UPMDIE
INDUSTRIALES

6 - Interrupciones

- *Conceptos generales*
- *Interrupciones externas*
- *Interrupciones temporales*
- *Ejemplos*

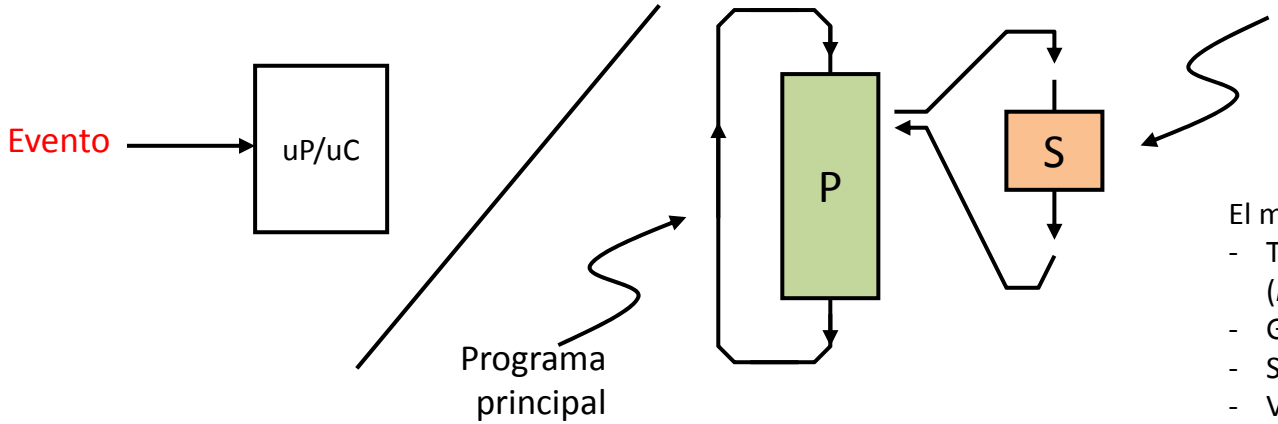
cei@upm.es

POLITÉCNICA

Interrupciones: Conceptos generales (I)

¿Qué es una interrupción?

- Es el desvío de la ejecución normal de un programa a petición de un periférico
- El origen de la petición es un evento externo al uP, y por tanto, asíncrono respecto al programa (no podemos predecir en qué parte del programa, ni cuándo, ocurrirá)
- Si el micro la acepta, ejecutará una función asociada para atender al periférico

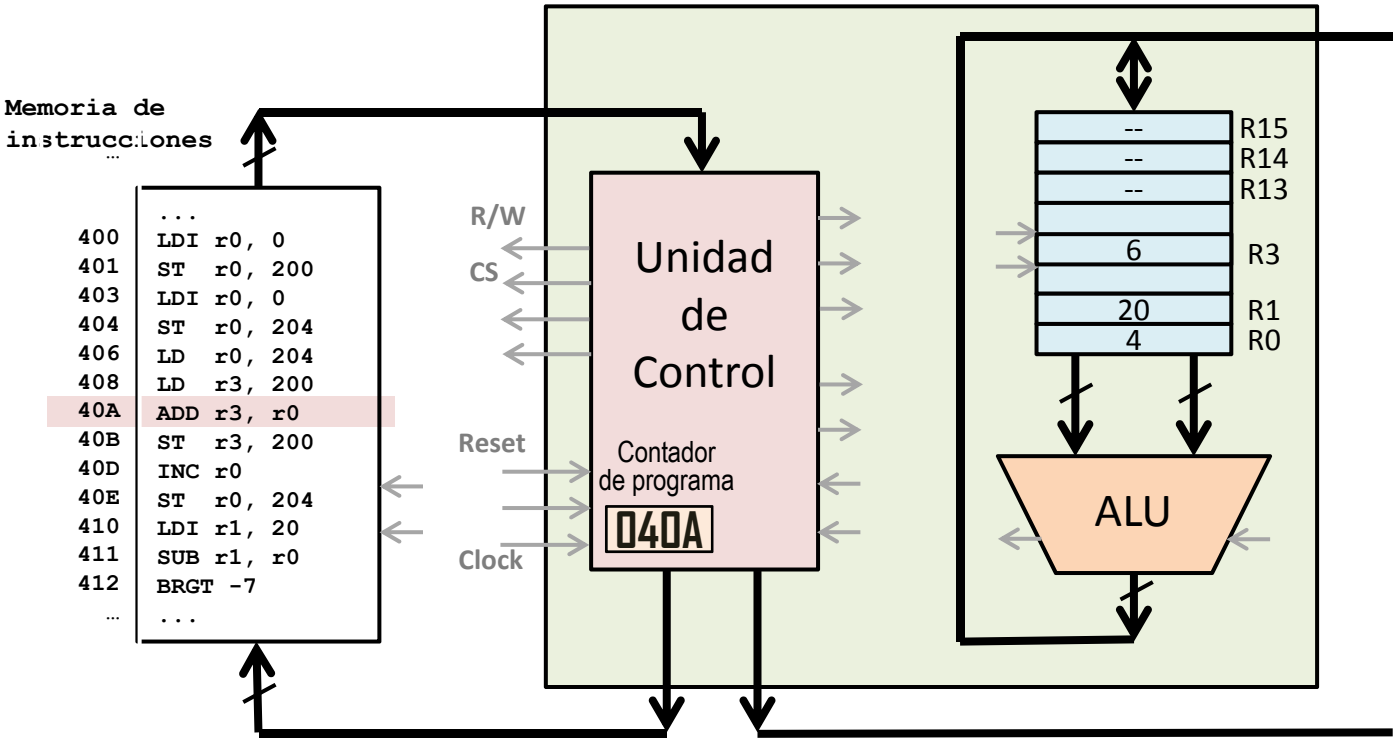


El micro debe (lo hace automáticamente):
- Terminar la instrucción en curso (*no en C, sino en código máquina*)
- Guardar el estado del micro
- Saltar a la subrutina y ejecutarla
- Volver (*dejándolo todo igual, en principio*)

La importancia de las interrupciones en el control

- El control se simplifica, ya que se atiende a los periféricos cuando éstos lo requieren, y no es necesario hacer *polling* o bloqueo
- En ocasiones, un programa de control está formado por un bucle de espera, en el que no se hace nada, y sólo se atienden las interrupciones que van llegando

Al atender una interrupción



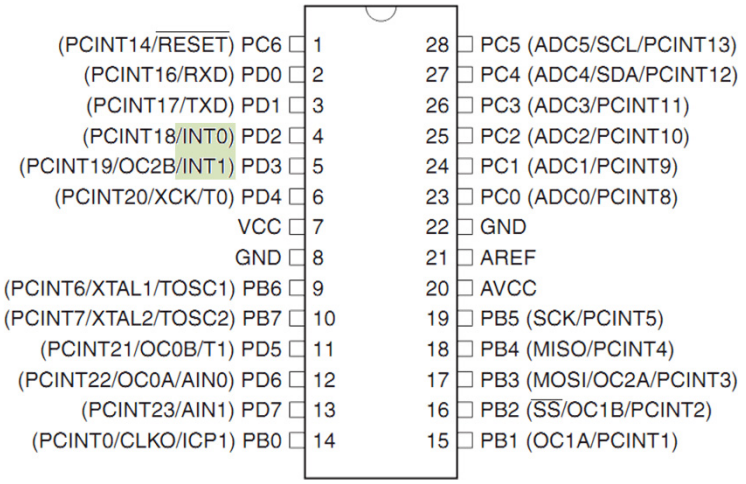
Qué hace el micro cuando tiene que atender una interrupción:

1 – Guarda el valor de Contador de programa
2 – Guarda los registros y el "estado" del micro
3 – Copia en el Contador de programa la dirección donde está la subrutina de interrupción
4 – Ejecuta el código en la nueva posición hasta que aparezca un 'return'
5 – Restauran el valor del Contador de programa, el estado del micro y los registros

Interrupciones externas

- Casi todos los uC disponen de señales externas que pueden producir interrupciones en determinadas condiciones (flanco de subida o bajada, nivel…)

- En el caso del Atmega168, todos los pines de E/S permiten producir interrupciones externas, pero sólo dos del puerto D son configurables (por flanco de subida, bajada, ambos…). *Sólo trataremos estos*.



Funciones de la API de Arduino para trabajar con las interrupciones externas:

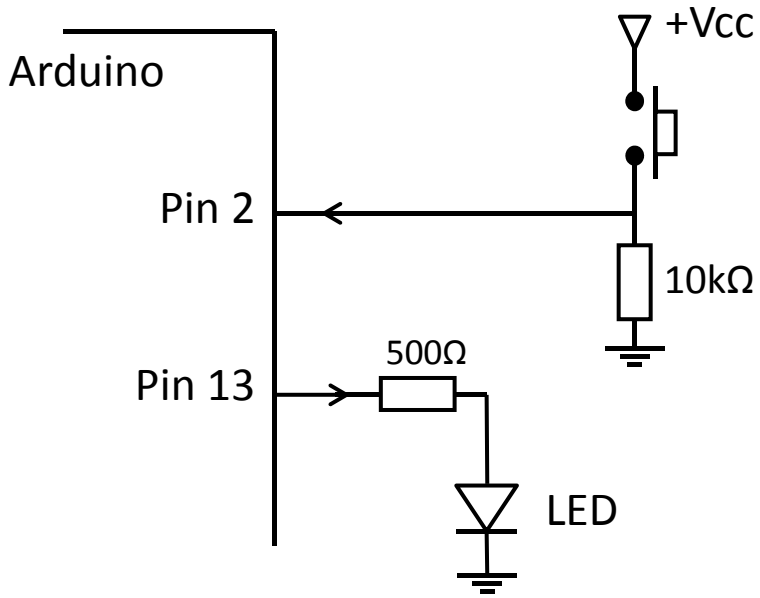
Los pines 2 y 3 de la placa de Arduino están conectados a PD2 y PD3, y por lo tanto tienen la posibilidad de interrupciones externas

- **attachInterrupt**(int_num, isr_name, condition); // Asocia una rutina a una interrupción bajo una determinada condición
  int_num : 0 para el pin 2, 1 para el pin 3
  isr_name: nombre de la rutina de servicio (función que no debe tener parámetros)
  condition: condición que produce la interrupción (FALLING, RISING, BOTH, LOW)

- **detachInterrupt**(int_num); // Desactiva la interrupción

Ejemplo: Controlar un LED con interrupciones

- Utilizando la API de Arduino ¿Qué habría que hacer para conmutar el LED cada vez que se pulsa el botón?




```
#define LED 13

void setup() {
  pinMode( LED , OUTPUT );
  // Asocio la función toggleLed a la interrupción por PD2
  attachInterrupt( 0, toggleLed, RISING );
}

// Función que se llamará en la interrupción
// No debe tener parámetros
void toggleLed() {
  int ledVal = digitalRead( LED );
  digitalWrite( LED, !ledVal );
}

void loop() {
  // Aquí no hay por qué hacer nada
}
```

⁇ Al comprobar el funcionamiento del sistema se observa que al pulsar o soltar, el led cambia de estado de forma no prevista, a veces cambia cuando no debe y otras no cambia cuando debe (debería cambiar de estado sólo cada vez que pulso)
→¿A qué puede ser debido ese comportamiento?
→¿Cómo podría solucionarse?

Interrupciones temporales y periódicas

- Los uC disponen de contadores programables que permiten producir interrupciones de forma periódica o pasado un cierto tiempo (en el caso del AtMega168, dispone de 3 contadores que permiten generar interrupciones periódicas y temporales de diversas formas con periodos desde pocos µseg hasta 32 segs).

Funciones de la API de para trabajar con las interrupciones periódicas y temporales (ATENCIÓN: estas funciones son un añadido a la API de Arduino hecho para esta asignatura. Para trabajar con ellas es necesario instalar un parche a la API de Arduino. Seguir las instrucciones en la descripción del trabajo voluntario).

- **attachPeriodicInterrupt**(isr_name, period_in_ms); // Asocia una rutina a una interrupción periódica
  isr_name: nombre de la rutina de servicio (función que no debe tener parámetros)
  period_in_ms: periodo en milisegundos. Debe ser un entero

- **detachPeriodicInterrupt**(isr_name); // Elimina la asociación

- **attachTimedInterrupt** (isr_name, time_in_ms); // Asocia una rutina para que se ejecute
  // una sola vez pasado el tiempo especificado

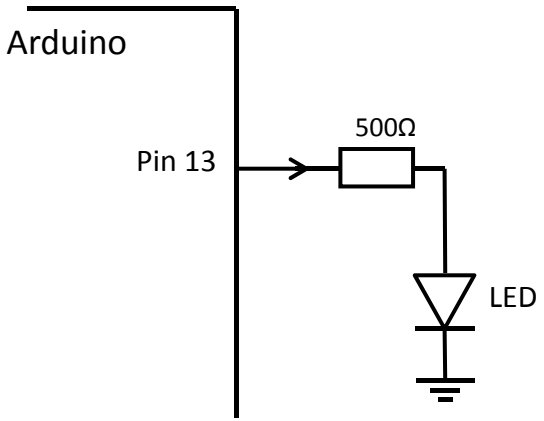
- **detachTimedInterrupt**(isr_name); // Elimina la asociación

Otras funciones asociadas con las interrupciones:

- **interrupts**(); // Habilita las interrupciones a nivel global
- **noInterrupts**(); // Deshabilita las interrupciones a nivel global

Ejemplo: Parpadear un LED con interrupciones

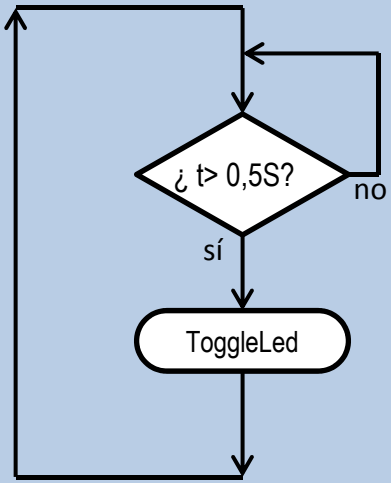
- Utilizando la API de Arduino ¿Qué habría que hacer para conmutar el LED cada 0,5 segs?




```
#define LED 13

void setup() {

  pinMode( LED , OUTPUT );

  // Asocio la función toggleLed a la interrupción periódica
  // que ocurre cada cada 500 milisegundos

  attachPeriodicInterrupt( toggleLed, 500 );
}

// Función que se llamará en la interrupción
// No debe tener parámetros
void toggleLed() {
  int ledVal = digitalRead( LED );
  digitalWrite( LED, !ledVal );
}

void loop() {
  // Aquí se podrían hacer otras cosas, sin que
  // nos interfiera lo que se hace en la interrupción
  ...
  ...
}
```

Ejemplo: Comparando soluciones flujogramas

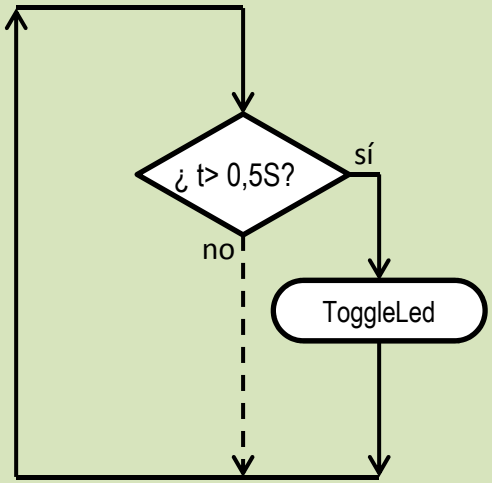
❑ **Bloqueo del proceso**:



❑ El microprocesador está bloqueado esperando que pase el tiempo.

❑ Es difícil dar cabida a otras tareas.

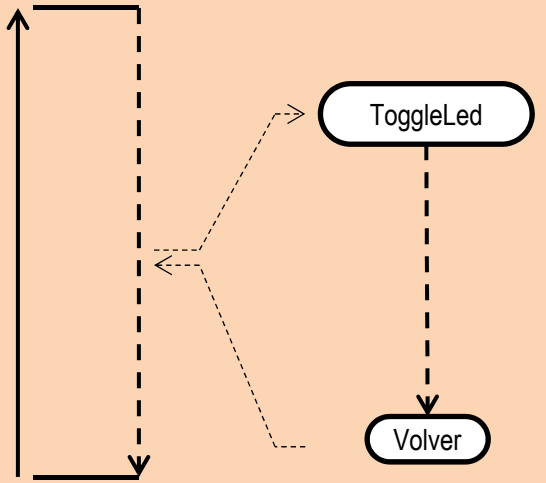
❑ **Consulta periódica**:



❑ El microprocesador comprueba cuánto tiempo ha pasado cada vez que pasa por ese punto del código.

❑ El resto del tiempo se puede aprovechar para realizar otras tareas.

❑ **Interrupciones**:



❑ Sólo cuando haya transcurrido el tiempo programado, el microprocesador atenderá la interrupción y cambiará el led.

Ejemplo: Comparando soluciones

- Utilizado la API de Arduino ¿Cómo resultaría el control del LED mediante los distintos procedimientos de entrada salida que conocemos (bloqueo, consulta periódica e interrupciones)?

```
// Solución por
//--------------------------
// BLOQUEO DEL PROCCESO
//--------------------------

#define LED 13

void setup() {
  pinMode( LED , OUTPUT );
}

void toggleLed() {
  int ledVal = digitalRead( LED );
  digitalWrite( LED, !ledVal );
}

void loop() {

  // la función delay espera
  // N ms, de forma bloqueante
  delay(500);
  toggleLed();

  // El código que hay aquí
  // solo se ejecutará cada
  // 0,5 segs
  ...
}
```

```
// Solución por
//--------------------------
// CONSULTA PERIÓDICA
//--------------------------

#define LED 13

void setup() {
  pinMode( LED , OUTPUT );
}

void toggleLed() {
  int ledVal = digitalRead( LED );
  digitalWrite( LED, !ledVal );
}

unsigned long nextTime = 500;

void loop() {

  // la función millis() devuelve
  // los ms pasados desde el reset
  if (millis() > nextTime) {
    nextTime = millis() + 500;
    toggleLed();
  }

  // Aquí se podrían hacer
  // otras cosas, siempre que no
  // tarden demasiado
  ...
}
```

```
// Solución mediante
//--------------------------
// INTERRUPCIONES
//--------------------------

#define LED 13

void setup() {
  pinMode( LED , OUTPUT );
  attachPeriodicInterrupt(
    toggleLed, 500 );
}

void toggleLed() {
  int ledVal = digitalRead( LED );
  digitalWrite( LED, !ledVal );
}

void loop() {

  // Aquí se podrían
  // hacer otras cosas,
  // sin que nos interfiera lo que
  // se hace en la interrupción
  // ni viceversa
  ...
}
```