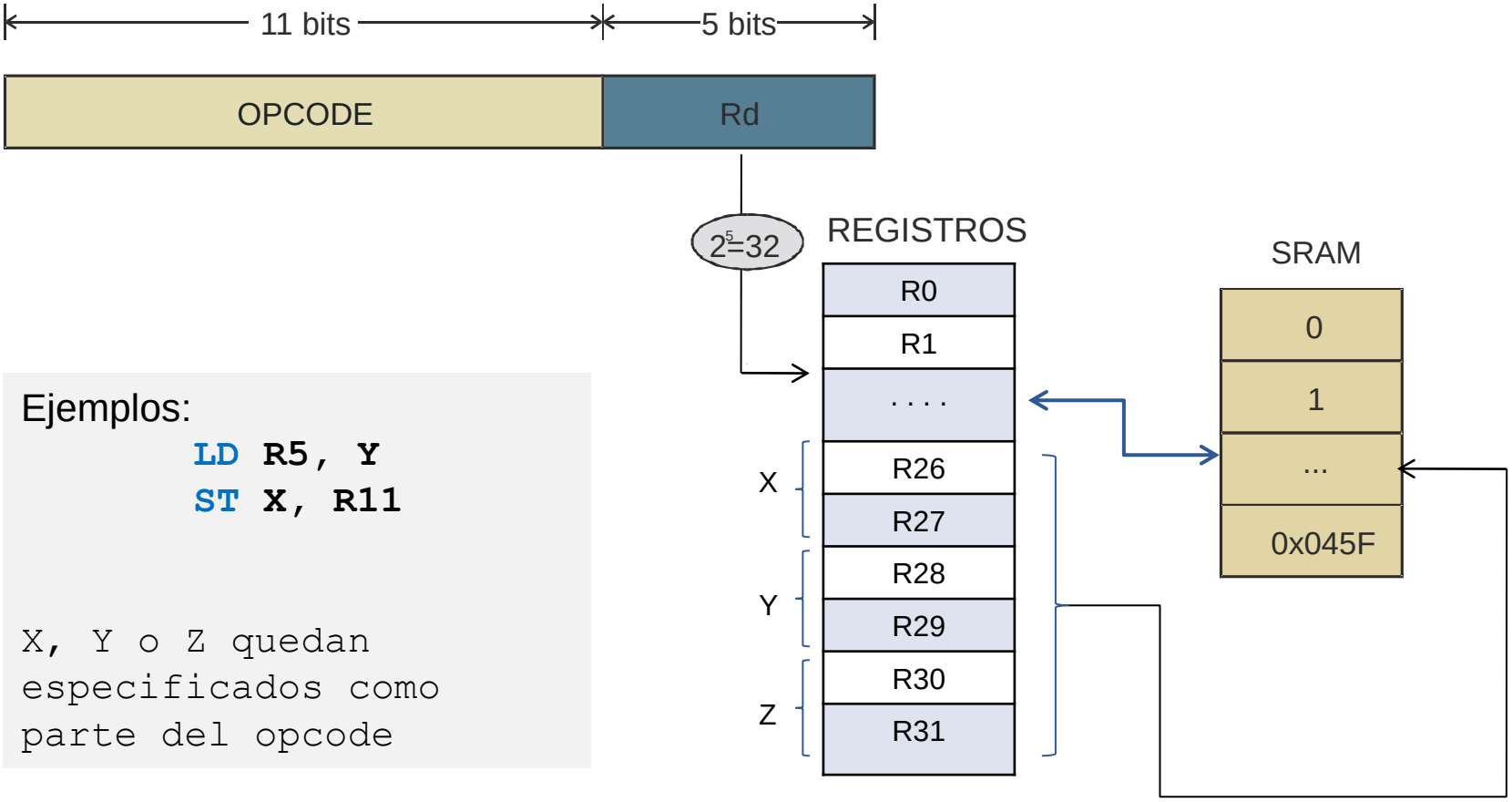
# Direccionamiento directo a Memoria de Datos




Ejemplos:

```
STS 0x0100, R5
LDS R16, 0x0110
```

# Direccionamiento indirecto a Memoria de Datos



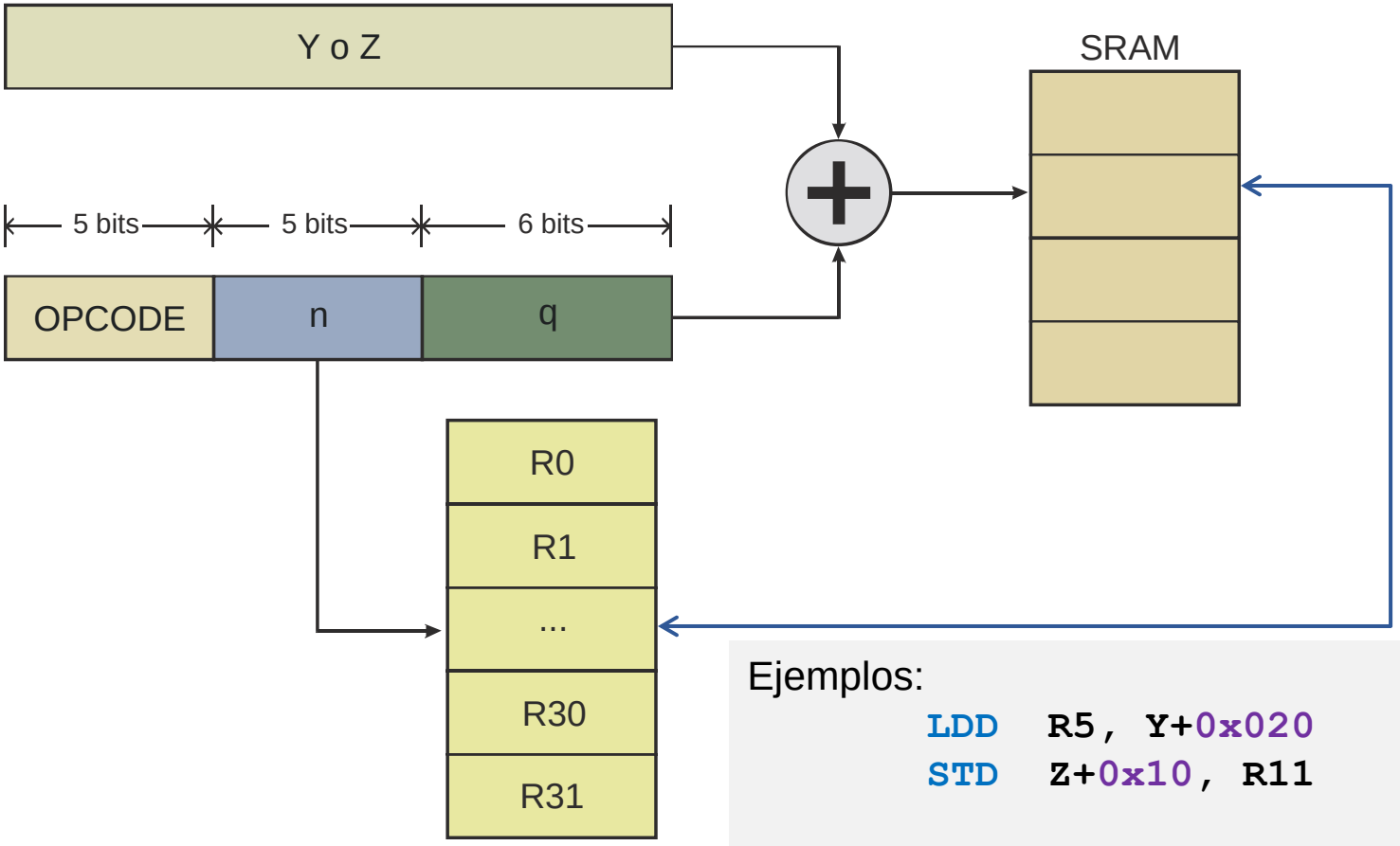

Ejemplos:

```
LD R5, Y
ST X, R11
```

X, Y o Z quedan especificados como parte del opcode

# Direccionamiento indirecto a Memoria de Datos

## Con desplazamiento



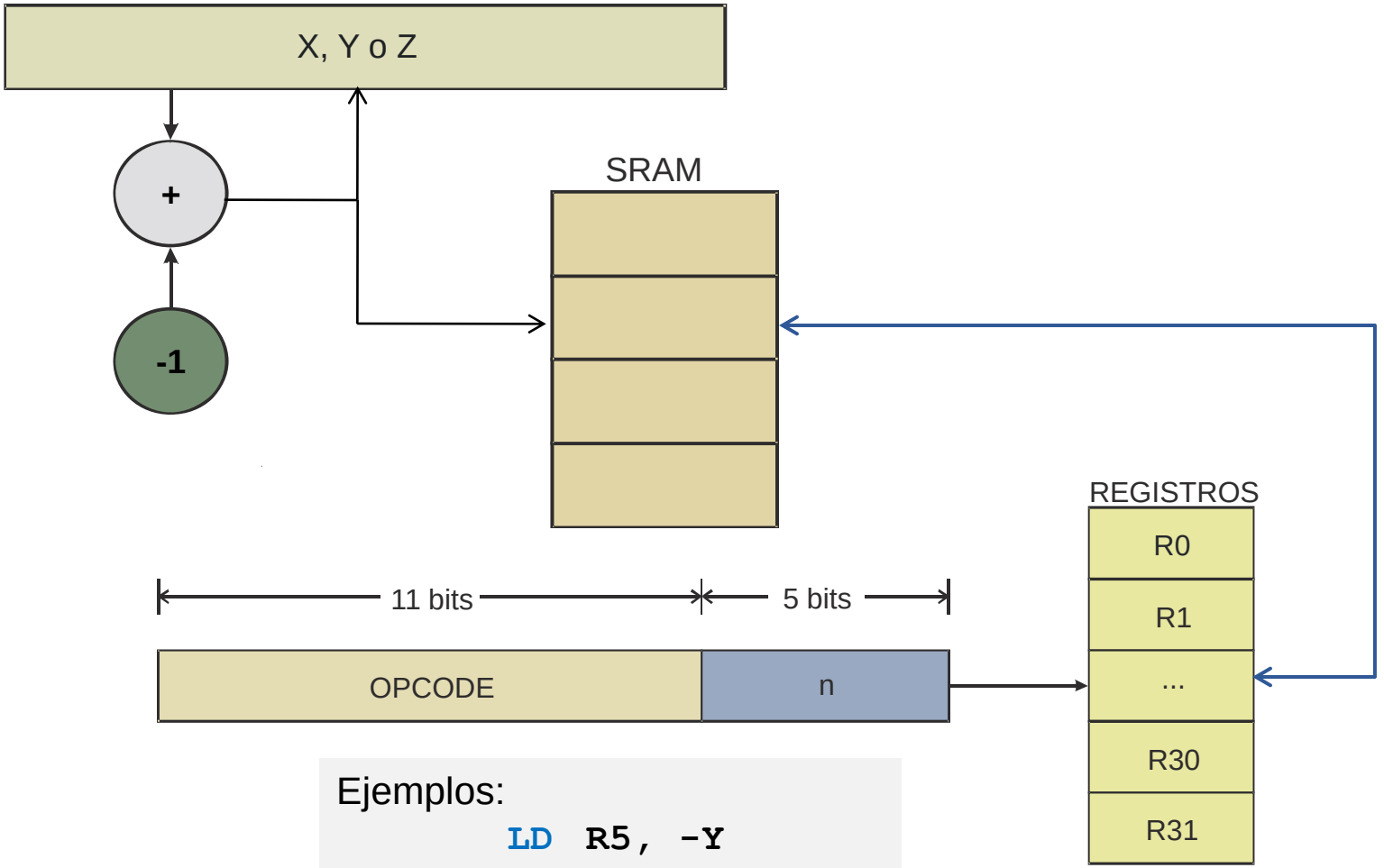

Ejemplos:

```
LDD   R5, Y+0x020
STD   Z+0x10, R11
```

Y o Z quedan especificados como parte del opcode

# Direccionamiento indirecto a Memoria de Datos

## Con pre-decremento

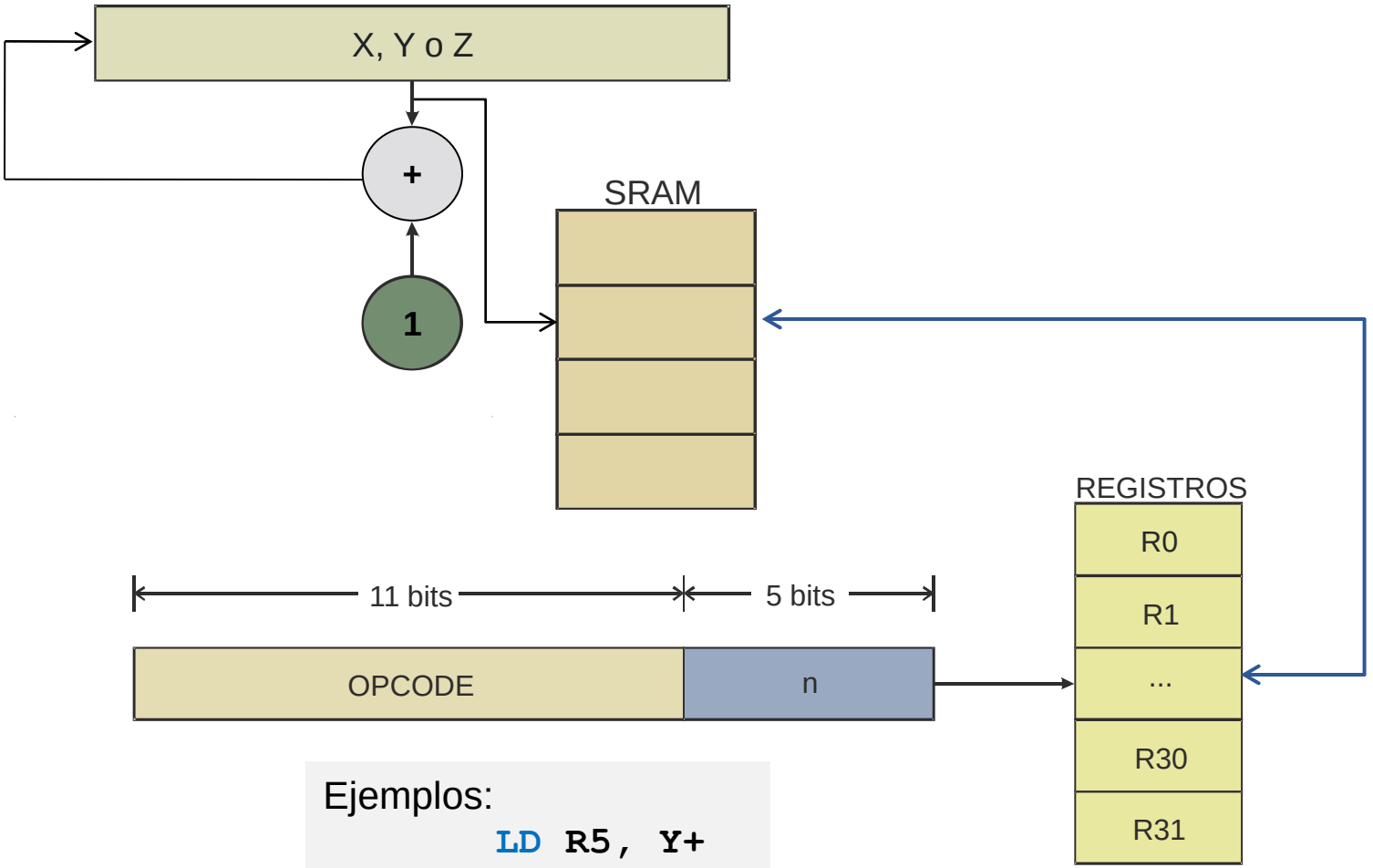



Ejemplos:

```
LD  R5, -Y
ST  -Z, R11
```

# Direccionamiento indirecto a Memoria de Datos

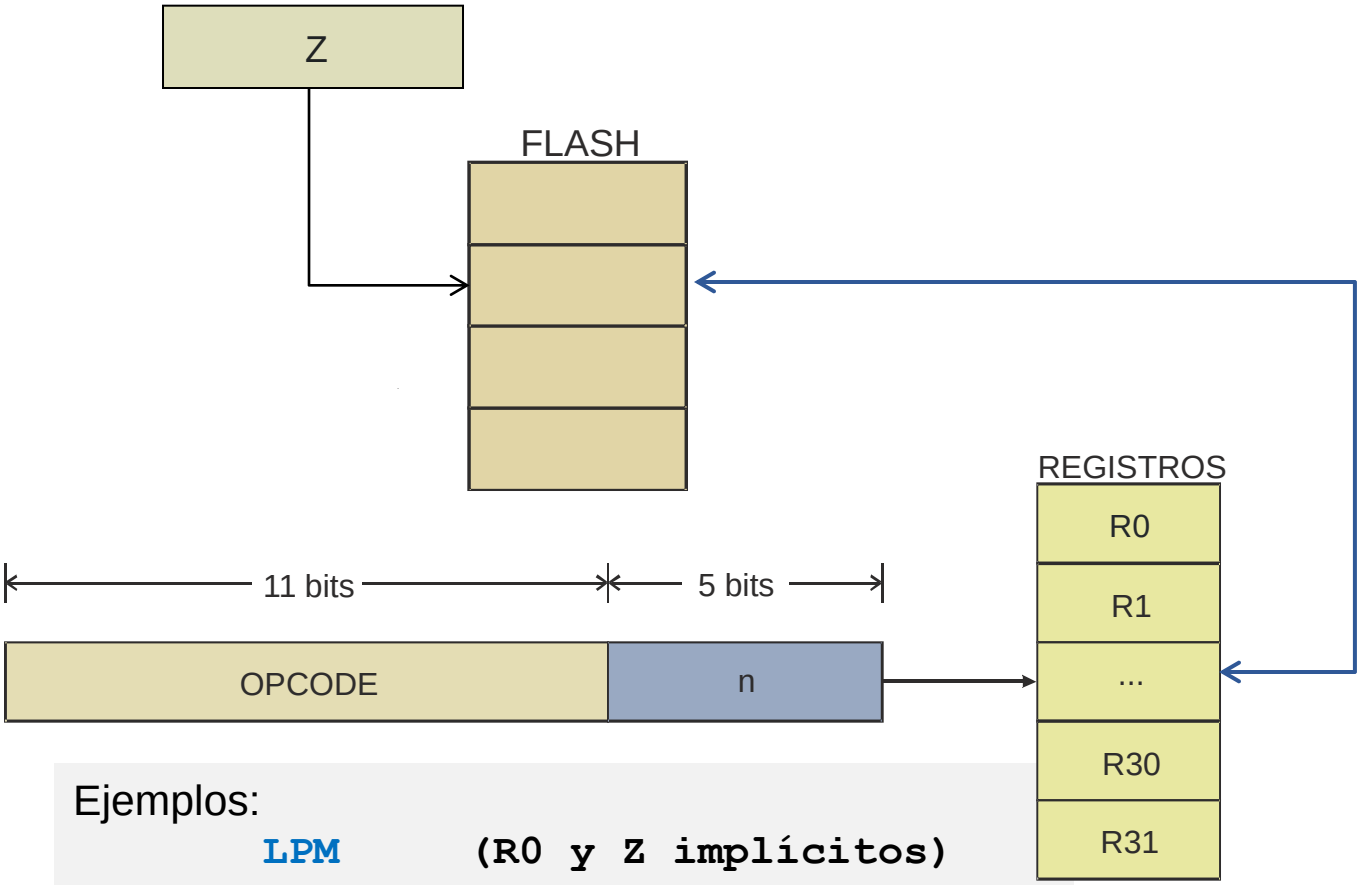
## Con post-incremento




Ejemplos:

```
LD R5, Y+
ST X+, R6
```

# Direccionamiento indirecto a Memoria de Código

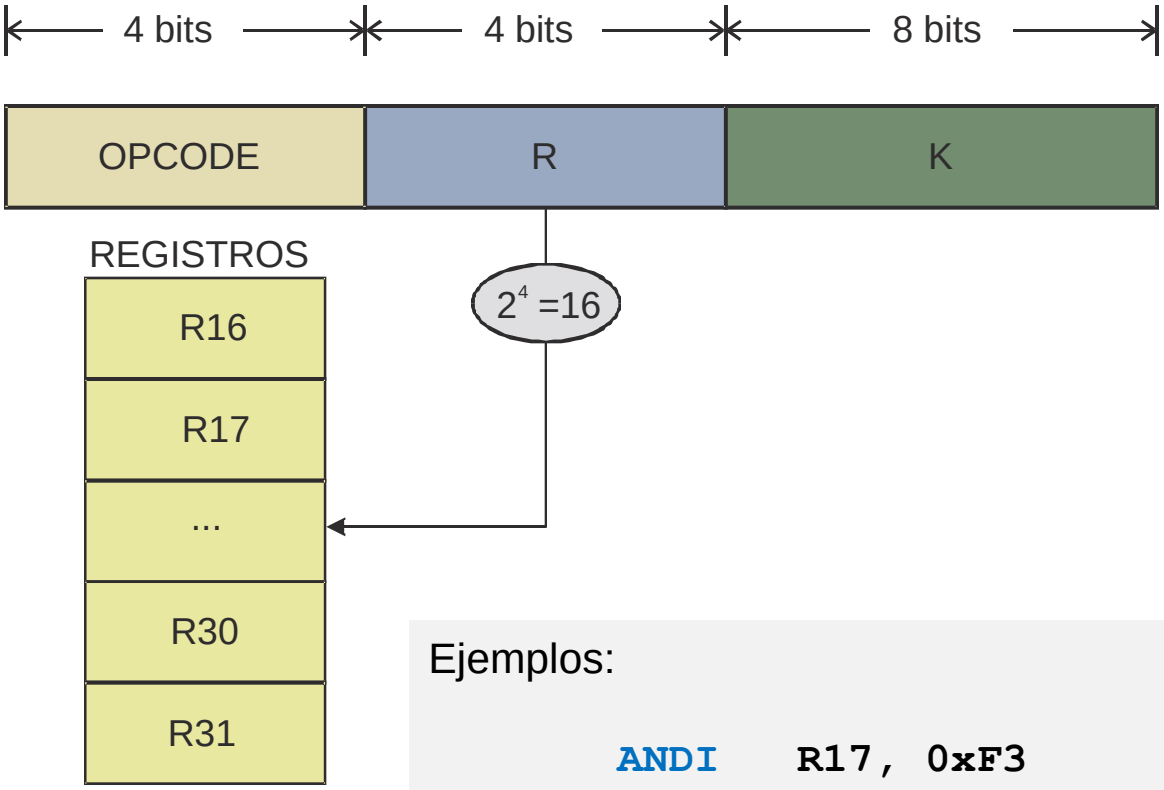



Ejemplos:

```
LPM      (R0 y Z implícitos)
LPM   R3, Z
SPM      (R1:R0 y Z implícitos)
```

**Nota:** También existe una carga indirecta de memoria de código con post-incremento del apuntador Z.

# Direccionamiento inmediato




Ejemplos:

```
ANDI    R17, 0xF3
SUBI    R19, 0x12
ORI     R31, 0x03
```

# Direccionamientos en Bifurcaciones

Las bifurcaciones o saltos (pueden ser condicionales o incondicionales) permiten cambiar el flujo secuencial, durante la ejecución de un programa. Esto se consigue modificando el valor del registro contador del programa (*Program Counter*).

Cuando no hay una bifurcación, el PC automáticamente debe incrementarse en 1.

## Indirectas

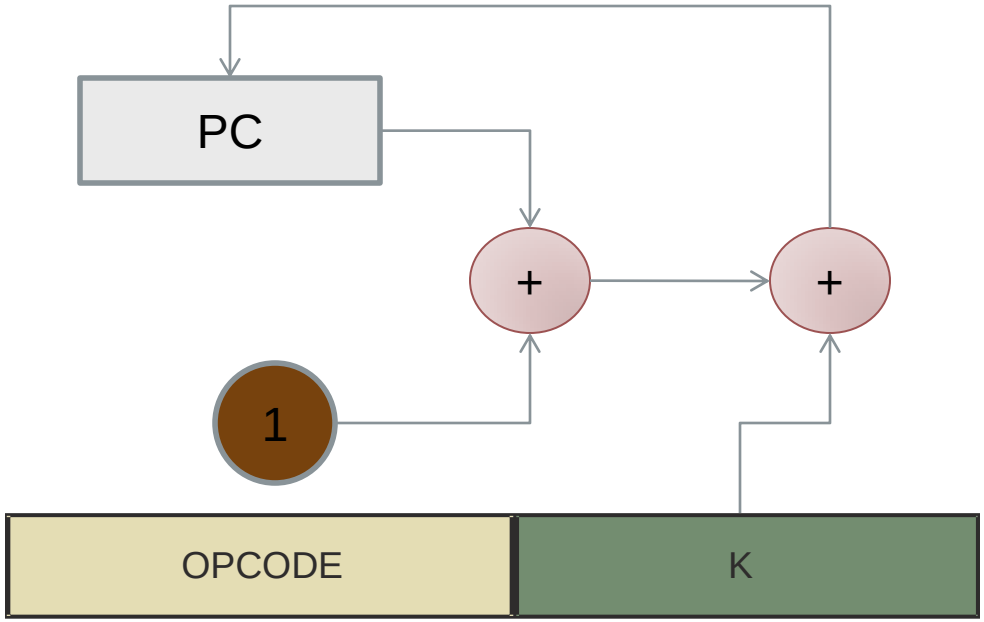

Ejemplos:

```
IJMP
ICALL
```

**Nota:** El apuntador Z queda implícito en la instrucción.

# Direccionamientos en Bifurcaciones

## Relativas




Ejemplos:

```
RJMP    -20
RCALL   32

BREQ    15
BRNE    -10
BRGE    10
```

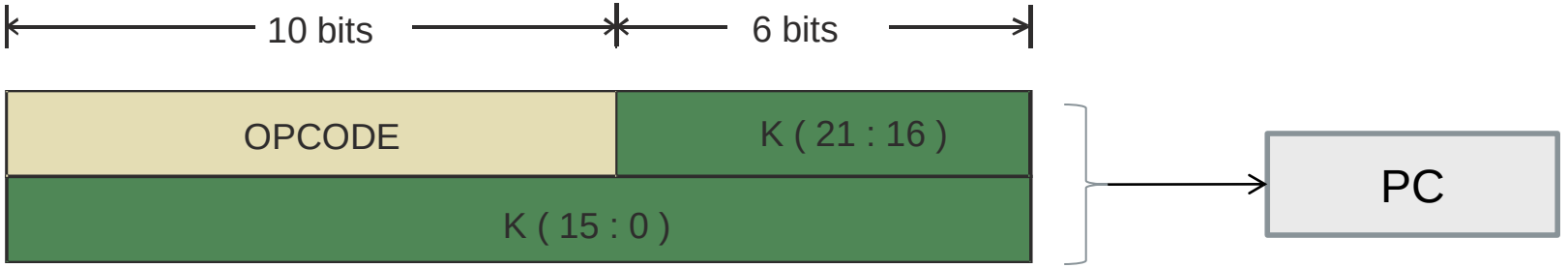
En bifurcaciones incondicionales, se dispone de 12 bits para K.

En bifurcaciones condicionales, Solo se dispone de 7 bits.

**Notas:** 1. En un programa se utilizan etiquetas, el ensamblador calcula el valor de las constantes.
2. Existen bifurcaciones condicionales que sólo brincan la siguiente instrucción, también son relativas al PC.

# Direccionamientos en Bifurcaciones

## Absolutas




Ejemplos:

```
JMP     0x300000
CALL    0x1FFFFF
```

# Ejercicio:

- Ignorando como se llevó un conjunto de 20 enteros sin signo a la memoria SRAM, ubicados a partir de la dirección 0x0200 y ocupando un byte por cada entero, realice una secuencia de instrucciones en ensamblador que use **direccionamiento indirecto** e incluya **comparaciones** para que, al final de la secuencia en el registro R20 quede el mayor de los 20 enteros.

  - *Sugerencia:* Revise cómo se resolvería el problema en alto nivel.

# Directivas comunes:

- **INCLUDE:** Se utiliza para leer el código fuente de otro archivo

En un ATMega328P se debe agregar:

**.include** <m328pdef.inc>

Directivas para definir diferentes tipos de segmentos en un programa:

- **CSEG:** Segmento de Código (por omisión). Se crea un *archivo.hex* para la memoria FLASH.
- **DSEG:** Segmento de Datos.  Para reservar espacio en la SRAM.
- **ESEG:** Segmento de EEPROM. Se crea un *archivo.eep*, a descargarse en la memoria EEPROM.

```
        .DSEG                   ; Inicia un segmento de datos
var1:   .BYTE   1               ; Variable de 1 byte en SRAM

        .CSEG                   ; Inicia un segmento de código
        ldi     R16, 0x25       ; Instrucciones para alguna tarea
        mov     r0,r16
        . . .

const:  .DB 0x02                ; Constante con 0x02 en flash

        .ESEG                   ; Inicia un segmento de EEPROM
eevar1:.DB      0x3f            ; Constante con 0x3f en EEPROM
```

- **DB y DW** (Define Byte y Define Word) Directivas para definir constantes o variables, en memoria de programa o en EEPROM.

```
   .CSEG
; Constantes en memoria FLASH
const1:   .DB       0x33
consts1:            .DB       0, 255, 0b01010101, -128, 0xaa
consts2:  .DW       0, 0xffff, 0b1001110001010101, -32768, 65535

.ESEG
; Constantes en EEPROM
eevar1:   .DB       0x37
eelist1:  .DB       1,2,3,4
eelist2:  .DW       0,0xffff,10
```

- **EQU:** Define una constante simbólica, que puede usarse dentro de las instrucciones.

- Ejemplo:

```
.EQU io_offset = 0x23
.EQU portA     = io_offset + 2

.CSEG                          ; Inicia el segmento de código

        clr     r2             ; Limpia el registro 2
        out     portA, r2      ; Escribe a portA
```

- **ORG:** Organizar a cualquiera de los segmentos de memoria, indica en qué dirección se ubicará la información subsecuente, pudiendo ser variables en SRAM, constantes en EEPROM o en FLASH, o código en FLASH.

```
        .DSEG
        .ORG    0x120
var1:   .Byte   1               ; Variable en la dirección 0x120 del
                                ; segmento de datos
  .CSEG
  .ORG 0x000            ; Código ubicado en la dirección 0x00
  RJMP inicio

  .ORG 0x010
inicio:                 ; Código ubicado en la dirección 0x10
  MOV  R1, R2
  . . . .
```

- **HIGH y LOW :** Se utilizan para separar constantes de 16 bits.
- **HIGH:** Byte alto.
- **LOW:** Byte bajo.

```
    LDI   R20, LOW (1022)
    LDI   R21, HIGH (1022)

   LDI    R30, LOW(tabla)
   LDI    R31, HIGH(tabla)
   . . . .
tabla: .DB        0x01,0x02,0x03,0x04
```

- **BYTE:** Reserva uno o más bytes en SRAM o EEPROM para el manejo de variables, la cantidad de bytes debe especificarse. Este espacio no es inicializado y posteriormente será referido con una etiqueta.

```
        .DSEG
var1:   .BYTE   1               ; variable de 1 byte
var2:   .BYTE   10              ; variable de 10 bytes

        .CSEG
        STS     var1, R17       ; acceso a var1 por dir.  Directo

        LDI     R30, LOW (var2)     ; Z apunta a Var2
        LDI     R31, HIGH (var2)
        LD      R1, Z               ; acceso indirecto a var2
```

# PROGRAMACIÓN EN ALTO NIVEL

Lenguaje C

# Tipos de Datos

| Tipo | Tamaño (bits) | Rango |
|---|---|---|
| char | 8 | -128 a 127 |
| unsigned char | 8 | 0 a 255 |
| signed char | 8 | -128 a 127 |
| int | 16 | -32, 768 a 32, 767 |
| short int | 16 | -32, 768 a 32, 767 |
| unsigned int | 16 | 0 a 65, 535 |
| signed int | 16 | -32, 768 a 32, 767 |
| long int | 32 | -2, 147, 483, 648 a 2, 147, 483, 647 |
| unsigned long int | 32 | 0 a 4, 294, 967, 295 |
| signed long int | 32 | -2, 147, 483, 648 a 2, 147, 483, 647 |
| float | 32 | $+/-\ 1.175 \times 10^{-38}$ a $+/- 3.402 \times 10^{+38}$ |
| double | 32 | $+/- 1.175 \times 10^{-38}$ a $+/- 3.402 \times 10^{+38}$ |

# Definiciones en WINAVR

| | |
|---|---|
| `typedef signed char` | **`int8_t`** |
| `typedef unsigned char` | **`uint8_t`** |
| `typedef signed int` | **`int16_t`** |
| `typedef unsigned int` | **`uint16_t`** |
| `typedef signed long int` | **`int32_t`** |
| `typedef unsigned long int` | **`uint32_t`** |

# Operadores

| Operadores Aritméticos de C | | |
|---|---|---|
| operador | descripción | ejemplo |
| * | Multiplicación | a * b |
| / | División | a / b |
| % | Módulo | a % b |
| + | Suma | a + b |
| - | Resta | a - b |

**ejemplo**

z *= 12;  equivalente a  z = z * 12;
z /= 2;  equivalente a  z = z / 12

**Operador ternario**
Expresión1 **?** Expresión2 **:** Expresión3**;**

| Operadores Asignación de C | | |
|---|---|---|
| operador | descripción / ejemplo | |
| = | asignación simple | a = b |
| *= | a *= b | a = a * b |
| /= | a /= b | a = a / b |
| %= | a %= b | a = a % b |
| += | a += b | a = a + b |
| -= | a -= b | a = a - b |
| &= | a &= b | a = a & b |
| \|= | a \|= b | a = a \| b |
| <<= | a <<= b | a = a << b |
| >>= | a >>= b | a = a >> b |
| ^= | a ^= b | a = a ^ b |

Asignación después de una manipulación de bits.

## Manipulación de bits

| | |
|---|---|
| Complemento Uno | ~ |
| Desplazamiento a la Izquierda | << |
| Desplazamiento a la Derecha | >> |
| AND | & |
| OR | \| |
| OR Exclusiva | ^ |

Trabajan sobre operandos que no son punto flotante (char, int, long) y afectan el resultado al nivel de bits.

| operador | descripción | ejemplo |
|---|---|---|
| ++ | incremento | a++, ++a |
| -- | decremento | a--, --a |

| Operadores Relacionales y de igualdad | | |
|---|---|---|
| operador | descripción | ejemplo |
| < | menor que | a < b |
| > | mayor que | a > b |
| <= | menor o igual que | a <= b |
| >= | mayor o igual que | a >= b |
| == | igual | a == b |
| != | no igual | a != b |

## Operadores Lógicos

| | |
|---|---|
| AND | && |
| OR | \|\| |
| NOT | ! |

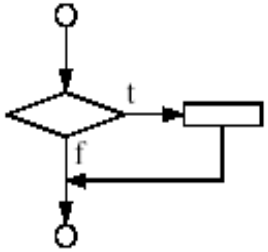
Tratan a los operandos como expresiones FALSAS o VERDADERAS.

# Estructuras de control

## Secuencial

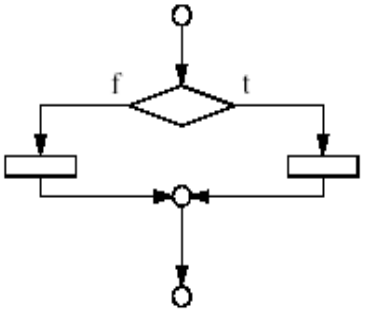
```
// Expresiones en secuencia
a = conta + 5;
. . .
contador++;
```

## Selección simple




```
if(expresión)
    Proposición
```

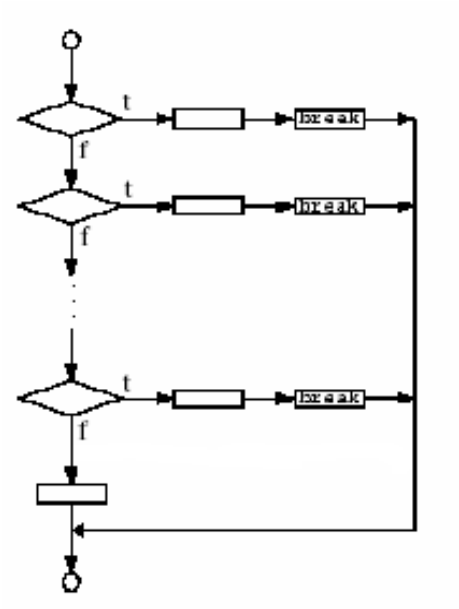
## Selección Doble




```
if(expresión)
    proposición1
else
    proposición2
```

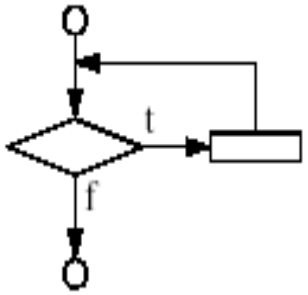
# Estructuras de control

Selección Múltiple




```
switch (expresión){
    case exp-const:
        proposiciones
    case exp-const:
        proposiciones
    default:
        proposiciones
}
```

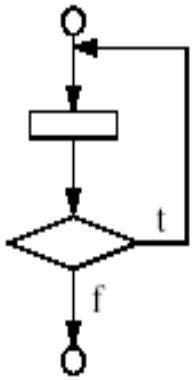
Repetición Mientras




```
while(expresión)
    proposición
```

# Estructuras de control

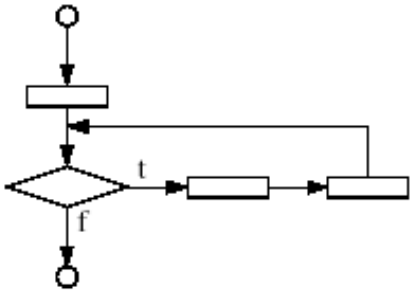
Repetición Hacer/Mientras




```
do
   proposición

while(expresión);
```

Repetición Para/Hasta




```
for (expr1; expr2; expr3)

   proposición
```

# Tipos de Memoria

## Datos en SRAM

**Variables:** Datos que van a ser leídos o escritos repetitivamente. SRAM este es el espacio de almacenamiento por default.

```
unsigned char x, y;
unsigned int  a, b, c;
```

Si es posible, el compilador utilizará los registros de propósito general (R0 a R31).

Los apuntadores son manejados en SRAM y hacen referencia a objetos de SRAM, un ejemplo de declaraciones y uso es:

```
char  cadena[]  = "hola mundo";
char  *pcad;

pcad = cadena;
```

# Datos en FLASH

**Constantes:** Datos que no cambiarán durante la ejecución normal de un programa.

Si las constantes se manejan en memoria FLASH, se liberarán espacios significativos de SRAM. Útil para cadenas de texto o tablas de búsqueda.

Se requiere el uso de la palabra reservada ***const*** e incluir al atributo **PROGMEM**, definido en la biblioteca ***pgmspace.h*** (en WinAVR).

Ejemplos de declaraciones:

**const char** cadena[] PROGMEM = “Cadena con un mensaje constante”;
**const unsigned char** tabla[] PROGMEM = { 0x24, 0x36, 0x48, 0x5A, 0x6C };

Declaración de constantes -> ámbito global.

Acceso -> funciones de la biblioteca ***pgmspace.h***

Ejemplos:

```
pgm_read_byte(address);          // Lee 8 bits
pgm_read_word(address);          // Lee 16 bits
pgm_read_dword(address);         // Lee 32 bits
```

Reciben como argumento la dirección del dato en FLASH.

```
x = pgm_read_byte(&tabla[i]);
```

Es posible el manejo de apuntadores, un apuntador a datos en memoria FLASH se declara como PGM_P.

La biblioteca incluye funciones que trabajan con bloques completos de memoria FLASH.

*strcpy_P:* Sirve para leer una cadena de memoria FLASH y depositarla en SRAM.

```
char    buf[32];        // Buffer destino, en SRAM
PGM_P   p;              // Apuntador a memoria FLASH

   p = cadena;          // p apunta a la cadena en FLASH
   strcpy_P(buf, p);    // copia de FLASH a SRAM
```

# Datos en EEPROM

Variables en las que se requiera conservar su contenido aún en ausencia de energía. Deberán almacenarse en la EEPROM.

Deberá utilizarse el atributo EEMEM, definido en la biblioteca *eeprom.h*, también en WinAVR.

Son declaraciones globales y se deben hacer antes de cualquier declaración enfocada hacia la SRAM.

```
#include <avr/eeprom.h>                  //Biblioteca para la EEPROM

EEMEM  int  contador = 0x1234;                  // Un entero:2 bytes
EEMEM  unsigned char  clave[4] = { 1, 2, 3, 4};          // 4 bytes
```

En la EEPROM:

34 12 01 02 03 04 FF FF FF . . .

# Acceso a los datos en la EEPROM

1) Por medio de los registros EEAR, EEDR y EECR, con funciones similares a las revisadas en la unidad anterior.

2) Funciones en la biblioteca *eeprom.h*, ejemplos:

   *eeprom_read_byte*: Lee un byte de la EEPROM, en la dirección que recibe como argumento.

   *eeprom_write_byte*: Escribe un byte en la EEPROM, como argumentos recibe la dirección y el dato.

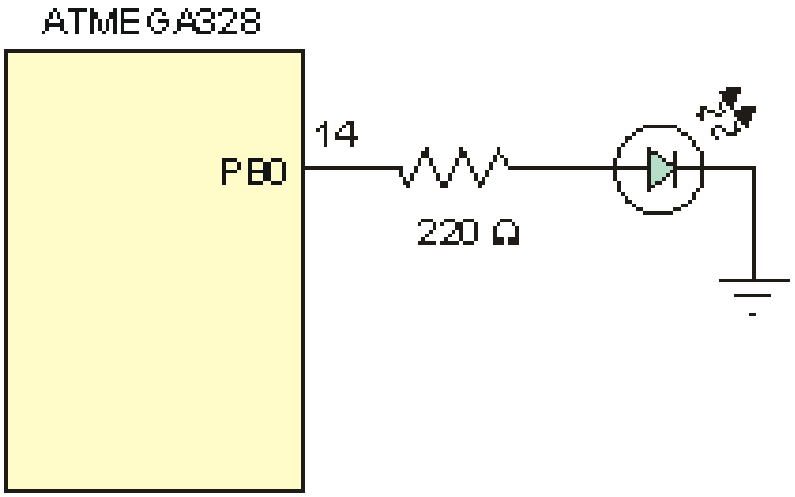
La biblioteca también incluye funciones para datos de 16 y 32 bits, así como para el manejo de bloques de memoria.

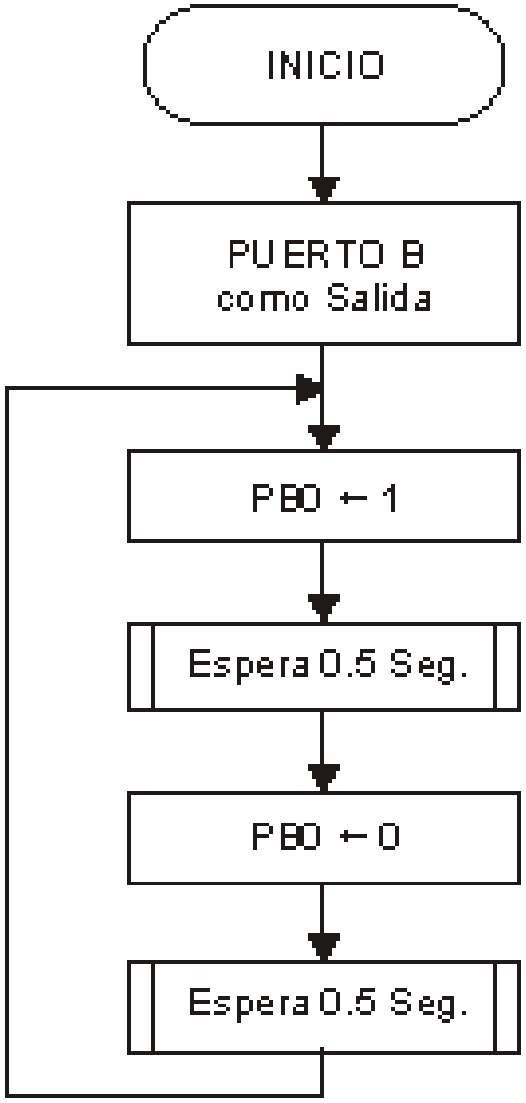
# PROGRAMAS DE EJEMPLO

En Ensamblador y en Lenguaje C

# Parpadeo de un LED

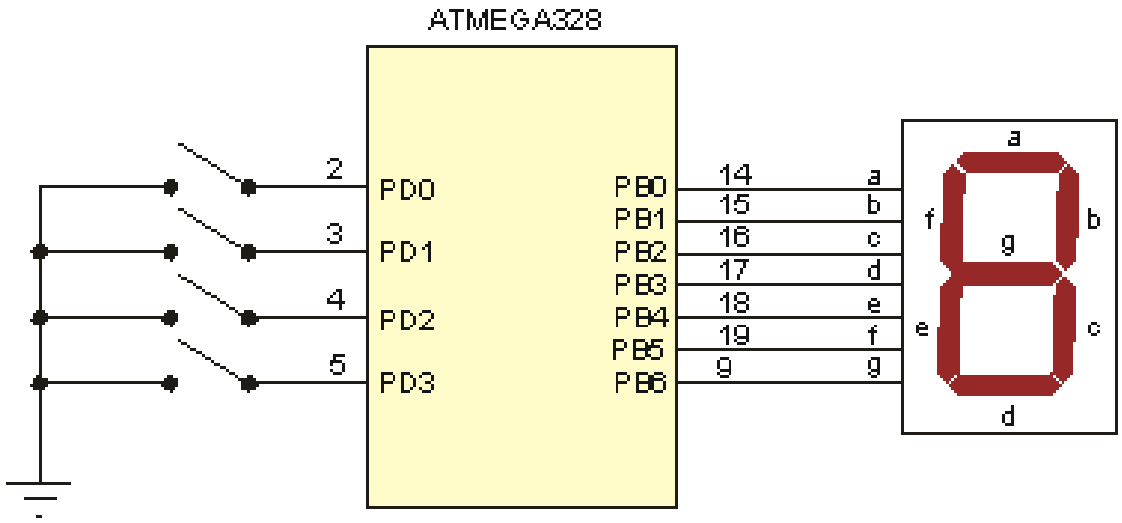
- Realice un programa que haga parpadear un LED conectado en la terminal PB0 a una frecuencia aproximada de 1 Hz (periodo de 1 seg.), considerando un ciclo útil del 50 % (½ seg. encendido y ½ seg. apagado).



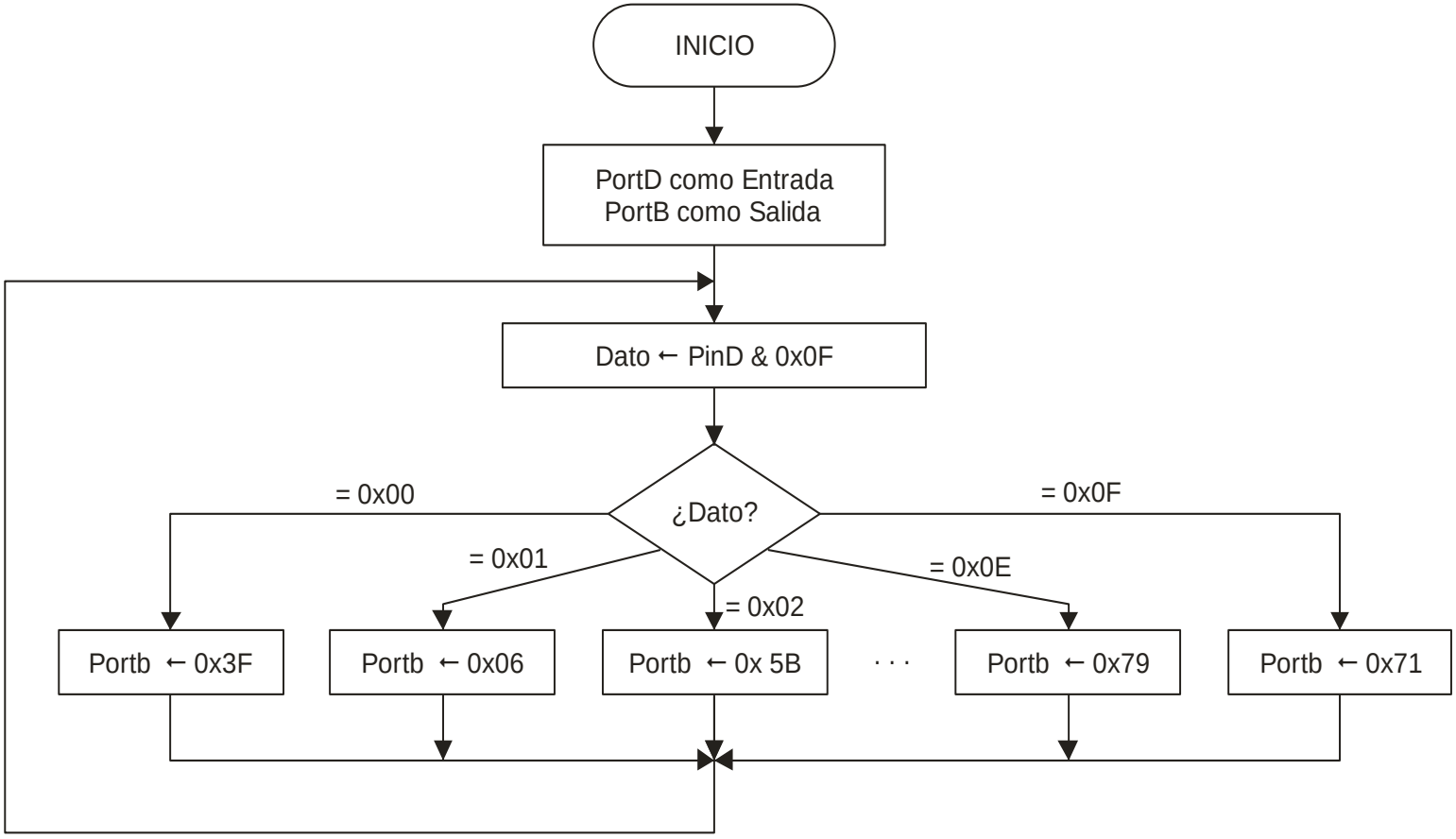
# Decodificador de binario a 7 Segmentos

- Desarrolle un programa que lea los 4 bits menos significativos del Puerto D [PD3:PD0] y genere su código en 7 segmentos en el Puerto B.



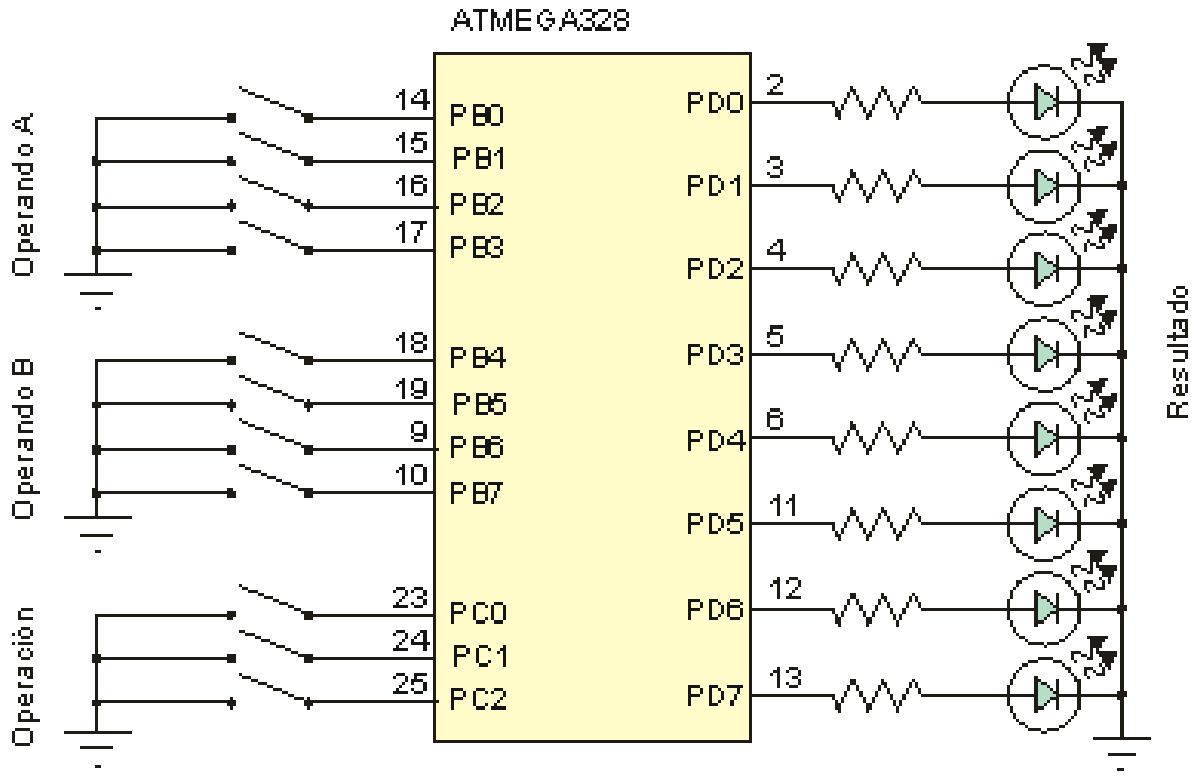

| Num | g | f | e | d | c | b | a | HEX |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0x3F |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0x06 |
| 2 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0x5B |
| 3 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0x4F |
| 4 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0x66 |
| 5 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0x6D |
| 6 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0x7D |
| 7 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0x07 |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0x7F |
| 9 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0x67 |
| A | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0x77 |
| B | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0x7C |
| C | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0x39 |
| D | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0x5E |
| E | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0x79 |
| F | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0x71 |

# Decodificador de binario a 7 Segmentos



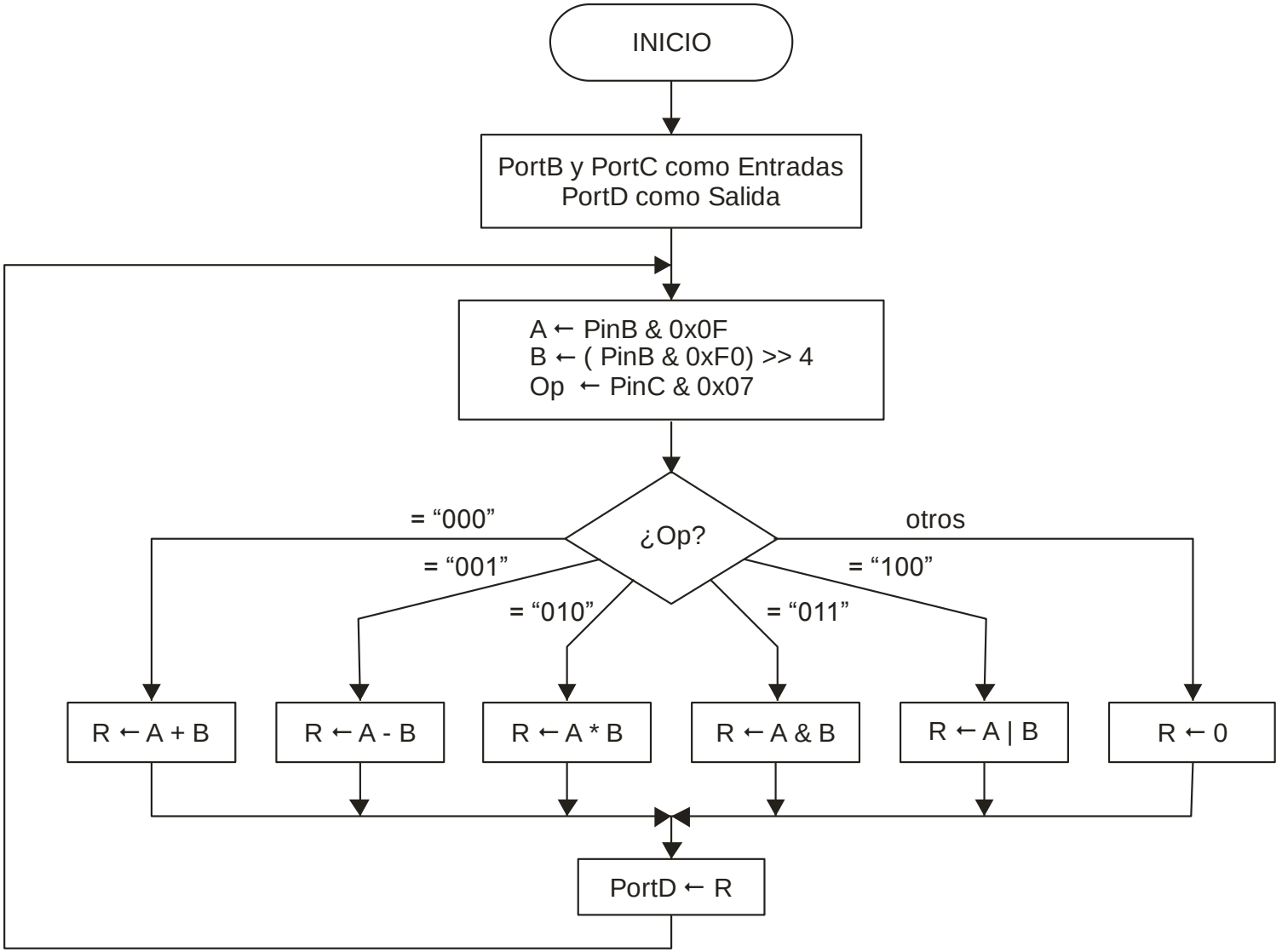
# Diseño de una ALU de 4 bits

- Construya una ALU de 4 bits utilizando un ATMega328P, en donde los operandos se lean del puerto B (nibble bajo para el operando A y nibble alto para el operando B), el resultado se genere en el puerto D y con los 3 bits menos significativos del puerto C se defina la operación.




| PortC[2:0] | Operación |
|---|---|
| 000 | R = A **+** B |
| 001 | R = A **-** B |
| 010 | R = A ***** B |
| 011 | R = A **AND** B |
| 100 | R = A **OR** B |
| 101 a 111 | R = 0 |

# Diseño de una ALU de 4 bits



# Máquina de estados (Puerta Automática)