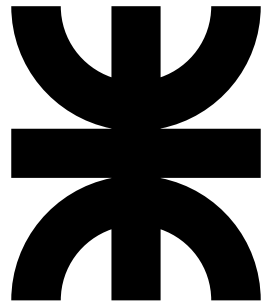
Informática II
Uniones y manipulación de bits

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2019 –

Uniones – Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Uniones – Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

Uniones – Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

Uniones – Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

- Ocupa en memoria lo suficiente para contener el miembro más grande

Uniones – Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

- Ocupa en memoria lo suficiente para contener el miembro más grande
- En general contienen dos o más tipos de datos

Uniones – Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

- Ocupa en memoria lo suficiente para contener el miembro más grande
- En general contienen dos o más tipos de datos
- En cada momento se puede referenciar un tipo de dato

Uniones – Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de una unión

Uniones – Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de una unión

Se pueden comparar las uniones?

Uniones – Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de una unión

Se pueden comparar las uniones? NO

Uniones – Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de una unión

Se pueden comparar las uniones? NO . . . y las estructuras?

Uniones – Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de una unión

Se pueden comparar las uniones? NO . . . y las estructuras? NO

Uniones – Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (&) de una unión

Se pueden comparar las uniones? NO . . . y las estructuras? NO

Inicialización: Se puede inicializar en la declaración con un valor del mismo tipo que el primer miembro de la union

```
union numero {
  int x;
  float y;
};

union numero u1 = {10};
union numero u1 = {0.02} /* Qué hace? */;
```

Ejemplo de union `int` y `float`

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};
```

Ejemplo de union `int` y `float`

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};

void imprimir_union_int_float(union int_float u)
{
  printf("- El miembro INT de la union es:   %d\n", u.entero);
  printf("- El miembro FLOAT de la union es: %f\n", u.real);
}
```

Ejemplo de union `int` y `float`

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};

void imprimir_union_int_float(union int_float u)
{
  printf("- El miembro INT de la union es:   %d\n", u.entero);
  printf("- El miembro FLOAT de la union es: %f\n", u.real);
}

int main(void)
{
  union int_float u;

  /* Solicitar valor 'int' e imprimir union */
  /* Solicitar varlor 'float' e imprimir union */

  return 0;
}
```

Más ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;
```

Más ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

Más ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

```
union uint_bytes {
  unsigned int uint;
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
} reg2;
```

Más ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

```
union uint_bytes {
  unsigned int uint;
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
} reg2;

printf("%d␣", reg2.bytes.byte0);
printf("%d␣", reg2.bytes.byte1);
printf("%d␣", reg2.bytes.byte2);
printf("%d␣", reg2.bytes.byte3);
```

Más ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

```
union uint {
  unsigned int uint;
  unsigned char ucvec[4];
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
};
```

```
union uint_bytes {
  unsigned int uint;
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
} reg2;

printf("%d␣", reg2.bytes.byte0);
printf("%d␣", reg2.bytes.byte1);
printf("%d␣", reg2.bytes.byte2);
printf("%d␣", reg2.bytes.byte3);
```

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`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

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

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

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

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

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

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

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

Ver código de ejemplo.

Operadores a nivel de bits

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

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

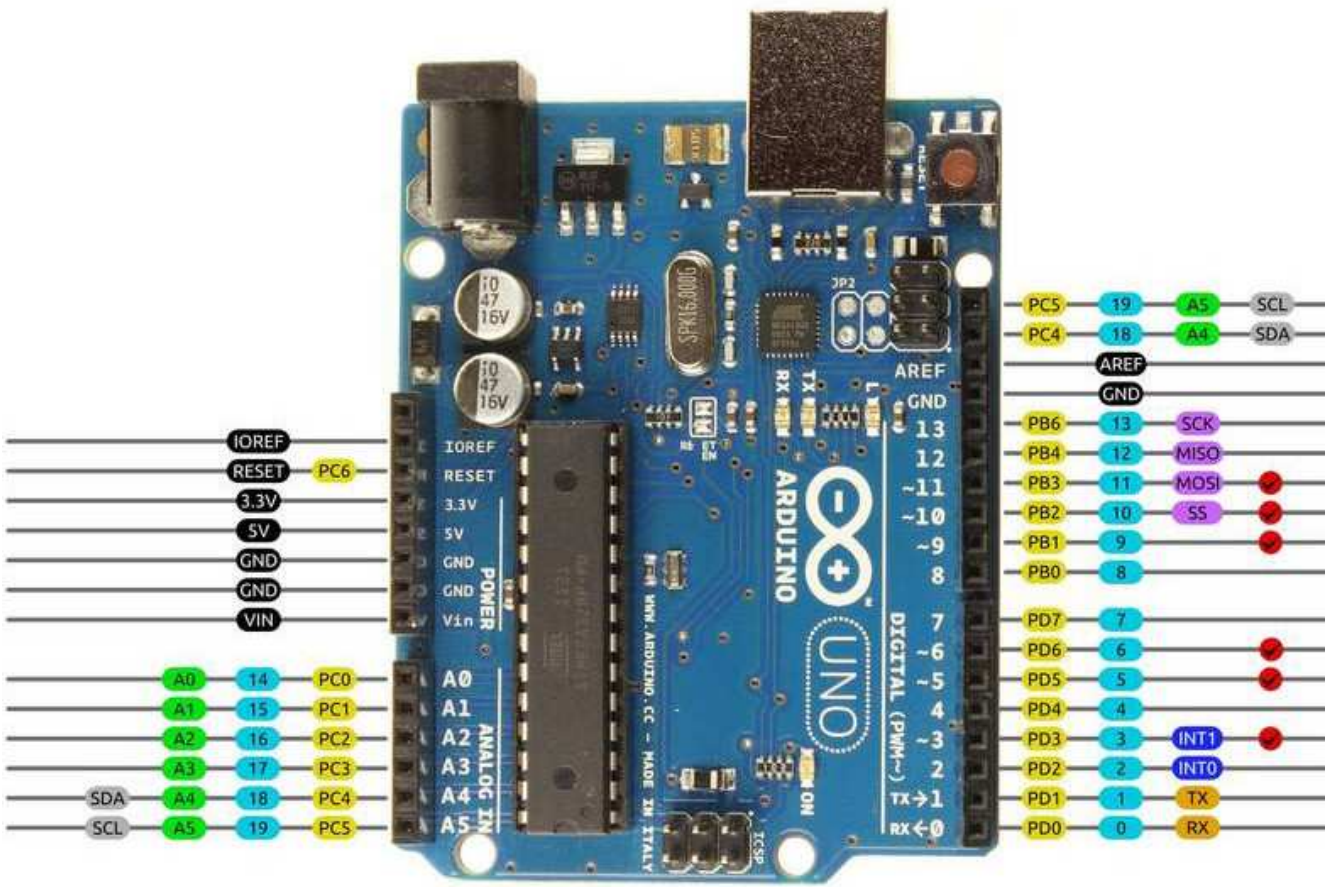
a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

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

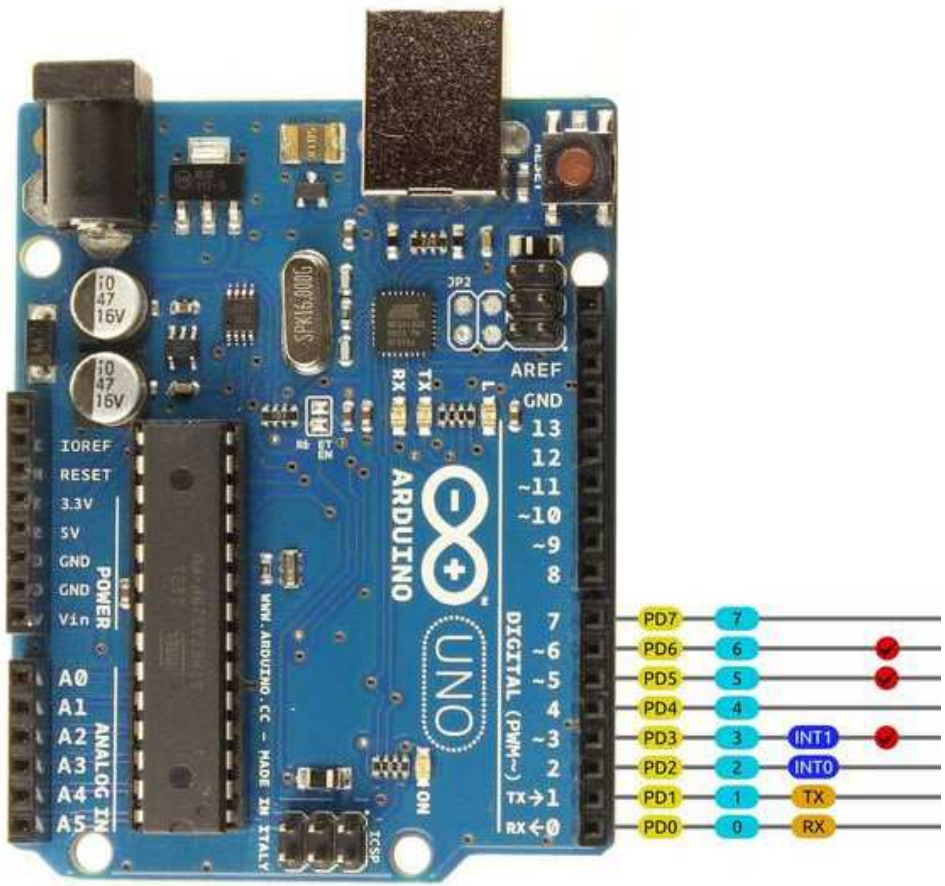
a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Cómo forzar a cero/uno, uno o varios bits?. . . y cómo hacer toggle?

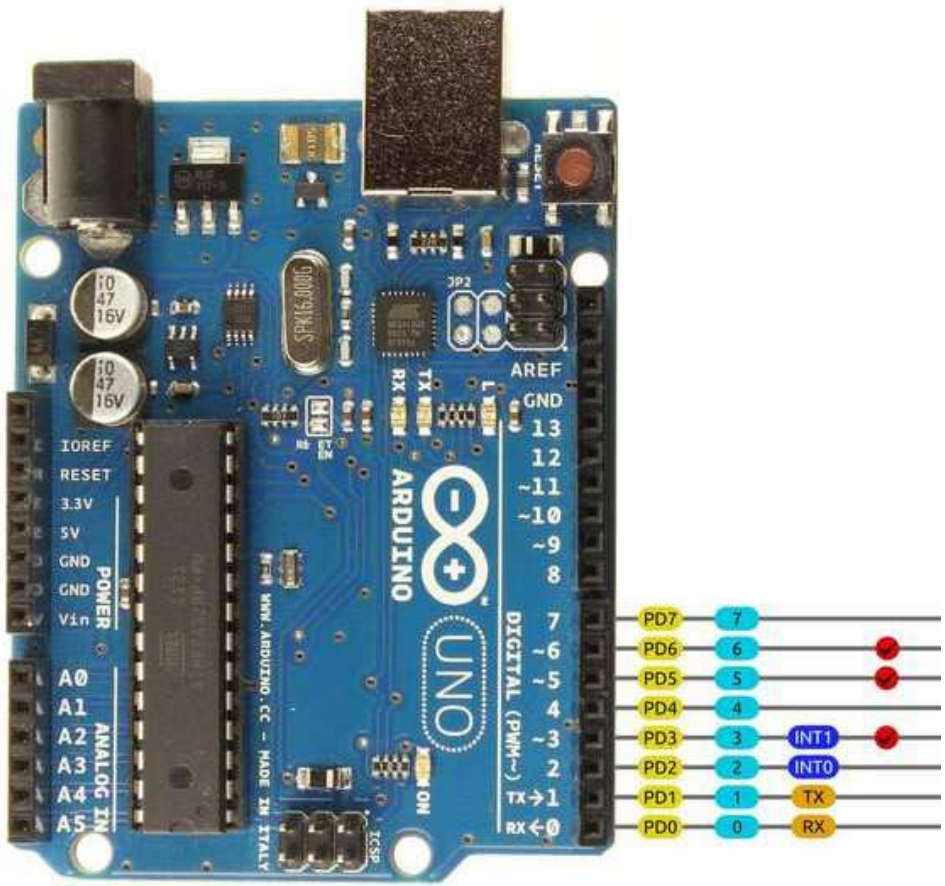
Operadores a nivel de bits



Operadores a nivel de bits

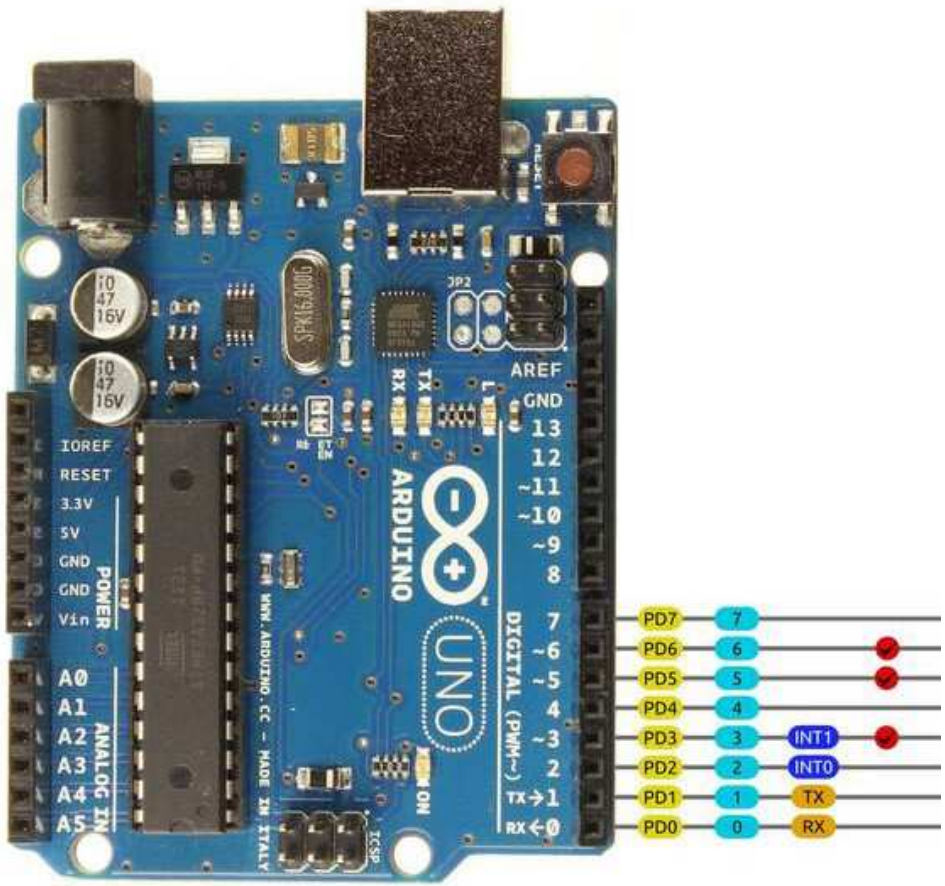


Operadores a nivel de bits



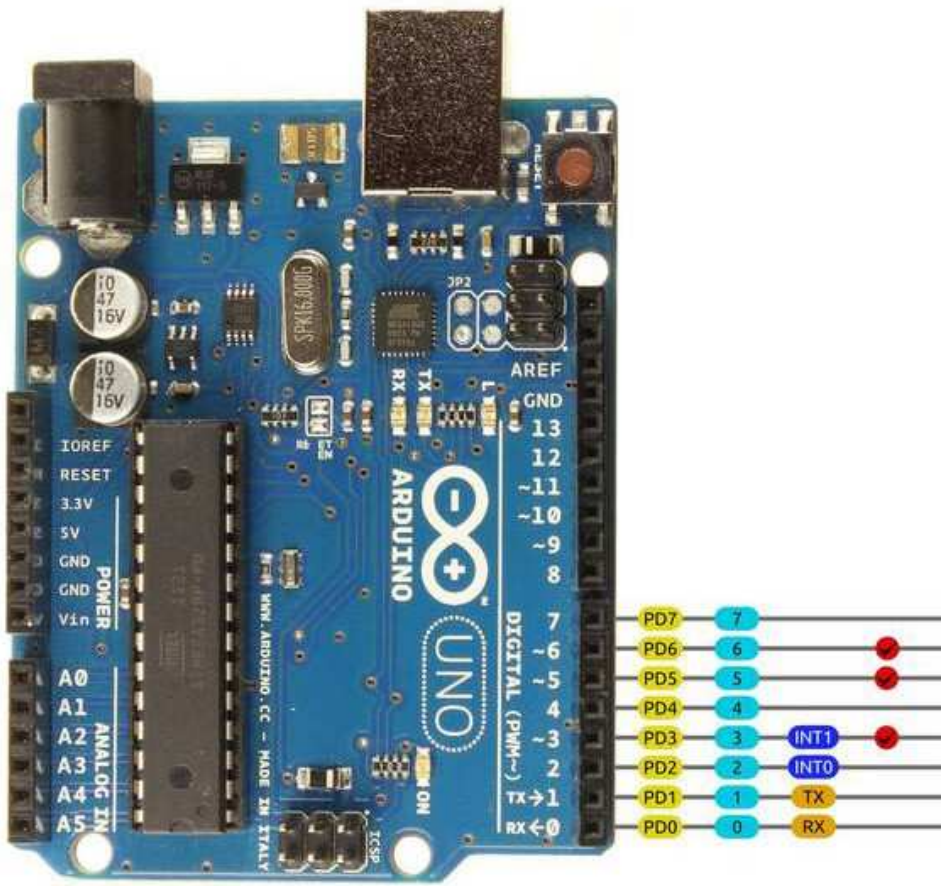
- ¿Cuantos bits tiene el puerto D?

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 &= ~(0x10); // Pone a 1 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 &= ~(0x10); // Pone a 1 PD5
3 escribirPuerto(puerto);
```

```
  0x6B = 0 1 1 0 1 0 1 1
~(0x10) = 1 1 1 0 1 1 1 1
          --------------- (AND)
          0 1 1 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 &= ~(0x10); // Pone a 1 PD5
3 escribirPuerto(puerto);
```

```
  0x6B = 0 1 1 0 1 0 1 1
~(0x10) = 1 1 1 0 1 1 1 1
          --------------- (AND)
          0 1 1 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 &= ~(0x10); // Pone a 1 PD5
3 escribirPuerto(puerto);
```

```
  0x6B = 0 1 1 0 1 0 1 1
~(0x10) = 1 1 1 0 1 1 1 1
          --------------- (AND)
          0 1 1 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 char val)
2  {
3
4
5
6
7
8
9
10
11
12
13
14
15 }
```

Imprimir variables en binario

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

Imprimir variables en binario

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

- ¿Cuanto es `sizeof(unsigned char)`?

Imprimir variables en binario

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

- ¿Cuanto es `sizeof(unsigned char)`? 8

Imprimir variables en binario

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

- ¿Cuanto es `sizeof(unsigned char)`? 8

```
unsigned char mask = 1<<7; // 1000 0000
```

Imprimir variables en binario

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

- ¿Cuanto es `sizeof(unsigned char)`? 8

```
unsigned char mask = 1<<7; // 1000 0000
```

Imprimir variables en binario

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

- ¿Cuanto es `sizeof(unsigned char)`? 8

```
unsigned char mask = 1<<7; // 1000 0000
```

Imprimir variables en binario

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

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

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

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

- ¿Cuanto es `sizeof(unsigned char)`? 8

```c
unsigned char mask = 1<<7; // 1000 0000
```

Campos de bits

Permite definir el número de bits en el cual se almacenan los miembros `unsigned` o `int` de una estructura o de una union.

Campos de bits

Permite definir el número de bits en el cual se almacenan los miembros `unsigned` o `int` de una estructura o de una union.

Los miembros de los campos de bits deben ser declarados como `unsigned` o `int`

Campos de bits

Permite definir el número de bits en el cual se almacenan los miembros `unsigned` o `int` de una estructura o de una union.

Los miembros de los campos de bits deben ser declarados como `unsigned` o `int`

```
struct bitCard {
  unsigned face : 4;
  unsigned suit : 2;
  unsigned color : 1;
};
```

- Nombre del campo seguido de dos puntos : y una constante entera del *ancho del campo*
- La cantidad de bits se fija según el rango de valores de cada miembro
- El acceso a los miembros se realiza como en cualquier estructura

Campos de bits – Ejemplos

Byte/registro para manejo de colores

```
struct RGB_color {
  unsigned char r : 2; /* 2-bits */
  unsigned char g : 2;
  unsigned char b : 2;
  unsigned char : 2; /* padding */
};
```

Campos de bits – Ejemplos

Byte/registro para manejo de colores

```
struct RGB_color {
  unsigned char r : 2; /* 2-bits */
  unsigned char g : 2;
  unsigned char b : 2;
  unsigned char : 2; /* padding */
};
```

Y si se quisiera modificar los bits RGB todos juntos?

Campos de bits – Ejemplos

Byte/registro para manejo de colores

```
struct RGB_color {
  unsigned char r : 2; /* 2-bits */
  unsigned char g : 2;
  unsigned char b : 2;
  unsigned char : 2; /* padding */
};
```

Y si se quisiera modificar los bits RGB todos juntos?

```
union RGB_color {
  struct {
    unsigned char r:2, g:2, b:2;
    unsigned char : 2;
  };
  struct {
    unsigned char rgb : 6;
    unsigned char : 2;
  };
};
```

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

3. Modificar la función `imprimir_binario` para que imprima cualquier variable de tipo entero utilizando un `typedef`.

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

3. Modificar la función `imprimir_binario` para que imprima cualquier variable de tipo entero utilizando un `typedef`.

4. Modificar la función `imprimir_binario` para que imprima un `float`.

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

3. Modificar la función `imprimir_binario` para que imprima cualquier variable de tipo entero utilizando un `typedef`.

4. Modificar la función `imprimir_binario` para que imprima un `float`.

5. Modificar el programa anterior para que reciba el valor `float` por la línea de comandos, para que pueda ser ejecutado y muestre la salida como:

```
./float2bin 123.123
123.123001 = 11111010 00111110 11110110 01000010
```