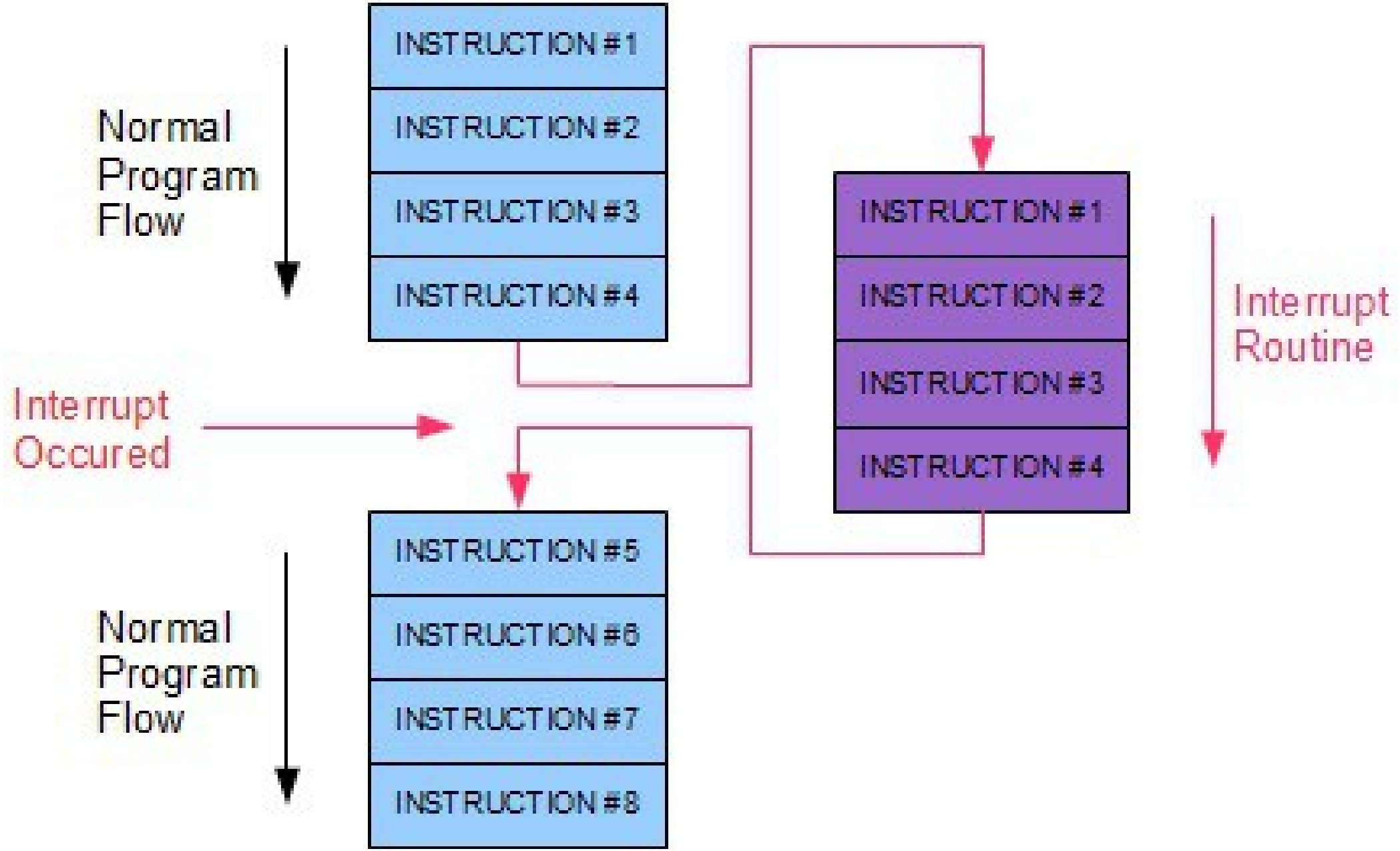
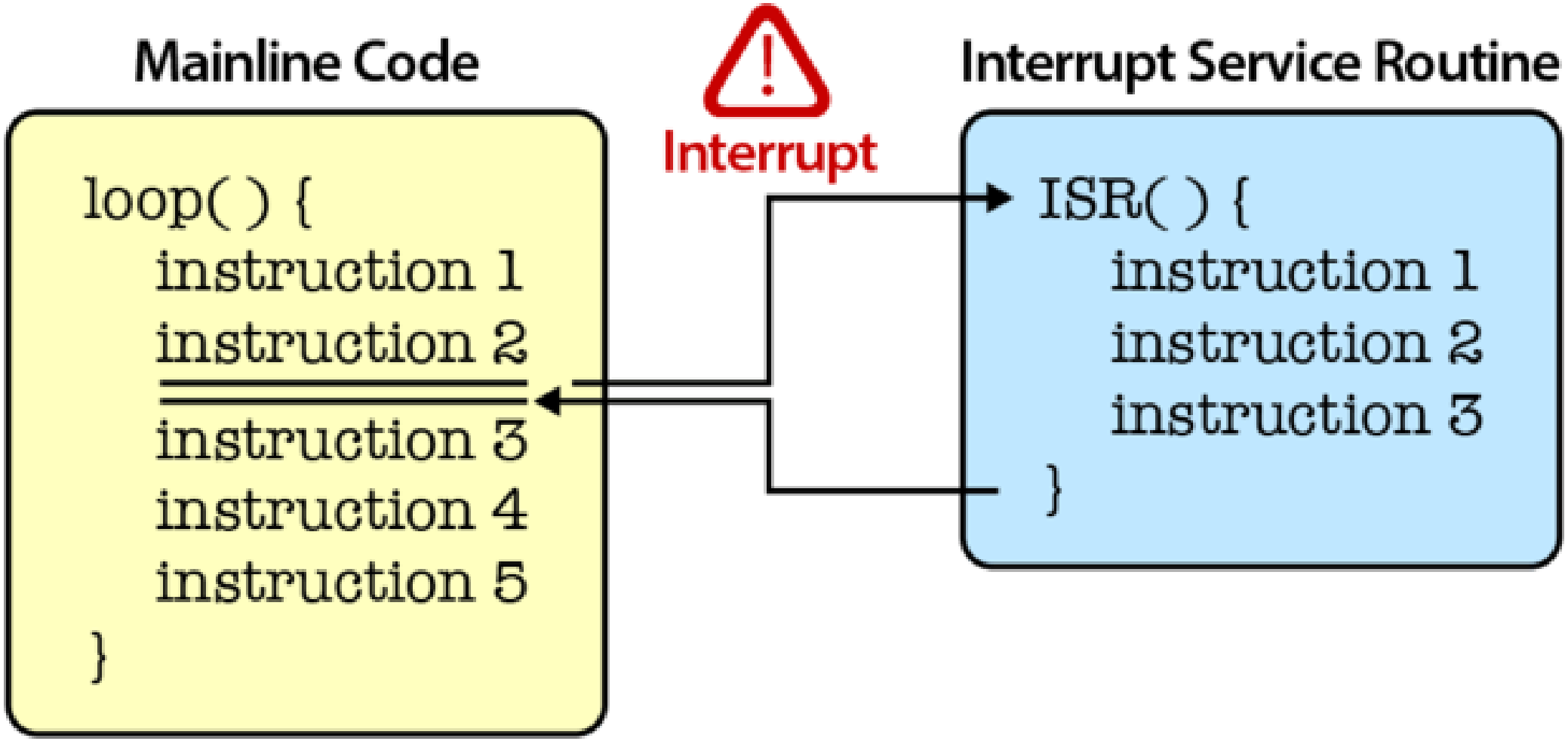
INTERRUPCIONES



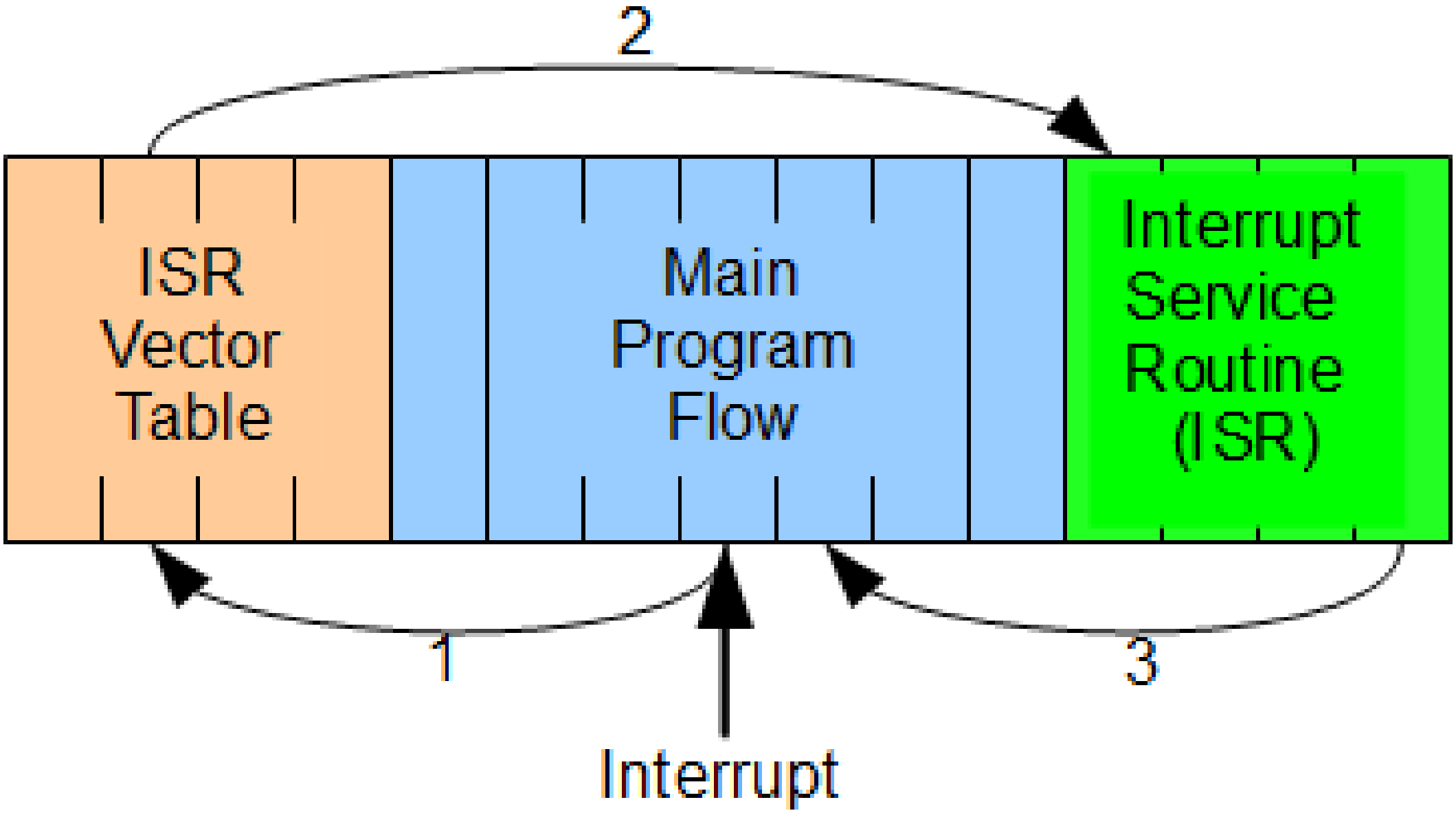
INTERRUPCIONES

Una interrupción es una señal recibida por el microprocesador para indicarle que debe interrumpir el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Una interrupción produce una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa. Una vez finalizada dicha subrutina, se reanuda la ejecución del programa.



https://aprendiendoarduino.wordpress.com/2016/11/13/interrupciones/


2
ISR
Vector
Table
Main
Program
Flow
Interrupt
Service
Routine
(ISR)
1
3
Interrupt

Diversas condiciones que pueden generar una interrupción:

- Cambio de estado en un pin, interrupción externa
- Poner en LOW el pin de RESET
- ADC, para indicar finalización de la conversión.
- Timers
- Controlador UART, para indicar que llegó información por el bus

Vectores de Interrupción en el ATmega328

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Interrupciones externas

interrupts() - habilita las interrupciones
noInterrupts() - desactiva las interrupciones
attachInterrupt() - programar interrupción
detachInterrupt() - desactivar interrupción

External Interrupts

- attachInterrupt()
- detachInterrupt()

Interrupts

- interrupts()
- noInterrupts()

noInterrupts()

Descripción
Desactiva las interrupciones (pueden reactivarse usando interrupts()).

Las interrupciones permiten que las operaciones importantes se realicen de forma transparente y están activadas por defecto.

Algunas funciones no funcionarán y los datos que se reciban serán ignorados mientras que las interrupciones estén desactivadas.

Las interrupciones pueden perturbar ligeramente el tiempo de temporizado, sin embargo puede que sea necesario desactivarlas para alguna parte crítica del código.

Parámetros Ninguno
Devuelve Nada

interrupts()

Descripción
Activa las interrupciones (después de haberlas desactivado con noInterrupts().

Las interrupciones permiten que se ejecuten ciertas tareas en segundo plano y están activadas por defecto.

Algunas funciones no funcionarán correctamente mientras las interrupciones estén desactivadas y la comunicación entrante puede ser ignorada.

Las interrupciones pueden perturbar ligeramente la temporización en el código y deben ser desactivadas sólo para partes particularmente críticas del código.

Parámetros Ninguno
Devuelve No devuelve nada

Ejemplo

```
void setup() {}

void loop()
{
  noInterrupts();   // código crítico y sensible al tiempo
  interrupts();     // otro código
}
```

attachInterrupt

attachInterrupt

Especifica la función a la que invocar cuando se produce una interrupción externa.

La placa Arduino UNO tienen dos interrupciones externas:

Interrupción número 0 (en el pin digital 2)
Interrupción número 1 (en el pin digital 3).

La Arduino Mega tiene otras cuatro: Las número 2 (pin 21), 3 (pin 20), 4 (pin 19) y 5 (pin 18).

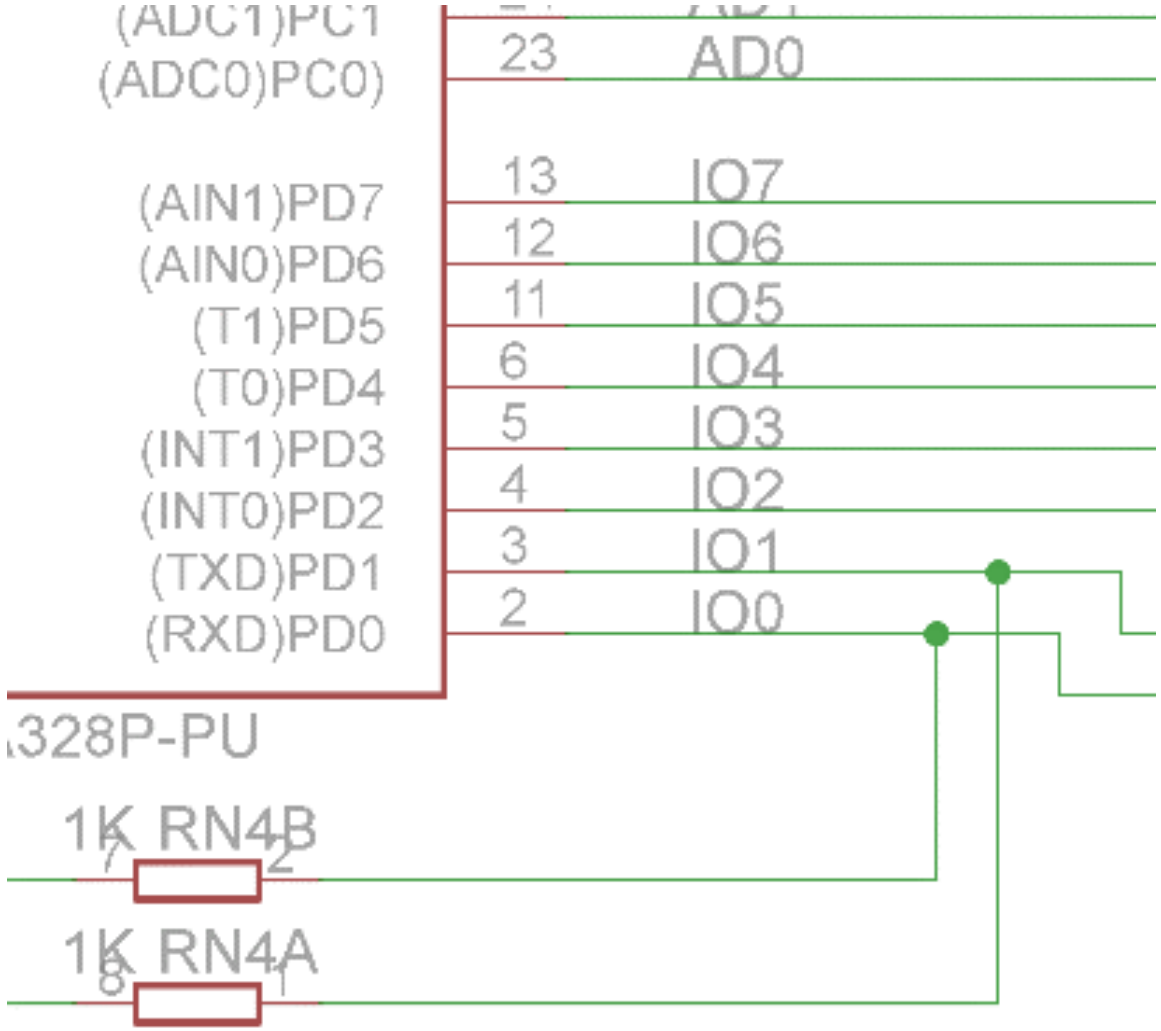
Syntax

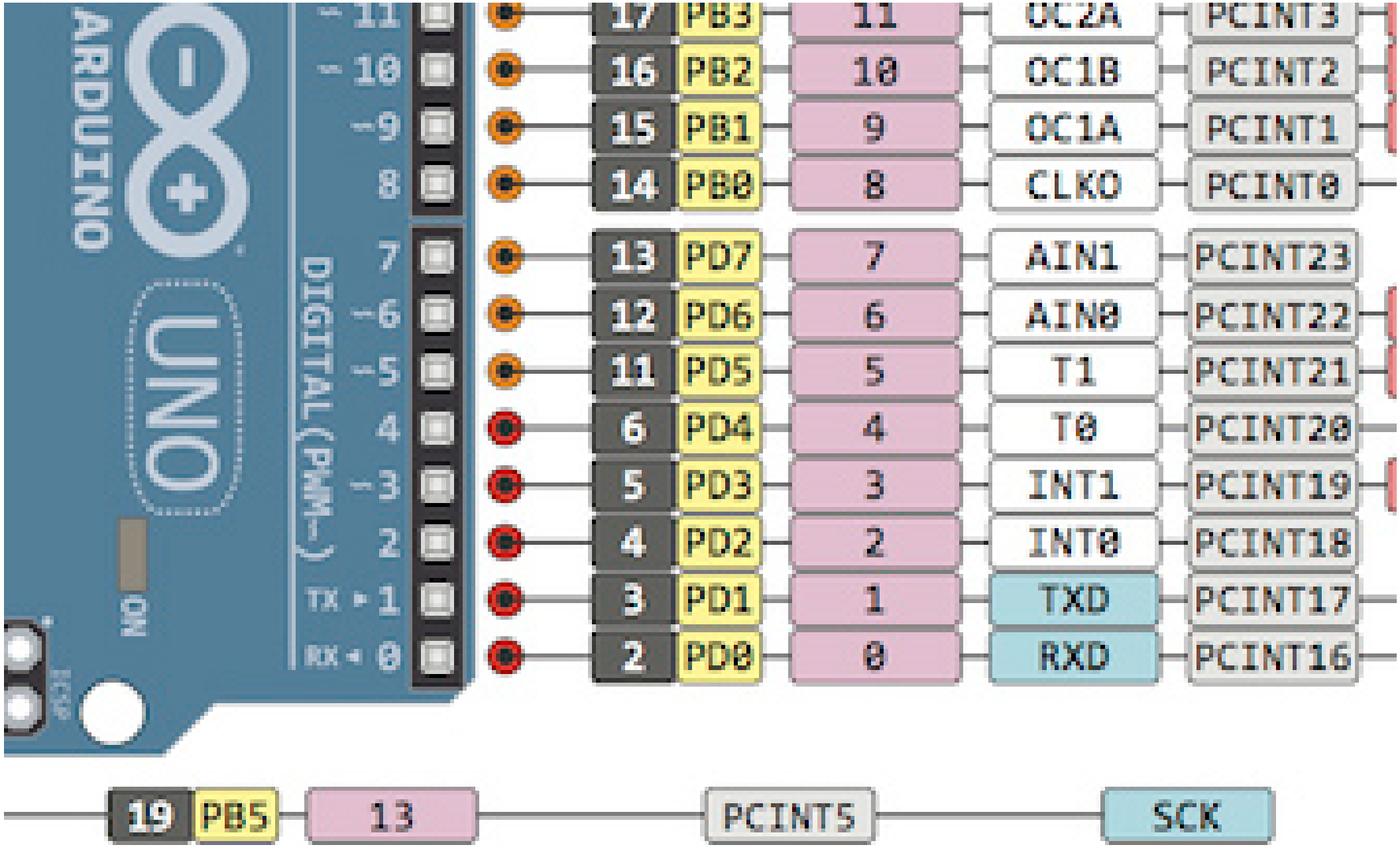
```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```
(*recommended*)

```
attachInterrupt(interrupt, ISR, mode);
```
(*not recommended*)

```
attachInterrupt(pin, ISR, mode) ;
```
(*not recommended Arduino Due, Zero, MKR1000 only*)

MODELO ARDUINO	INT 0	INT 1	INT 2	INT 3	INT 4	INT 5
UNO	Pin 2	Pin 3				
MEGA	2	3	21	20	19	18
DUE	Todos los pines del DUE pueden usarse para interrupciones.					
Leonardo	3	2	0	1	7	


(ADC1)PC1
(ADC0)PC0)
23
AD0
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
13
12
11
6
5
4
3
2
IO7
IO6
IO5
IO4
IO3
IO2
IO1
IO0
328P-PU
1K RN4B
1K RN4A


ARDUINO
UNO
ON
DIGITAL (PWM~)
~11
~10
~9
8
7
~6
~5
4
~3
2
TX → 1
RX ← 0
17 PB3 11 OC2A PCINT3
16 PB2 10 OC1B PCINT2
15 PB1 9 OC1A PCINT1
14 PB0 8 CLKO PCINT0
13 PD7 7 AIN1 PCINT23
12 PD6 6 AIN0 PCINT22
11 PD5 5 T1 PCINT21
6 PD4 4 T0 PCINT20
5 PD3 3 INT1 PCINT19
4 PD2 2 INT0 PCINT18
3 PD1 1 TXD PCINT17
2 PD0 0 RXD PCINT16
19 PB5 13 PCINT5 SCK

attachInterrupt(interrupcion, función, modo)

interrupción: el número de la interrupción (int)

función: la función a la que invocar cuando la interrupción tiene lugar; esta función **no debe tener parámetros ni devolver nada**.

Esta función es a veces referenciada como **rutina de interrupción de servicio**

Modo: define cuando la interrupción debe ser disparada.

- **LOW** para disparar la interrupción en cualquier momento que el pin se encuentre a valor bajo(LOW).
- **CHANGE** cuando el pin cambie de valor.
- **RISING** cuando el pin pase de valor alto (HIGH) a bajo (LOW).
- **FALLING** para cuando el pin cambie de (HIGH) a (LOW) .

1)

attachInterrupt(interrupcion, función, modo)

interrupción: el número de la interrupción (int) (0,1)

El primer parámetro para **attachInterrupt** es el número de interrupción. Normalmente debería utilizar digitalPinToInterrupt(pin) para traducir el pin digital real al número de interrupción específica.

Por ejemplo, si se conecta al pin3, utilice digitalPinToInterrupt(3) como el primer parámetro para **attachInterrupt**.

attachInterrupt(digitalPinToInterrupt(pin), función, modo)

2)

attachInterrupt(interrupcion, función, modo)

función: la función a la que invocar cuando la interrupción tiene lugar; esta función **no debe tener parámetros ni devolver nada**.

Esta función es a veces referenciada como **rutina de interrupción de servicio ISR**

interrupt service routine ISR

La función que se ejecuta cuando se dispara la interrupción se llama rutina de servicio de interrupción o en inglés interrupt service routine.

Las ISR deben tan cortas como sea posible.

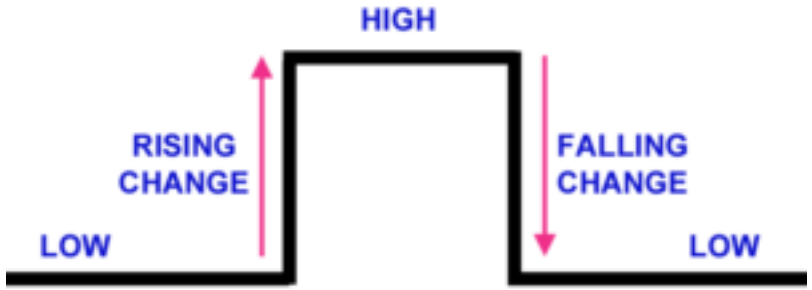
Esta rutina no admite ni parámetros de entrada ni devuelve nada

Por tanto debe ser del tipo **void nombrefuncion ()** como lo son por ejemplo las funciones “setup” y “loop” que tampoco tienen parámetros ni devuelven nada.

3) attachInterrupt(interrupcion, función, modo)

Modo: define cuando la interrupción debe ser disparada.

LOW para disparar la interrupción en cualquier momento que el pin se encuentre a valor bajo(LOW).
CHANGE cuando el pin cambie de valor.
RISING cuando el pin pase de valor alto (HIGH) a bajo (LOW).
FALLING para cuando el pin cambie de (HIGH) a (LOW) .



Retorno

Ninguno

Nota

- *Dentro de la función enlazada, la función delay() no funciona y el valor devuelto por la función millis() no se incrementará.*

- *Los datos serie recibidos en el transcurso de esta interrupción pueden perderse.*

- *Deberías declarar como volátil cualquier variable que modifiques dentro de la función.*

1)

Si el programa usa varias ISRs, sólo una puede ejecutarse a la vez, las otras interrupciones se ejecutarán después que finalice la actual en orden dependiendo de la prioridad que tengan.

Las funciones de tiempo delay(), micros(), milis() requieren una interrupción para trabajar, y por tanto no trabajaran dentro de una ISR.

delayMicroseconds() trabajará normalmente

2)

También puede ocurrir y hay que tener en cuenta, que si se envían datos serie desde el exterior, por ejemplo del PC a Arduino, cuando se encuentra “ocupado” con una interrupción, pueden perderse, pues está volcado en hacer “otra cosa”, lo que mande hacer la interrupción.

3)

Normalmente se usan variables globales para pasar datos entre una ISR y el programa principal.

Para asegurarse que son correctamente actualizadas hay que declararlas como “volatile”.

Ejemplo 2 – Un Led cambia de estado cada vez que pulsamos un boton

```
const byte ledPin = 13;          //declaramos una constante con el número de pin
const byte interruptPin = 2;     //declaramos una constante con el pin de la interrupci
volatile byte estadop = LOW;     //Declaramos un variable global tipo byte "volatile"

void setup() {
  pinMode(ledPin, OUTPUT);     //configuramos el pin como salida
  pinMode(interruptPin, INPUT_PULLUP);  //configuramos el pin 2 como entrada
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);  //configura la
}

void loop() {
  digitalWrite(ledPin, estadop);  //se enciende el led de acuerdo al valor de la varia
}

void blink() {       //se declara la función ISR
 estadop = !estadop;  //instrucción a ejecutar en la ISR
}
```

https://www.arduinoparatodos.com/las-interrupciones-arduino/

detachInterrupt

detachInterrupt(interrupt)

Descripción

Apaga la interrupción dada.

Parámetros

interrupt: el número de interrupción a invalidar (0 o 1).

Ámbito de las variables y cualificadores

Ámbito de las variables

static - estático

volatile - volátil

const - constante

Variables volátiles

Cuando declaramos una variable como “*volatile*” lo que estamos haciendo es instruyendo al compilador a cargar siempre la variable desde la memoria RAM y no desde uno de los registros internos del procesador.

La razón por la que hacemos esto es que, bajo ciertas condiciones, el valor de una variable almacenada en los registros internos puede sufrir alteraciones. En Arduino el único sitio en el que esto puede suceder es en las partes de código asociado con interrupciones (*interrup service routines*).

Para una variable normal, declarada sin volatile, el compilador se toma la libertad de optimizar el código, y dejar una copia de la variable en algún registro de la CPU mientras se está utilizando, para ganar velocidad.

Si se produce una interrupción en ese momento, y dentro de la ISR se utiliza esa variable, se estará leyendo un valor distinto; y si se hacen cambios a la variable, al regresar al programa principal y restaurar el contexto, los cambios se perderán.

Volatile previene que ocurra esto, impidiendo al compilador optimizar, y forzando que la variable se utilice siempre desde la RAM.

Se deben declarar como “volatile” cualquier variable que sea modificada dentro de la función llamada por una interrupción

Static

La palabra reservada static se utiliza para crear variables estáticas, estas variables tienen la peculiaridad de que no se crean y se destruyen cada vez que se llama al bloque de código en el que está definidas, sino que su valor se guarda para las sucesivas llamadas.

Las variables que se declaran como estáticas sólo se crearan e inicializarán la primera vez que se ejecute el bloque de código en el que están contenidas.

La palabra reservada (keyword) “*static*” se usa para crear variables que son visibles únicamente para una funcion.

La diferencia con las variables locales es que éstas son creadas y destruidas cada vez que se llama a una función.

Las variables definidas como “*static*” persisten cuando la función ha terminado y conservan los datos almacenados en ellas disponibles para la próxima vez que se llame a esa función. Estas variables se crean e inicializan solamente la primera vez que la función que las crea es llamada.

```
int randomWalk(int moveSize){
  static int  place;     // variable estática  los valores se conservaron entre las siguientes
                         // llamadas a la función. Ninguna otra función puede cambiar su valor
```

Interrupciones pin change (PCINT)

cuyo modo de funcionamiento es similar, pero actúan en número muy superior de pines del procesador.

las PCINT también tienen algunas desventajas respecto a las habituales INT.

En primer lugar, a diferencia de las interrupciones INT que actúan sobre un único pin, las PCINT **actúan sobre un grupo de pines** de forma simultánea (normalmente sobre un puerto). Cuando se produzca un cambio en cualquier pin del grupo actúa la interrupción, para saber el pin sobre el que se ha actuado se debe hacer una consulta posterior de un registro.

En segundo lugar, a diferencia de las interrupciones INT que permiten configurar el disparo CHANGE, FALLING, RISING, LOW y HIGH, las interrupciones INT **únicamente distinguen eventos de CHANGE**. Si queremos detectar flancos de subida o de bajada deberemos guardar el estado del registro en una variable y realizar la comparación con el estado anterior en la ISR.

Finalmente, por los motivos anteriores, **son ligeramente más lentas** que las interrupciones INT.

Simple Pin Change Interrupt on all pins

It is possible to use pin change interrupts on "all" pins of the arduino using Pin Change Interrupt Requests. The example below uses some macros from the pins_arduino.h library.

The interrupt can be enabled for each pin individually (analog and digital!), but there are only 3 interrupt vectors, so 6-8 pins share one service routine:

- ISR (PCINT0_vect) pin change interrupt for D8 to D13
- ISR (PCINT1_vect) pin change interrupt for A0 to A5
- ISR (PCINT2_vect) pin change interrupt for D0 to D7

PinChangeInt Example

This code counts the number of times pin 15 (aka Analog 1) changes state and prints the count when it recives a p on the serial port.

```
1. /*
2. Copyright 2011 Lex.V.Talionis at gmail
3. This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
4. */
```