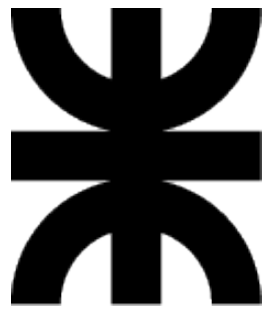
Informática II
Operadores a nivel de bits

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)
- Desplazamiento a la derecha: `>>` (mayor que)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)
- Desplazamiento a la derecha: `>>` (mayor que)
- Complemento: `~` (tilde)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)
- Desplazamiento a la derecha: `>>` (mayor que)
- Complemento: `~` (tilde)

Tienen su equivalente en operadores de asignación:

`&=` `|=` `^=` `<<=` `>>=`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)
- Desplazamiento a la derecha: `>>` (mayor que)
- Complemento: `~` (tilde)

Tienen su equivalente en operadores de asignación:

`&=` `|=` `^=` `<<=` `>>=`

Ejemplos:

```
a = b & c;
a = a & b;
a &= b;
```

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)
- Desplazamiento a la derecha: `>>` (mayor que)
- Complemento: `~` (tilde)

Tienen su equivalente en operadores de asignación:

`&=` `|=` `^=` `<<=` `>>=`

Ejemplos:

```
a = b & c;
a = a & b;
a &= b;
```

```
a = b | c;
a = a | b;
a |= b;
```

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`).

- `AND` a nivel de bit: `&` (ampersand o et)
- `OR` inclusivo a nivel de bit: `|` (barra vertical o pleca)
- `OR` exclusivo a nivel de bit: `^` (caret o sombrero)
- Desplazamiento a la izquierda: `<<` (menor que)
- Desplazamiento a la derecha: `>>` (mayor que)
- Complemento: `~` (tilde)

Tienen su equivalente en operadores de asignación:

`&=` `|=` `^=` `<<=` `>>=`

Ejemplos:

```
a = b & c;
a = a & b;
a &= b;
```

```
a = b | c;
a = a | b;
a |= b;
```

```
a = b << 1;
a = a << 2;
a <<= 2;
```

Operadores a nivel de bits

AND a nivel de bits

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Operadores a nivel de bits

AND a nivel de bits

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            0001 = 0x1 = 1d
```

Operadores a nivel de bits

AND a nivel de bits

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            0001 = 0x1 = 1d

185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               1001 1000 = 0x98 = 152d
```

Operadores a nivel de bits

AND a nivel de bits

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            0001 = 0x1 = 1d

185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               1001 1000 = 0x98 = 152d
```

OR a nivel de bits

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Operadores a nivel de bits

AND a nivel de bits

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            0001 = 0x1 = 1d

185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               1001 1000 = 0x98 = 152d
```

OR a nivel de bits

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            1101 = 0xD = 13d
```

Operadores a nivel de bits

AND a nivel de bits

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            0001 = 0x1 = 1d

185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               1001 1000 = 0x98 = 152d
```

OR a nivel de bits

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

```
5d = 0x5 = 0101
9d = 0x9 = 1001
           ------
            1101 = 0xD = 13d

185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               1011 1011 = 0xBB = 187d
```

Operadores a nivel de bits

OR exclusiva a nivel de bits

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

```
185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               0010 0011 = 0x23 = 35d
```

Operadores a nivel de bits

OR exclusiva a nivel de bits

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

```
185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               0010 0011 = 0x23 = 35d
```

Complemento

```
~170d = ~(1010 1010) = 0101 0101 = 0x55 = 85d
```

Operadores a nivel de bits

OR exclusiva a nivel de bits

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

```
185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               0010 0011 = 0x23 = 35d
```

Complemento

```
~170d = ~(1010 1010) = 0101 0101 = 0x55 = 85d
```

Desplazamiento a la izquierda

```
      22d << 2
0001 0110 << 2 = 0101 1000 = 0x58 = 88d
```

Operadores a nivel de bits

OR exclusiva a nivel de bits

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

```
185d = 0xB9 = 1011 1001
154d = 0x9A = 1001 1010
              ----------
               0010 0011 = 0x23 = 35d
```

Complemento

```
~170d = ~(1010 1010) = 0101 0101 = 0x55 = 85d
```

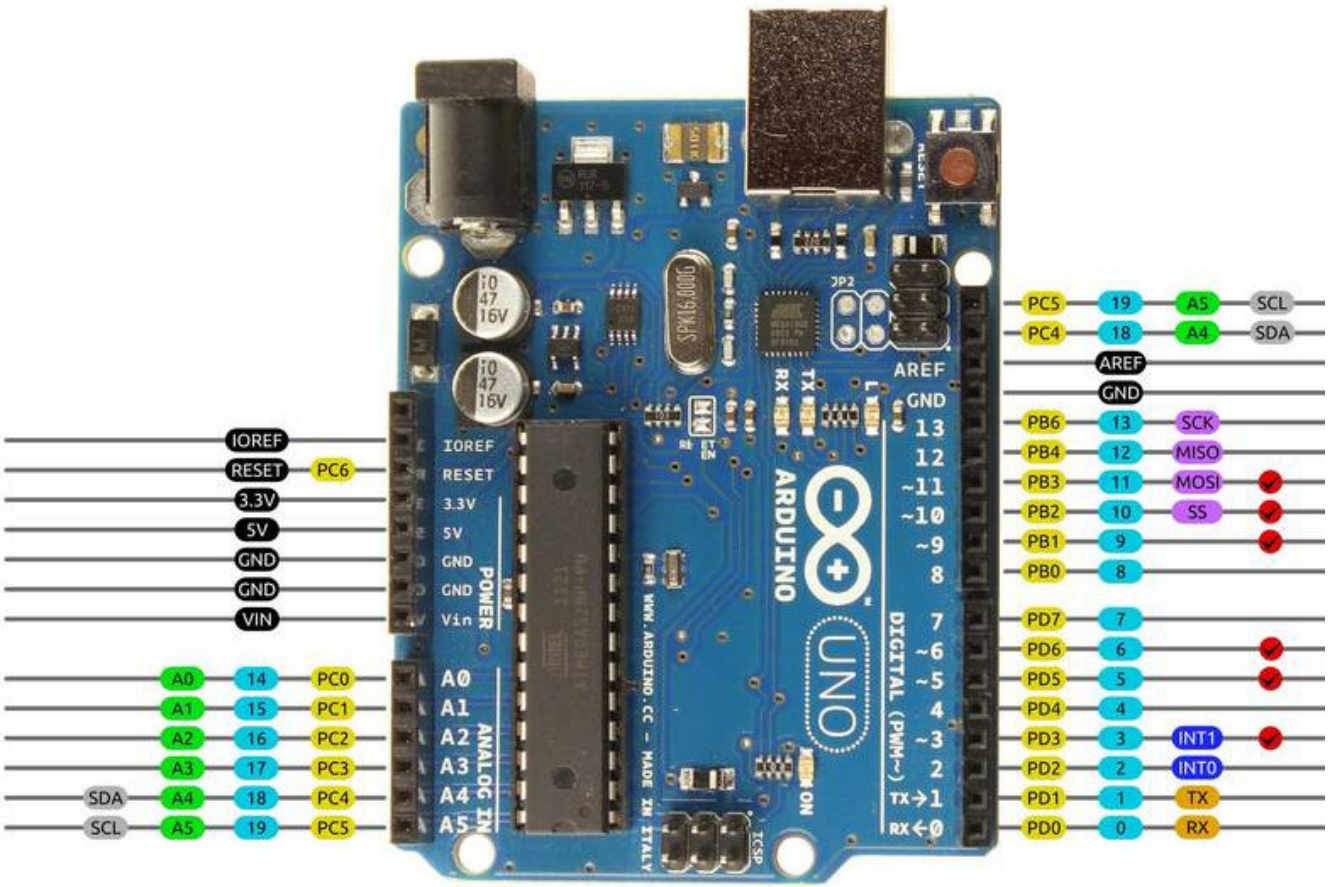
Desplazamiento a la izquierda

```
      22d << 2
0001 0110 << 2 = 0101 1000 = 0x58 = 88d
```

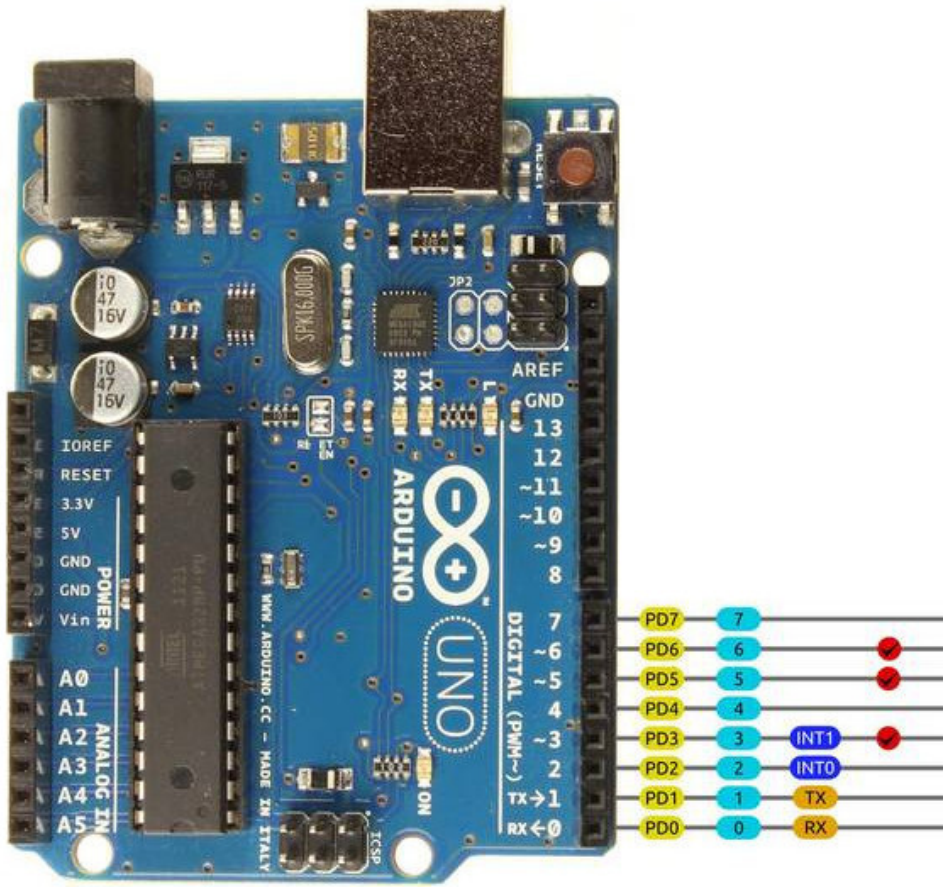
Desplazamiento a la derecha

```
      154d >> 3
1001 1010 >> 3 = 0001 0011 = 0x13 = 19d
```

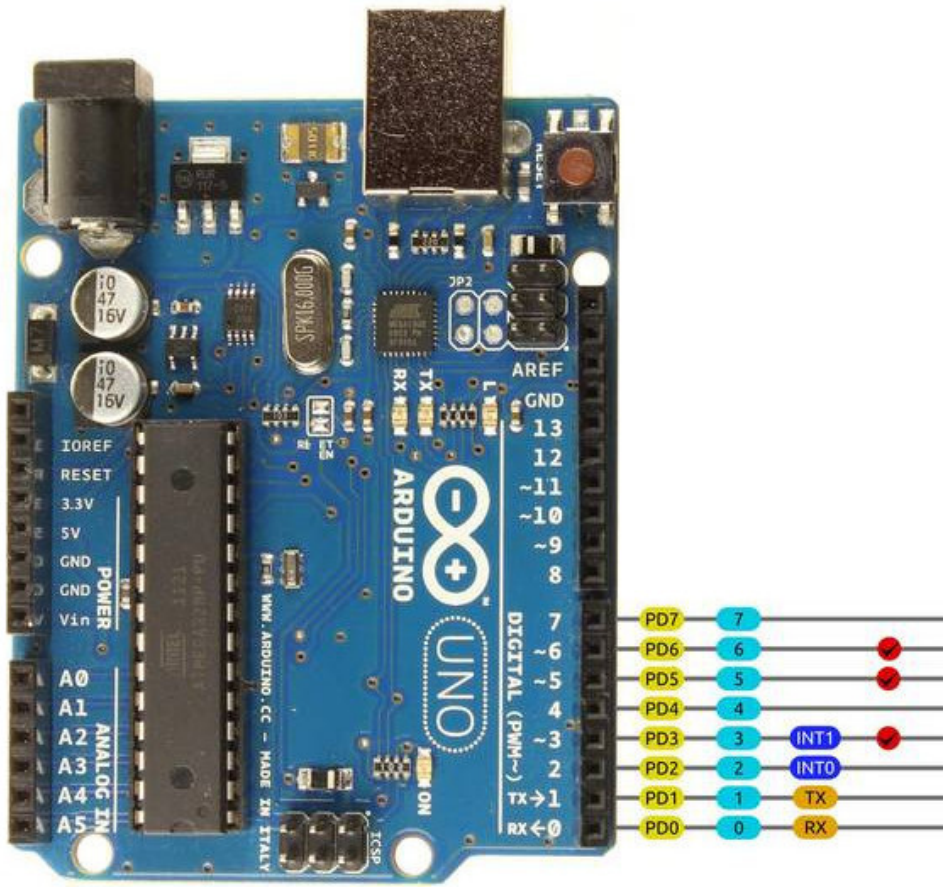
Operadores a nivel de bits



Operadores a nivel de bits

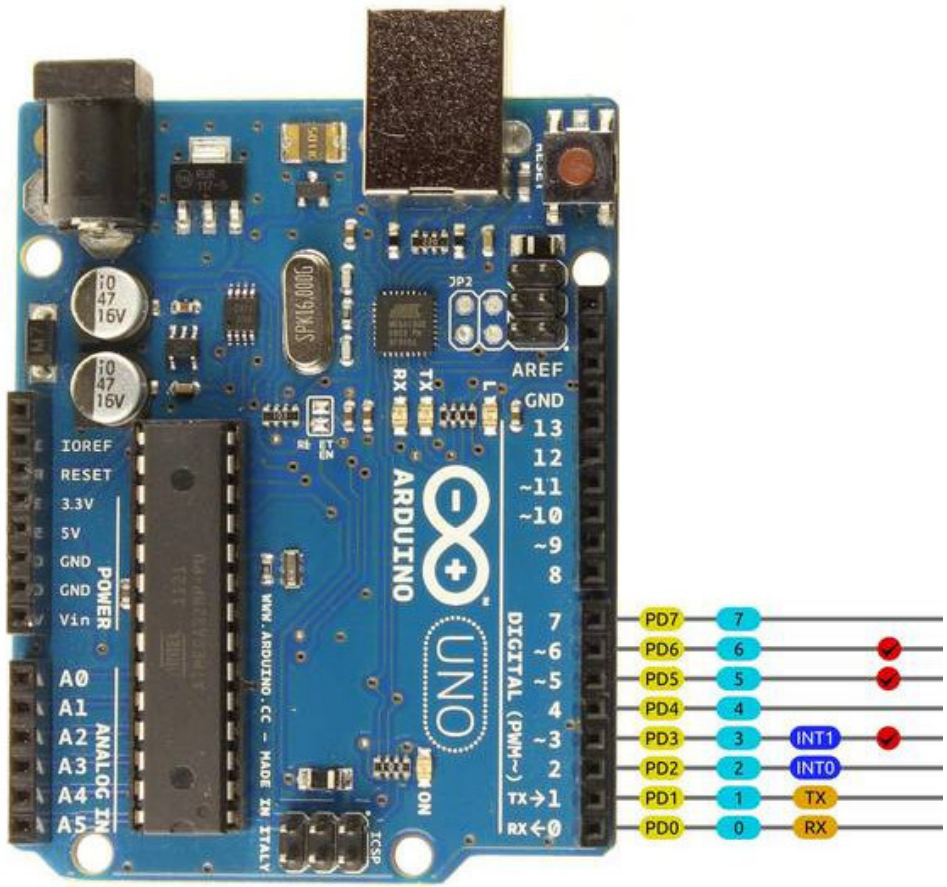


Operadores a nivel de bits



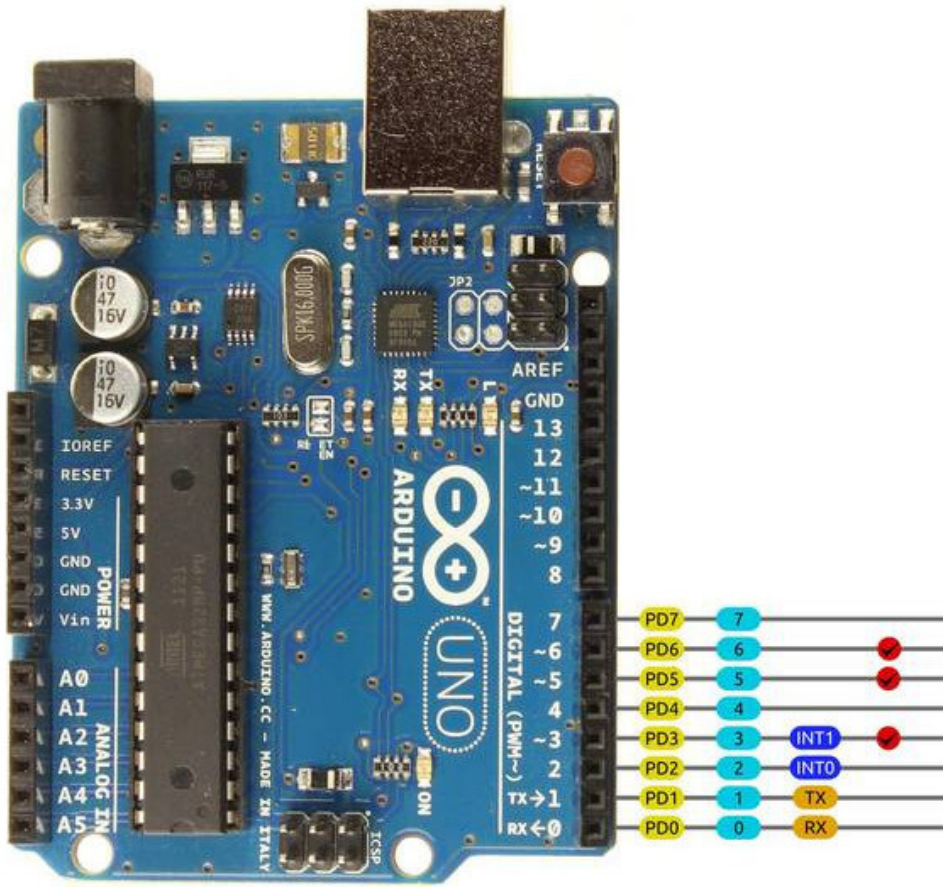
- ¿Cuantos bits tiene el puerto D?

Operadores a nivel de bits



- ¿Cuantos bits tiene el puerto D?
- ¿Qué tipo de dato se puede utilizar para almacenar el valor?

Operadores a nivel de bits



- ¿Cuantos bits tiene el puerto D?
- ¿Qué tipo de dato se puede utilizar para almacenar el valor?

```
unsigned char puerto;
```

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

- Valor en hexadecimal:

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

- Valor en hexadecimal: `0x6B`

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

- Valor en hexadecimal: `0x6B`

```
1 unsigned char puerto;
2
3 puerto = leerPuerto();
4 /* Modificar el valor de un bit */
5 escribirPuerto(puerto);
```

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

- Valor en hexadecimal: `0x6B`

```
1 unsigned char puerto;
2
3 puerto = leerPuerto();
4 /* Modificar el valor de un bit */
5 escribirPuerto(puerto);
```

Qué operación utilizar para:

- Poner a 1 un único bit sin alterar el resto

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

- Valor en hexadecimal: `0x6B`

```
1 unsigned char puerto;
2
3 puerto = leerPuerto();
4 /* Modificar el valor de un bit */
5 escribirPuerto(puerto);
```

Qué operación utilizar para:

- Poner a 1 un único bit sin alterar el resto
- Poner a 0 un único bit sin alterar el resto

Operadores a nivel de bits

Valor actual del puerto D:

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
0	1	1	0	1	0	1	1

- Valor en hexadecimal: `0x6B`

```
1 unsigned char puerto;
2
3 puerto = leerPuerto();
4 /* Modificar el valor de un bit */
5 escribirPuerto(puerto);
```

Qué operación utilizar para:

- Poner a 1 un único bit sin alterar el resto
- Poner a 0 un único bit sin alterar el resto
- Invertir el valor de un único bit sin alterar el resto

Operadores a nivel de bits

Pone a 1 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto |= 0x10; // Pone a 1 PD4
3 escribirPuerto(puerto);
```

Operadores a nivel de bits

Pone a 1 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto |= 0x10; // Pone a 1 PD4
3 escribirPuerto(puerto);
```

```
0x6B = 0 1 1 0 1 0 1 1
0x10 = 0 0 0 1 0 0 0 0
       ---------------- (OR)
       0 1 1 1 1 0 1 1
```

Operadores a nivel de bits

Pone a 1 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto |= 0x10; // Pone a 1 PD4
3 escribirPuerto(puerto);
```

```
0x6B = 0 1 1 0 1 0 1 1
0x10 = 0 0 0 1 0 0 0 0
       ---------------- (OR)
       0 1 1 1 1 0 1 1
```

Pone a 0 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto &= ~(0x20); // Pone a 0 PD5
3 escribirPuerto(puerto);
```

Operadores a nivel de bits

Pone a 1 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto |= 0x10; // Pone a 1 PD4
3 escribirPuerto(puerto);
```

```
0x6B = 0 1 1 0 1 0 1 1
0x10 = 0 0 0 1 0 0 0 0
       ---------------- (OR)
       0 1 1 1 1 0 1 1
```

Pone a 0 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto &= ~(0x20); // Pone a 0 PD5
3 escribirPuerto(puerto);
```

```
  0x6B = 0 1 1 0 1 0 1 1
~(0x20) = 1 1 0 1 1 1 1 1
          ---------------- (AND)
          0 1 0 0 1 0 1 1
```

Operadores a nivel de bits

Pone a 1 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto |= 0x10; // Pone a 1 PD4
3 escribirPuerto(puerto);
```

```
0x6B = 0 1 1 0 1 0 1 1
0x10 = 0 0 0 1 0 0 0 0
       ---------------- (OR)
       0 1 1 1 1 0 1 1
```

Pone a 0 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto &= ~(0x20); // Pone a 0 PD5
3 escribirPuerto(puerto);
```

```
  0x6B = 0 1 1 0 1 0 1 1
~(0x20) = 1 1 0 1 1 1 1 1
          ---------------- (AND)
          0 1 0 0 1 0 1 1
```

Invierte el valor de un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto ^= 0x02; // Invierte PD1
3 escribirPuerto(puerto);
```

Operadores a nivel de bits

Pone a 1 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto |= 0x10; // Pone a 1 PD4
3 escribirPuerto(puerto);
```

```
0x6B = 0 1 1 0 1 0 1 1
0x10 = 0 0 0 1 0 0 0 0
       ---------------- (OR)
       0 1 1 1 1 0 1 1
```

Pone a 0 un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto &= ~(0x20); // Pone a 0 PD5
3 escribirPuerto(puerto);
```

```
  0x6B = 0 1 1 0 1 0 1 1
~(0x20) = 1 1 0 1 1 1 1 1
          ---------------- (AND)
          0 1 0 0 1 0 1 1
```

Invierte el valor de un bit:

```
1 unsigned char puerto = leerPuerto();
2 puerto ^= 0x02; // Invierte PD1
3 escribirPuerto(puerto);
```

```
0x6B = 0 1 1 0 1 0 1 1
0x10 = 0 0 0 0 0 0 1 0
       ---------------- (exOR)
       0 1 1 0 1 0 0 1
```

Operadores de desplazamiento

Desplazamiento a la izquierda

- Los valores desplazados se pierden
- Los valores a la derecha se rellenan con ceros

Operadores de desplazamiento

Desplazamiento a la izquierda

- Los valores desplazados se pierden
- Los valores a la derecha se rellenan con ceros

Desplazamiento a la derecha

- Los valores desplazados se pierdes
- Los valores a la izquierda dependen del tipo de dato (`signed`/`unsigned`)

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3
4
5
6
7
8
9
10
11
12
13
14
15 }
```

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3    unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);
4
5
6
7
8
9
10
11
12
13
14
15 }
```

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3    unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);
4
5
6
7
8
9
10
11
12
13
14
15 }
```

- ¿Cuanto es `sizeof(unsigned int)`?

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3    unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);
4
5
6
7
8
9
10
11
12
13
14
15 }
```

- ¿Cuanto es `sizeof(unsigned int)`? 4

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3    unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);
4
5
6
7
8
9
10
11
12
13
14
15 }
```

- ¿Cuanto es `sizeof(unsigned int)`? 4

```
unsigned int mask = 1 << 31; // 0x80000000
```

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3    unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);
4
5    for(b = 1; b <= 8*sizeof(unsigned int); b++)
6    {
7
8
9
10
11
12   }
13
14
15 }
```

- ¿Cuanto es `sizeof(unsigned int)`? 4

```
unsigned int mask = 1 << 31; // 0x80000000
```

Imprimir variables en binario

```
1  void imprimir_binario(unsigned int val)
2  {
3    unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);
4
5    for(b = 1; b <= 8*sizeof(unsigned int); b++)
6    {
7      putchar(val & mask ? '1' : '0');
8      val <<= 1;
9
10
11
12   }
13
14
15 }
```

- ¿Cuanto es `sizeof(unsigned int)`? 4

```
unsigned int mask = 1 << 31; // 0x80000000
```

Imprimir variables en binario

```c
void imprimir_binario(unsigned int val)
{
  unsigned int b, mask = 1 << (8*sizeof(unsigned int)-1);

  for(b = 1; b <= 8*sizeof(unsigned int); b++)
  {
    putchar(val & mask ? '1' : '0');
    val <<= 1;

    if(b % 8 == 0)
      putchar(' ');
  }

  putchar('\n');
}
```

- ¿Cuanto es `sizeof(unsigned int)`? 4

```c
unsigned int mask = 1 << 31; // 0x80000000
```