CIRCUITOS DIGITALES Y MICROCONTROLADORES 2022

Facultad de Ingeniería UNLP

Interrupciones
concepto y programación

Ing. José Juárez

Abril 2021

Introducción

- La CPU de un microcontrolador ejecuta instrucciones secuencialmente.
- Sin embargo, las aplicaciones requieren del uso de diferentes periféricos (internos o externos) y por lo tanto la CPU debe contar con un mecanismo para interactuar con ellos y dar respuesta adecuada a sus demandas.
- Los periféricos generalmente requieren la atención de la CPU de manera aleatoria en respuesta a algún evento.

  Por ej: "se recibió un byte desde la interfaz serie y requiere ser almacenado"

1) Una manera de interactuar con un periférico es utilizando el método de encuesta o polling:

Ejemplo:

- Esperar se presione una tecla (caso visto en Tp1 y Tp2)

Otro ejemplo:

- Esperar recepción de un byte desde la PC (utilizando interfaz serie)

- En ambos ejemplos, la CPU consumirá ciclos de reloj "en espera" a que se produzca el evento determinado sin realizar otra tarea.
- Existen técnicas para hacer uso eficiente del polling, por ejemplo las MEF.

Introducción

2) Otra manera de interactuar con un periférico es a través del mecanismo de INTERRUPCION

- Si un periférico requiere la atención de la CPU (para ejecutar un determinado código) éste envía un pedido de interrupción a la CPU a través de una señal de hardware (con un nivel lógico).
- Una INTERRUPCION es entonces un MECANISMO DE HARDWARE que permite suspender temporalmente la ejecución normal de un programa para brindar servicio a la solicitud requerida por un periférico.
- Si la CPU acepta la interrupción, suspende la ejecución del programa en curso y pone en ejecución una rutina de código que interacciona con dicho periférico. Esta rutina especial se denomina *"Rutina de Servicio de Interrupción"* o *"Manejador de interrupción"* y se ejecuta a través de la administración *del controlador de interrupciones*.
- Luego se debe reanudar la ejecución normal del programa en el punto donde fue suspendido.

Tipos de Interrupciones

- Las interrupciones internas

  son las producidas por los periféricos integrados en el mismo chip

  - por ejemplo: TIMER, SPI, UART, A/D, RESET

- Las interrupciones externas

  pueden ser producidas por dispositivos conectados al MCU al los terminales externos IRQ o a los puertos E/S que tengan capacidad de producir interrupciones.

  - por ejemplo: otro microcontrolador, un codec de audio, un teclado táctil, un encoder, entre otros.

Otros Tipos de Interrupciones

- Interrupciones especiales o Excepciones:
  - Son anomalías que ocurren durante la ejecución de instrucciones en la CPU
  - Por ejemplo, error de bus interno, error de acceso a memoria, división x cero, instrucción ilegal (opcode no válido)
  - Los microcontroladores AVR NO poseen excepciones.

- Interrupción por software:
  - En algunos microcontroladores existe una *instrucción de assembler* que permite generar un interrupción al ejecutarla SWI (Software Interrupt)
  - Se usa por ejemplo para forzar un cambio de modo o de contexto dentro de un sistema operativo multitarea.
  - Los microcontroladores AVR NO poseen este tipo de instrucción pero pueden implementarse interrupción x software (con terminales I/O)

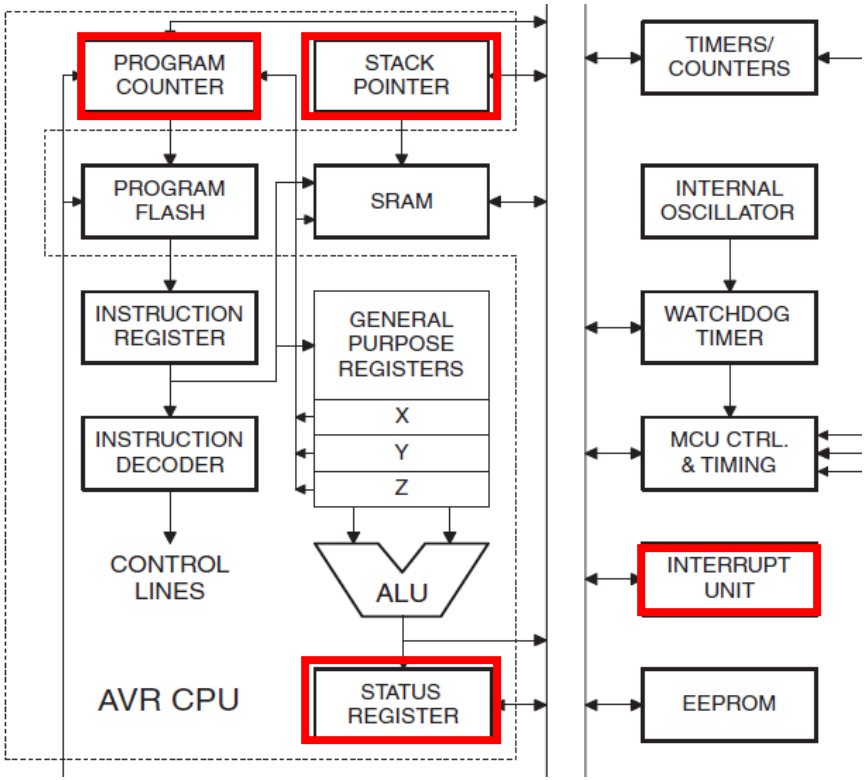
Interrupciones

- La RSI es similar a una subrutina de assembler (o función en C) excepto que las interrupciones son ejecutadas por el mismo hardware mientras que las subrutinas o funciones son planificadas por el programador y ejecutadas por medio de una instrucción de assembler del tipo “CALL”
- Una subrutina de assembler finaliza cuando se ejecuta una instrucción de retorno “RET”, mientras que una RSI finaliza cuando se ejecuta una instrucción de retorno de interrupción “RETI”
- ya veremos el porqué…

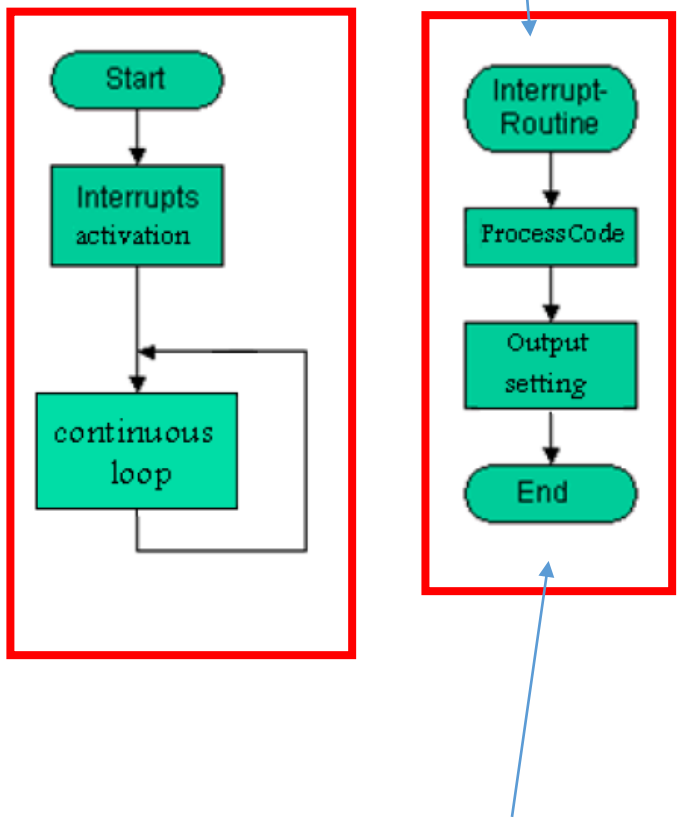
Interrupciones AVR

las interrupciones son puestas en ejecución mediante el controlador de interrupciones

- Hardware



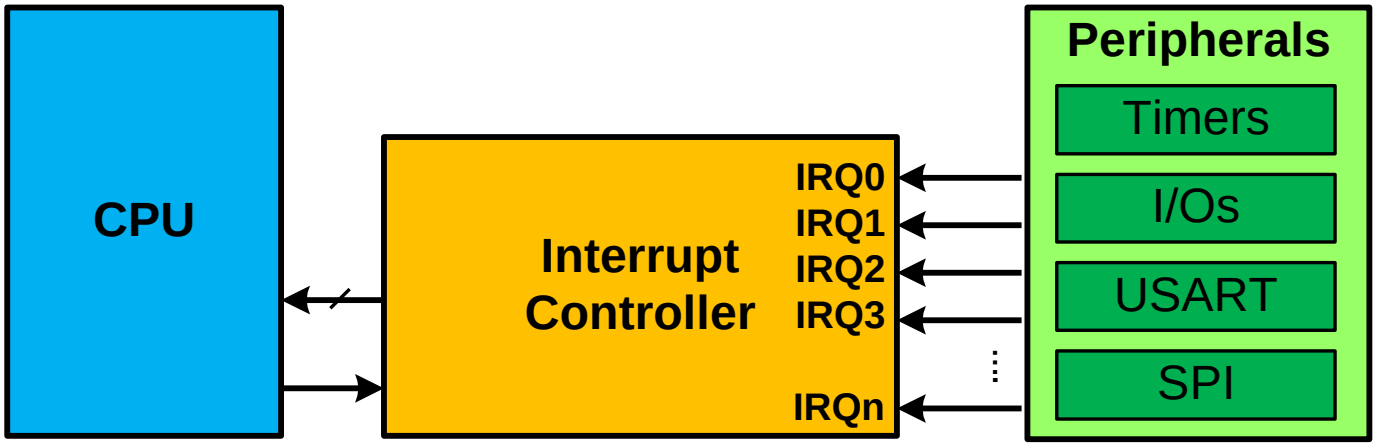
Software



Una RSI finaliza cuando se ejecuta una instrucción de retorno de interrupción "RETI"

Interrupt Unit

Cada fuente de interrupción tiene asociada un lugar en memoria de programa donde se encuentra la dirección o la referencia a la RSI a ejecutar (llamado Vector de Interrupción)



Vectores de interrupción

- El grupo de localidades de memoria destinadas a guardar las direcciones de las RSI, se llama "Tabla de Vectores de Interrupción"
- Para cada fuente de interrupción distinta debe existir una sola RSI asociada que pueda ejecutarse.
- El programador diseña la RSI que desea se ejecute en cada caso como si fuese una función especial.
- El fabricante especifica donde disponer de esta tabla, el orden de ubicación dentro de la misma y la prioridad de atención, en caso que se den varios pedidos de interrupción simultáneamente.
- En la mayoría de los uC la tabla está al principio de la memoria de programa FLASH o al final.
- El mecanismo de vector permite distinguir rápidamente entre múltiples pedidos de interrupción y determinar su origen para ejecutar a la RSI que corresponda.
- El fabricante reserva direcciones de memoria especificas (llamadas vector) para cada interrupción con una determinada Prioridad dada por el orden que aparecen en la tabla.

Tabla de Vectores- AVR (ver hoja de datos)

alta prioridad

Baja prioridad

Table 11-6. Reset and Interrupt Vectors in ATmega328P

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
…	…	…	…

- Atmega328 tiene 26 interrupciones distintas

Implementación de los Vectores de interrupción en AVR

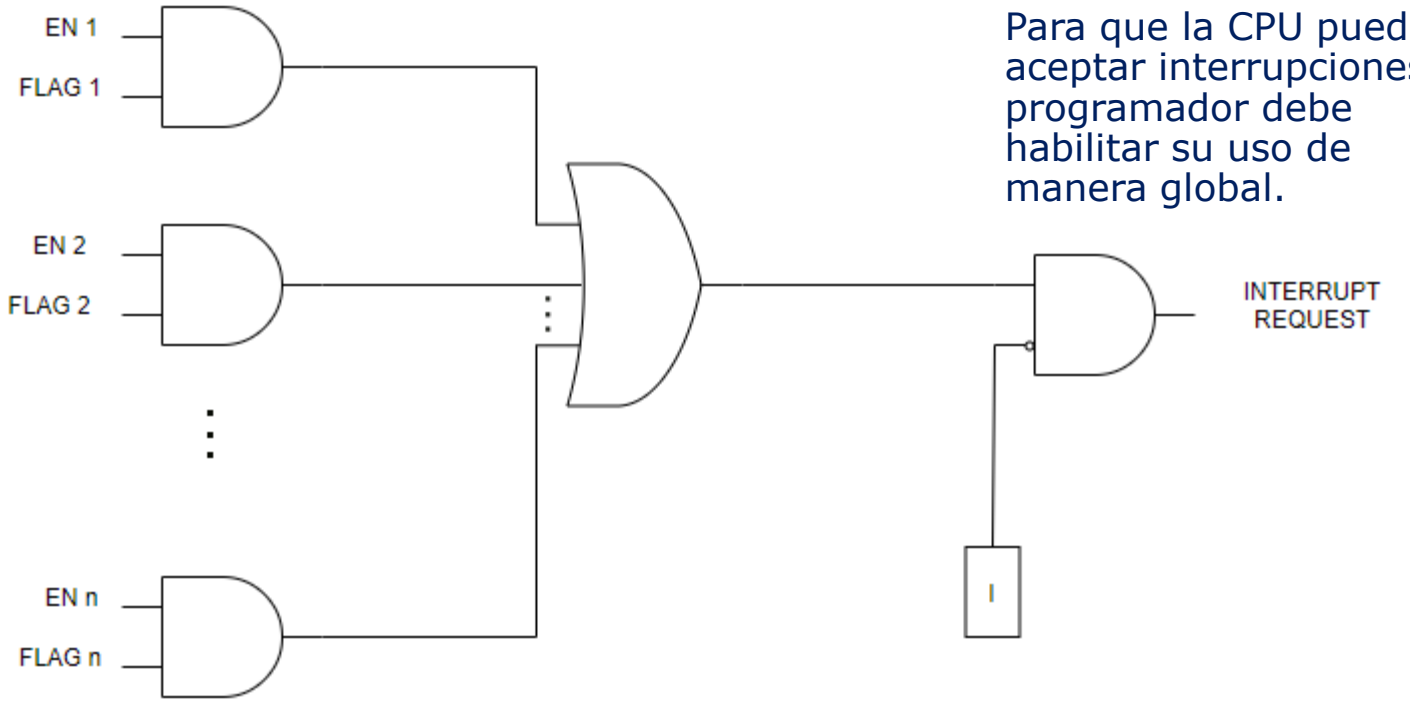
Instrucción de assembler que carga el PC con la dirección de la rutina de interrupción correspondiente

```
Address  Labels Code                    Comme
0x0000          jmp    RESET           ; Res
0x0002          jmp    EXT_INT0        ; IRQ
0x0004          jmp    EXT_INT1        ; IRQ1 Handler
0x0006          jmp    PCINT0          ; PCINT0 Handler
0x0008          jmp    PCINT1          ; PCINT1 Handler
0x000A          jmp    PCINT2          ; PCINT2 Handler
0x000C          jmp    WDT             ; Watchdog Timer Handler
0x000E          jmp    TIM2_COMPA      ; Timer2 Compare A Handler
0x0010          jmp    TIM2_COMPB      ; Timer2 Compare B Handler
0x0012          jmp    TIM2_OVF        ; Timer2 Overflow Handler
0x0014          jmp    TIM1_CAPT       ; Timer1 Capture Handler
0x0016          jmp    TIM1_COMPA      ; Timer1 Compare A Handler
0x0018          jmp    TIM1_COMPB      ; Timer1 Compare B Handler
0x001A          jmp    TIM1_OVF        ; Timer1 Overflow Handler
```

Habilitando las Interrupciones

- Cada fuente de interrupción se puede habilitar o deshabilitar independientemente (excepto el RESET).
- Luego de un RESET, todas las interrupciones están deshabilitadas (generalmente se dice: enmascaradas)

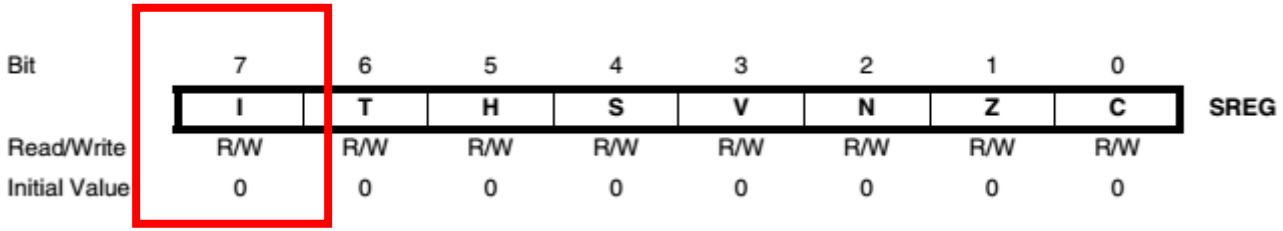
Cada periférico posee bits específicos para habilitar o deshabilitar la generación de interrupciones.



Para que la CPU pueda aceptar interrupciones el programador debe habilitar su uso de manera global.

Habilitando las Interrupciones

- El registro de Estado SREG, contiene el bit I para tal fin:



- El bit I: se llama mascara global de interrupciones y debe estar en “1” para permitir que las mismas sean atendidas por la CPU
- En assembler se utiliza la instrucción SEI para habilitar las interrupciones y la instrucción CLI para deshabilitarlas.
- En C se utilizan los macros cli() y sei()

Habilitando las Interrupciones

- La programación de los periféricos y la posibilidad de que generen interrupciones es responsabilidad del programador (en las siguientes clases veremos como configurar los distintos periféricos para que generen interrupciones).
- Para que una interrupción generada por un periférico sea aceptada, la máscara global I debe estar en 1.
- Si I=0, la CPU no acepta interrupción y la interrupción “queda pendiente a ser atendida” hasta que I=1
- A si mismo, si la CPU acepta la interrupción se debe “informar ” al periférico que fue atendida mediante alguna forma de reconocimiento de interrupción
- Cada periférico posee un *flag de estado* de interrupción y este debe ser borrado para poder procesar nuevas solicitudes.
- Esta tarea se realiza al principio de la RSI para permitir atender un nuevo pedido del mismo periférico tan pronto termine el pedido en curso (en algunos casos este reconocimiento es automático pero en otros debe hacerlo el programador).

Pasos de ejecución de una interrupción

1. Cuando un pedido de interrupción es reconocido y aceptado, la CPU finaliza la instrucción que estaba ejecutando y almacena el contador de programa PC (que apunta a la siguiente instrucción) en la PILA (indicada por SP – stack pointer).
2. Se deshabilita la máscara global I (I=0).
3. Se busca en la “Tabla de Vectores de Interrupción” la posición de memoria asignada a esa interrupción y se ejecuta el salto (cambio del PC) a la RSI que corresponda ejecutar.
4. LA CPU ejecuta la RSI hasta encontrar la instrucción de retorno RETI.
5. Al ejecutar RETI, el PC original que estaba en PILA es restaurado para poder continuar con la ejecución normal del programa que se estaba ejecutando antes de la interrupción y se habilita nuevamente la máscara global I.

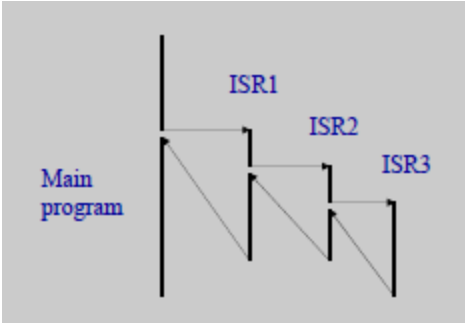
Mecanismo de Prioridad

- La prioridad de una interrupción se utiliza para decidir cual interrupción debe ser atendida primero, si se diera el caso de que hay múltiples solicitudes de interrupciones al mismo tiempo.
- La prioridad esta determinada por el orden que aparecen los vectores en la tabla. Así las primeras son las más prioritarias y las últimas las menos prioritarias.
- Ejemplo: durante la ejecución de una determinada RSI se deshabilita automáticamente la mascara I (I=0). Al finalizar y ejecutar RETI se vuelve a habilitar (I=1) y se encuentra que dos periféricos han pedido servicio de interrupción: INT0 (interrupción externa 0) y UART Rx (recepción serie). Entonces el controlador de interrupciones decide que por prioridad la solicitud de INT0 será atendida primero y a continuación la de UART Rx.

- Nota: al menos una instrucción del programa principal será ejecutada antes de atender el próximo pedido de interrupción (si hubiera pendiente).

Interrupciones anidadas

- Una interrupción NO puede interrumpir a otra interrupción que se este procesando.
- Sin embargo, si el programador lo desea, puede volver a habilitar la mascara I (I=1) dentro de una RSI para permitir este comportamiento.
- Esto se denomina anidamiento de interrupciones.
- Una interrupcion en curso solo puede ser interrumpida por otra interrupción de mayor prioridad



- No es Recomendable el anidamiento de interrupciones.
- Es casi imposible crear código que se ajuste bien para todas las posibles combinaciones de interrupciones, aun siendo priorizadas.
- Se reduce la posibilidad de predecir el comportamiento del sistema en todas las posibles condiciones, es decir se pierde confiabilidad ya que no todas las condiciones son testeadas.

Latencia de una Interrupción

- Es el tiempo que tarda el Controlador de interrupciones en dar respuesta a una interrupción.

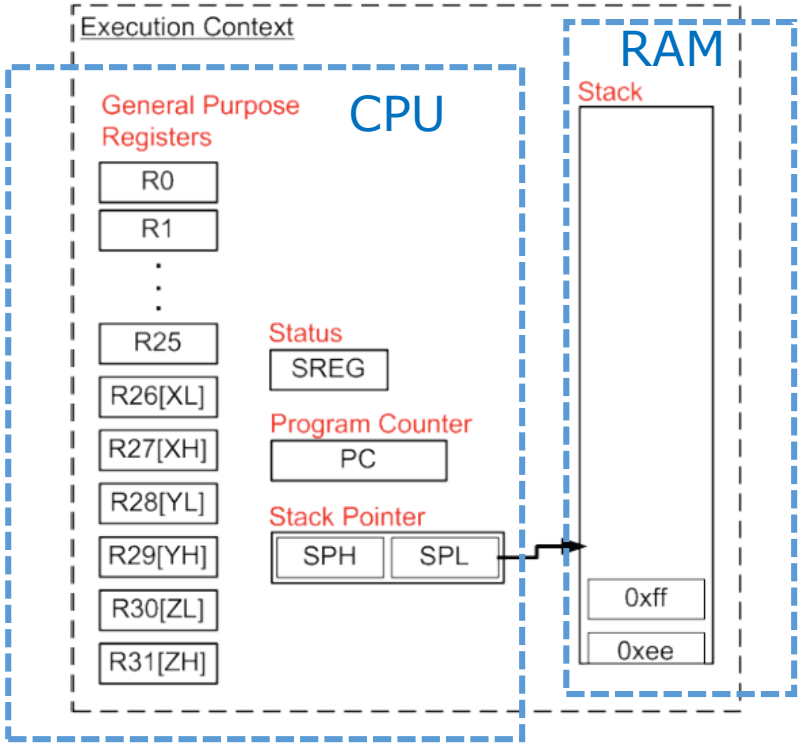
- Se mide desde que se recibe el pedido hasta que efectivamente se ejecuta la primer instrucción de la RSI correspondiente.

- En los AVR la latencia es de 4 ciclos de reloj como mínimo.
  - Durante este tiempo, se guarda el PC en la pila, se pone I en 0 y se busca el vector de interrupción de mayor prioridad que corresponda.
- El retorno de la interrupción (RETI) tambien lleva 4 ciclos.

- En el caso en que el micro este en modo SLEEP, la latencia es de 8 ciclos.

Guardando el contexto (uso del stack)

- Dado que las interrupciones son asincrónicas al programa principal, los registro CPU y el registro de estado SREG deben guardarse (respaldo) al comienzo de la RSI, para que la ejecución normal del programa pueda continuar sin problemas después de atender la interrupción.
- El estado actual de los registros se denomina “contexto”
- La pila (stack) de programa, es el mecanismo que permite “guardar el contexto” de ejecución y volver a restablecerlo cuando la interrupción haya terminado.



Interrupciones y el compilador AVR-GCC

- Para utilizar interrupciones en un programa debemos agradar el archivo de cabecera de Interrupciones:
  - #include<avr\interrupt.h>

- El código que queremos ejecutar en respuesta a una interrupción será RSI o ISR (Interrupt service rutine en inglés):

```
ISR (interrupt vector name)
{
  //nuestro código

}
```

  - Notar que es una función especial que recibe como parámetro el nombre del vector de interrupción y no devuelve ningún argumento de salida.
- Para habilitar o deshabilitar la máscara global se utilizan los Macros cli(), sei():
  - Son implementaciones equivalentes de las instrucciones de assembler

```
#define  cli()  __asm__ __volatile__ ("cli" ::)
#define  sei()  __asm__ __volatile__ ("sei" ::)
```

Interrupciones y el compilador AVR-GCC

- **ISR(Vector Name):** **definidos en io.h**


```
#define INT0_vect          _VECTOR(1)   /* External Interrupt Request 0 */
#define INT1_vect          _VECTOR(2)   /* External Interrupt Request 1 */
#define PCINT0_vect        _VECTOR(3)   /* Pin Change Interrupt Request 0 */
#define PCINT1_vect        _VECTOR(4)   /* Pin Change Interrupt Request 0 */
#define PCINT2_vect        _VECTOR(5)   /* Pin Change Interrupt Request 1 */
#define WDT_vect           _VECTOR(6)   /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect  _VECTOR(7)   /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect  _VECTOR(8)   /* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect    _VECTOR(9)   /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect   _VECTOR(10)  /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect  _VECTOR(11)  /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect  _VECTOR(12)  /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect    _VECTOR(13)  /* Timer/Counter1 Overflow */
```

Interrupciones y el compilador AVR-GCC

- **ISR(Vector Name):** **definidos en io.h**

```
#define TIMER0_COMPA_vect _VECTOR(14)  /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect _VECTOR(15)  /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect   _VECTOR(16)  /* Timer/Couner0 Overflow */
#define SPI_STC_vect      _VECTOR(17)  /* SPI Serial Transfer Complete */
#define USART_RX_vect     _VECTOR(18)  /* USART Rx Complete */
#define USART_UDRE_vect   _VECTOR(19)  /* USART, Data Register Empty */
#define USART_TX_vect     _VECTOR(20)  /* USART Tx Complete */
#define ADC_vect          _VECTOR(21)  /* ADC Conversion Complete */
#define EE_READY_vect     _VECTOR(22)  /* EEPROM Ready */
#define ANALOG_COMP_vect  _VECTOR(23)  /* Analog Comparator */
#define TWI_vect          _VECTOR(24)  /* Two-wire Serial Interface */
#define SPM_READY_vect    _VECTOR(25)  /* Store Program Memory Read */
```

Interrupciones y el compilador AVR-GCC

```
#include <avr/io.h>
#include <avr/interrupt.h>

int main (void) {
   DDRB |= (1<<DDRB0);
   TIMSK=(1<<TOIE0);
   sei();
   while(1)   { ; }

return 0;
}

ISR (TIMER0_OVF_vect)
{
  PORTB |= (1<<PORTB0);
}
```

Ejemplo de programa usando interrupciones

Assembler de la ISR:

```
ISR (TIMER0_OVF_vect)
{
        push    r0              ;r0->(SP)
        in      r0, SREG
        push    r0              ;SREG->(SP)

 PORTA |= (1<<PA0);
        sbi     PORTA, 0

        pop     r0
        out     SREG, r0        ;(SP)->SREG
        pop     r0              ;(SP)->r0
        reti
}
```

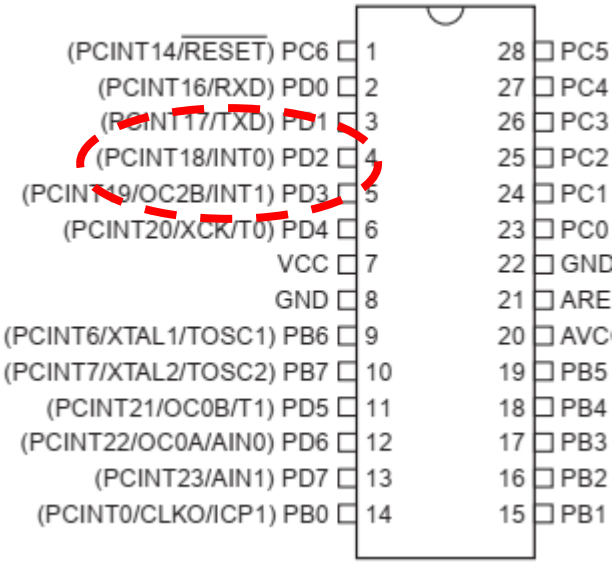
Interrupciones y el compilador AVR-GCC

- Importante:
  - El compilador, automáticamente agrega instrucciones al principio y al final de cada RSI para guardar y restablecer el contexto de TODOS los registros CPU y del SREG en la pila.
  - No es posible pasar parámetros a una RSI. Se deben utilizar variables globales, especificadas como *volatile* para compartir información con otra parte del programa.
  - *Volatile* indica al compilador que no es posible saber cuando la variable puede cambiar su valor (*Ejemplo: los puertos I/O o los flags de estado de los periféricos)* y por lo tanto no debe aplicar optimizaciones sobre la misma.
  - Ejemplo del porqué: pág 126 en Libro de Espinosa.

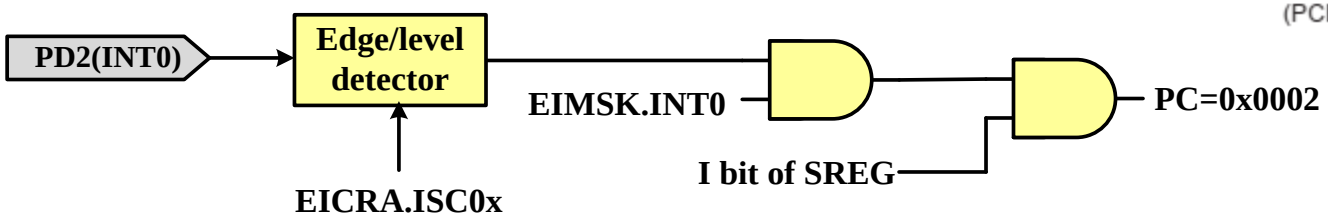
Las Interrupciones Externas

- Hay dos terminales INT0(PD2), INT1(PD3) en el ATMEGA328p que pueden generar interrupciones de periféricos externos
- Se habilitan con el registro **EIMSK – External Interrupt Mask Register**

7	6	5	4	3	2	1	0
–	–	–	–	–	–	INT1	INT0
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0



INT0 =1 Enable Int
INT0 =0 Disable Int



INT0 o INT1 son "activas en nivel bajo" por defecto.

Tabla de Vectores- ATMEGA

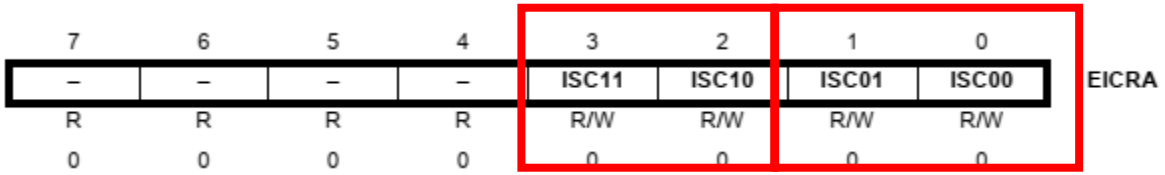
- 26 interrupciones distintas

Table 11-6. Reset and Interrupt Vectors in ATmega328P

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

Las Interrupciones Externas

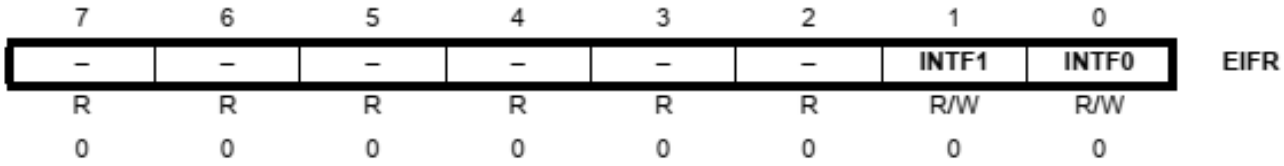
- El tipo de activación de las interrupciones INT0 e INT1 se puede configurar en el registro **EICRA**: External Interrupt Control Register



ISCx1	ISCx0	
0	0	
0	1	
1	0	
1	1	

Las Interrupciones Externas

- Las interrupciones son registradas en un Flag y el mecanismo de reconocimiento es automático



- Las bandera se limpia automáticamente por hardware cuando se concluye con la atención a la interrupción..

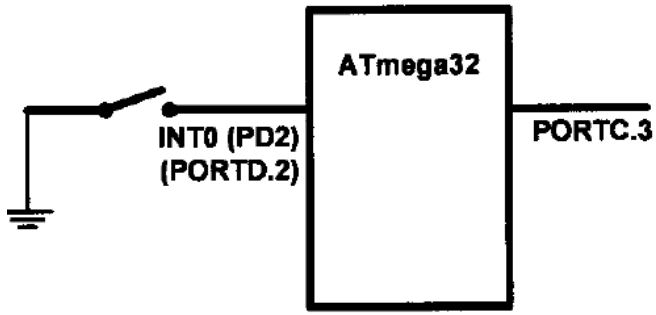
Tarea para la casa:

Copiar y ejecutar los ejemplos en el simulador.
Modificar el programa del TP1 para cambiar la secuencia de encendido de LEDS utilizando un pulsador conectado a una entrada de interrupción externa.

Las Interrupciones Externas

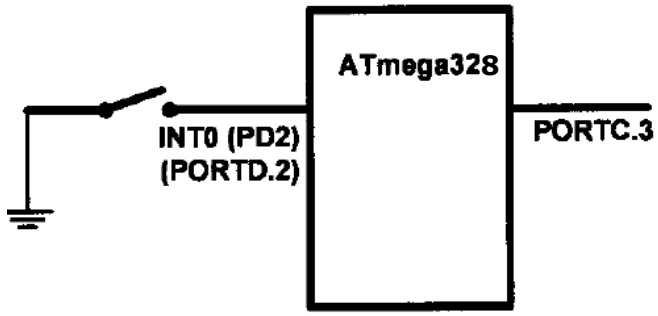
- Ejemplo: Un nivel bajo en INT0 interrumpe para negar PORTC bit 3

```
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    DDRC |=(1<<DDC3);
    PORTD |=(1<<PORTD2);
    EIMSK |=(1<<INT0);  //Habilitar int INT0
    sei();              //Habilitar int globales
    while (1) ;
}

ISR(INT0_vect){
    PORTC ^=(1<<PORTC3);
}
```



Las Interrupciones Externas

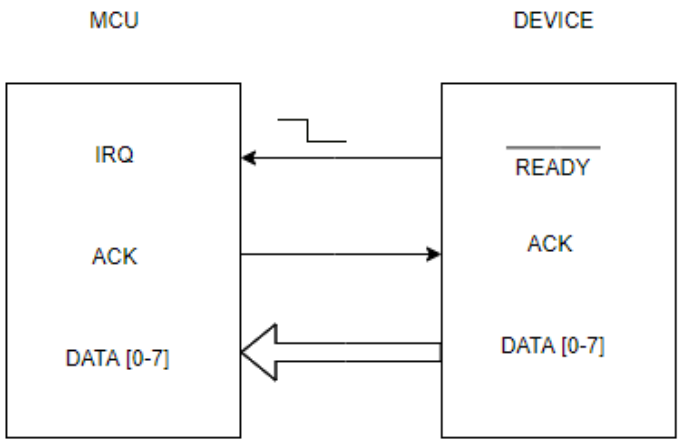
- Ejemplo: Flanco de bajada en INT0 interrumpe para negar PORTC bit 3




```
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    DDRC |=(1<<DDC3);
    PORTD |=(1<<PORTD2);
    EICRA |=(1<<ISC01); //INT0 x flanco de bajada
    EIMSK |=(1<<INT0);  //Habilitar int INT0
    sei();              //Habilitar int globales
    while (1) ;
}

ISR(INT0_vect){
    PORTC ^=(1<<PORTC3);
}
```

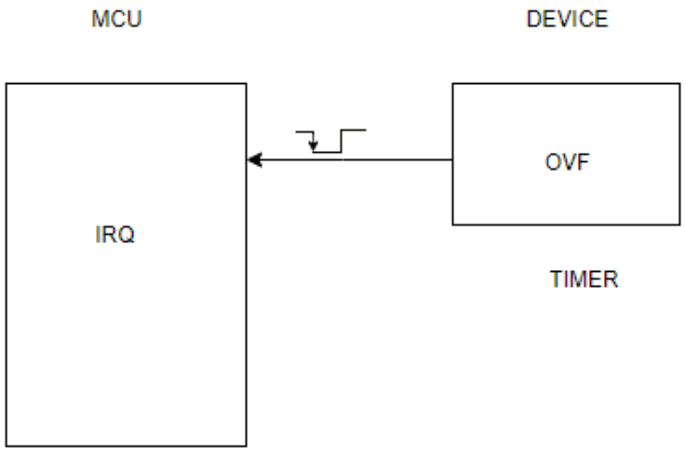
¿Interrupciones por nivel o por flanco?

- Si una interrupción funciona por nivel quiere decir que el periférico que la genera "coloca y mantiene" el nivel en la línea para que el uC atienda a esa petición. Durante la atención, el uC debería indicar al periférico externo, de algún modo, que ha sido atendido para que éste libere el nivel de la línea.
- Notar que al no ser una "petición registrada", si el nivel bajo no está presente cuando las interrupciones están habilitadas, el pedido no será tenido en cuenta.
- Por otro lado, si el periférico no retira el nivel de la línea, continuará solicitando interrupción indefinidamente.
- Por lo tanto, las interrupciones por nivel, no tienen memoria, y requieren de un aviso al periférico para que no se procese la misma interrupción múltiples veces.



¿Interrupciones por nivel o por flanco?

- Si una interrupción funciona por flanco quiere decir que el periférico produce un flanco en la línea y este pedido queda registrado en un Flag (Flip Flop) pidiendo interrupción. Típicamente el uC borra este flag para indicar que esta interrupción ya ha sido atendida sin necesidad de comunicárselo al periférico.
- De esta manera, si las interrupciones están deshabilitas al momento de producirse el flanco, los pedidos quedan "pendientes" y serán atendidos por prioridad cuando se active la máscara de interrupción I.



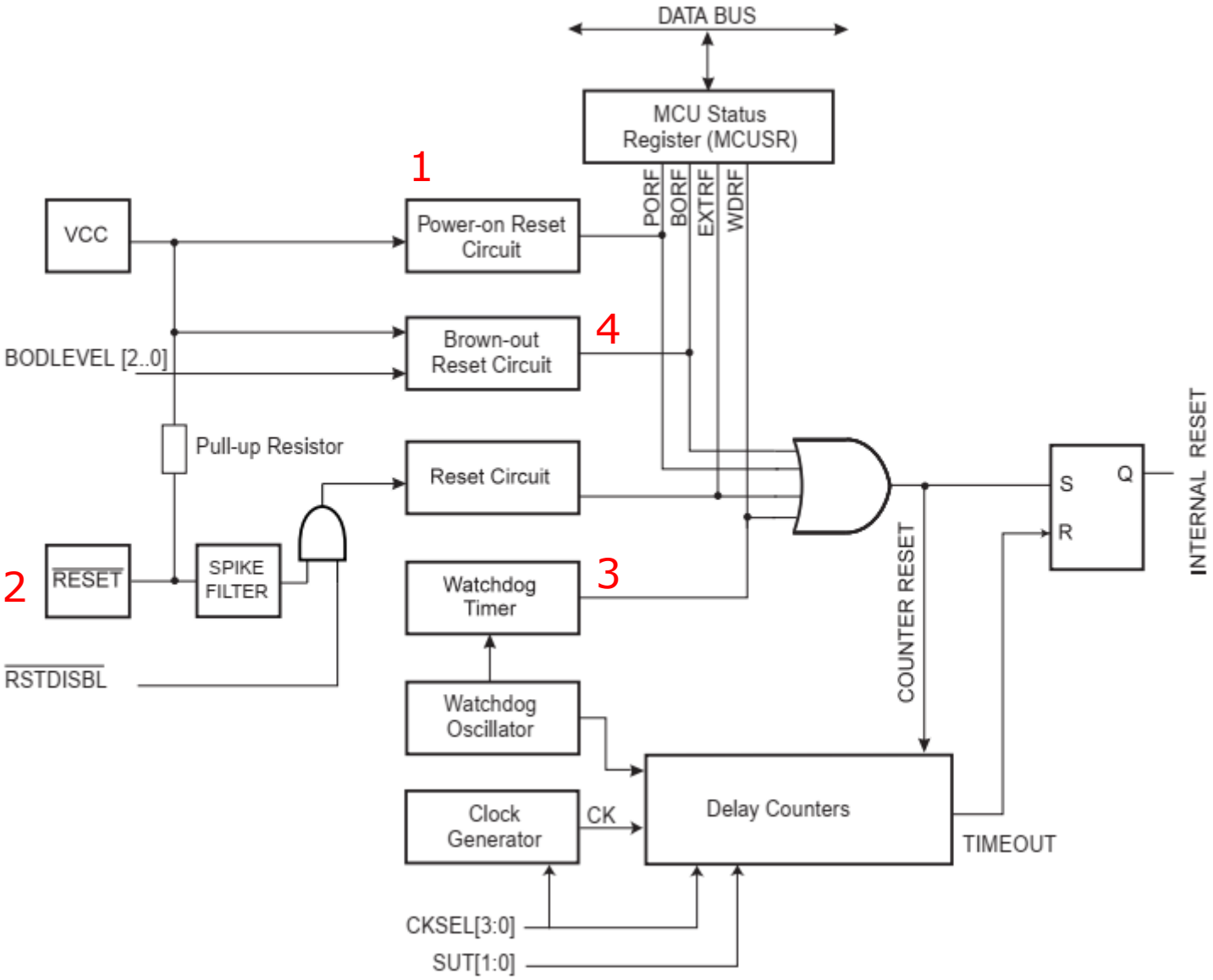
El RESET

- El Reset es un evento que permite inicializar un sistema en un estado por defecto perfectamente conocido.
- En este caso el RESET es una señal digital que puede interpretarse como un pedido de interrupción especial.
- El RESET no puede deshabilitarse o enmascararse.

- Atmega328 posee 4 fuentes distintas de RESET:

  - 1 -Power on (Alimentación): Es un circuito que genera la señal de reset cuando el sistema se enciende.
  - 2 -Reset externo (pulsador): es un terminal donde el usuario puede colocar un pulsador.
  - 3 -Perro guardián (WATCHDOG): es un temporizador capaz de generar un reset bajo ciertas circunstancias.
  - 4 -Brown Out reset (Monitor de Alimentación del micro): es un circuito que genera un reset si la tensión VCC está fuera de los márgenes seguros de operación.

El RESET


DATA BUS
MCU Status Register (MCUSR)
PORF
BORF
EXTRF
WDRF
1
VCC
Power-on Reset Circuit
Brown-out Reset Circuit
4
BODLEVEL [2..0]
Pull-up Resistor
Reset Circuit
2
RESET
SPIKE FILTER
Watchdog Timer
3
RSTDISBL
Watchdog Oscillator
COUNTER RESET
S
Q
R
INTERNAL RESET
Clock Generator
CK
Delay Counters
TIMEOUT
CKSEL[3:0]
SUT[1:0]

El RESET

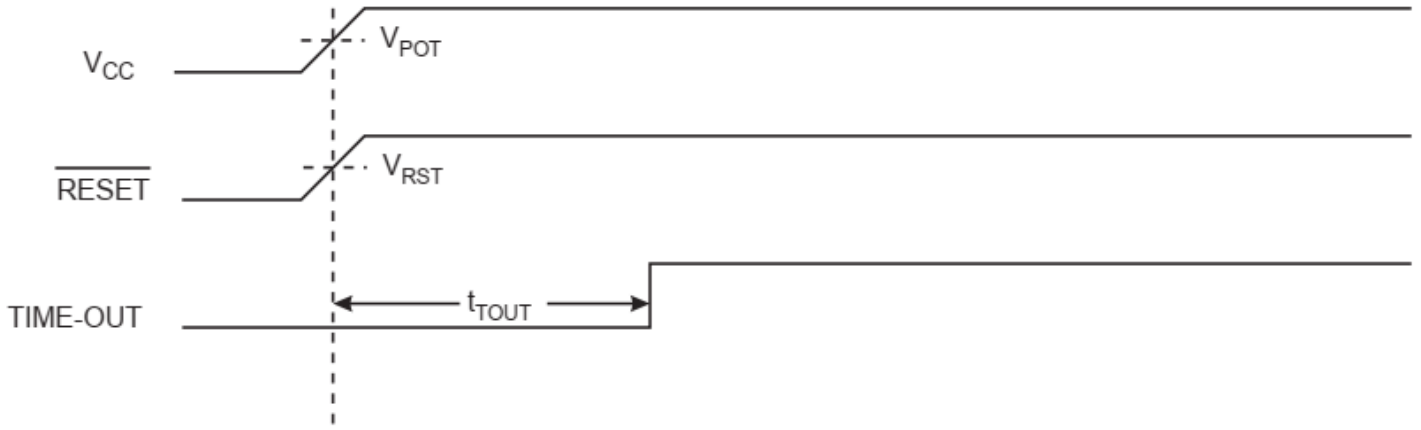
- Cuando se produce un Reset, por alguna de las causas anteriores, el controlador de interrupciones suspende la ejecución de instrucciones CPU y realiza una secuencia de acciones destinada a inicializar todo el sistema.
- Todos los Registros I/O son inicializados a sus valores por defecto.
- Se espera que la alimentación y la fuente de reloj seleccionada estén estables.
- El PC se inicializa con el vector de RESET (primer dirección de la tabla) para ejecutar la primer instrucción
- Para saber cual fue la causa del reset el programa debe revisar el registro

MCUSR – MCU Status Register

7	6	5	4	3	2	1	0	
–	–	–	–	WDRF	BORF	EXTRF	PORF	MCUSR
R	R	R	R	R/W	R/W	R/W	R/W	

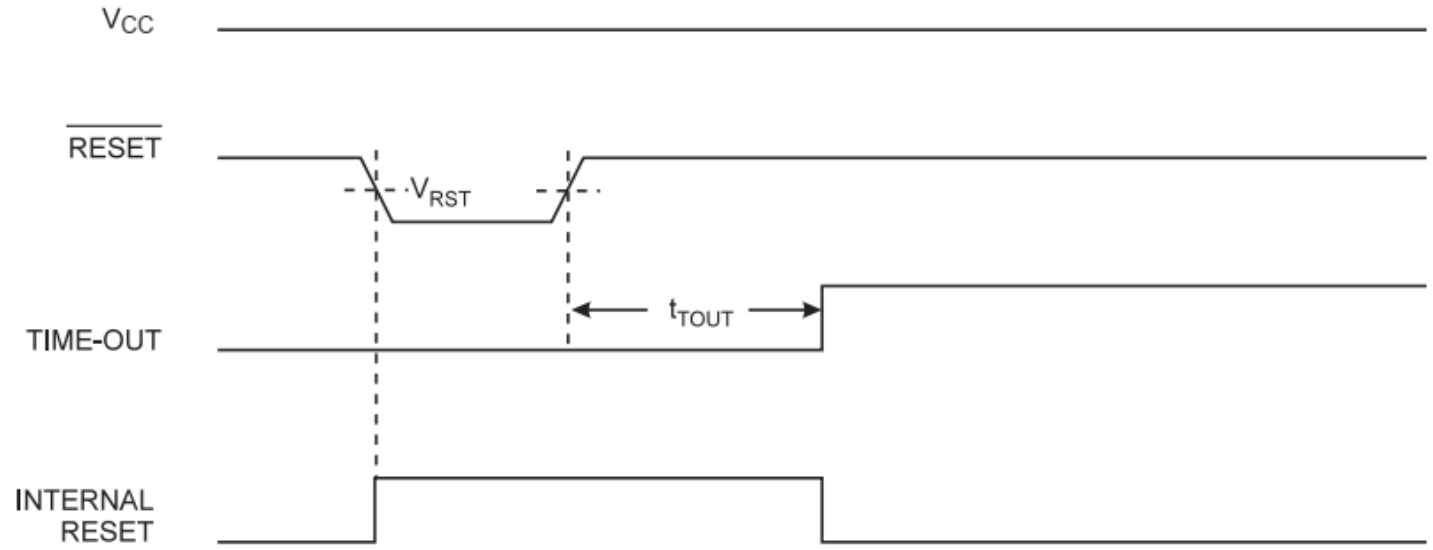
El RESET

Figure 10-2. MCU Start-up, $\overline{\text{RESET}}$ Tied to V_{CC}



1

Figure 10-4. External Reset During Operation

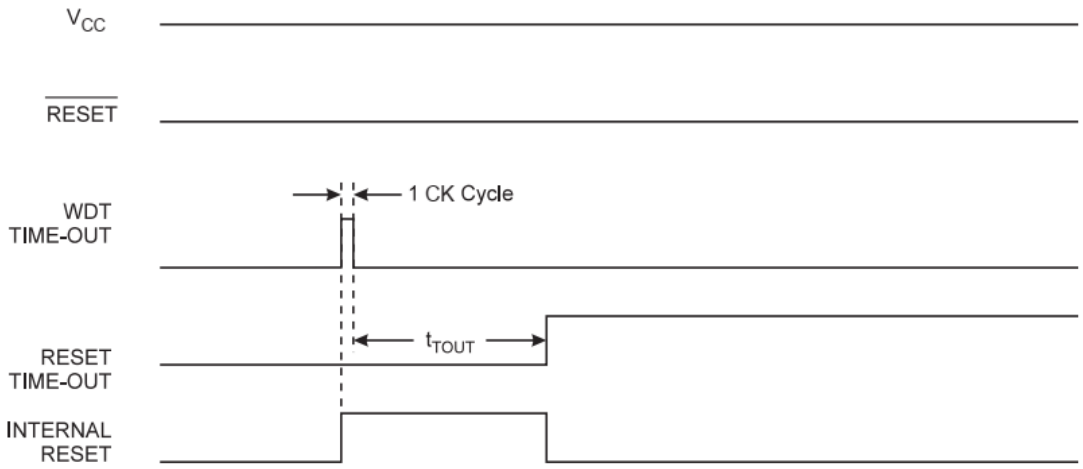


2

EI RESET

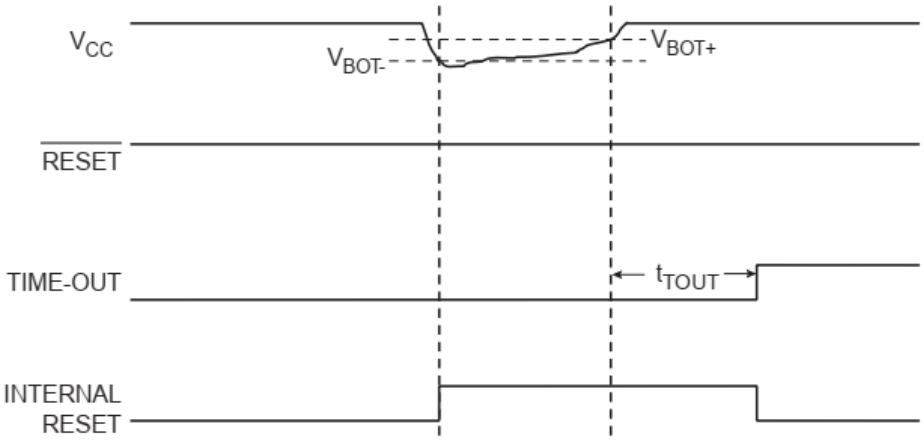
3

Figure 10-6. Watchdog System Reset During Operation



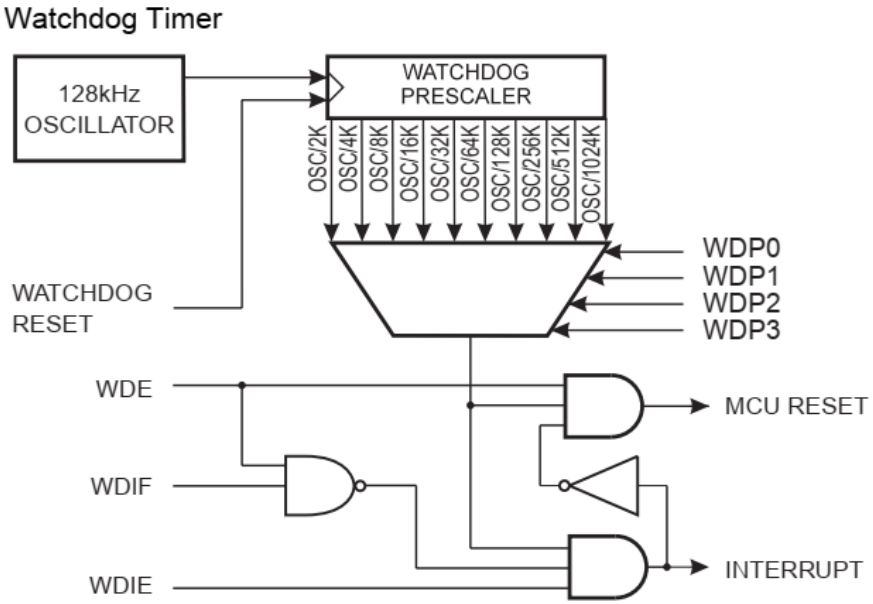
4

Figure 10-5. Brown-out Reset During Operation



El Perro guardian (WDT - Watchdog Timer)

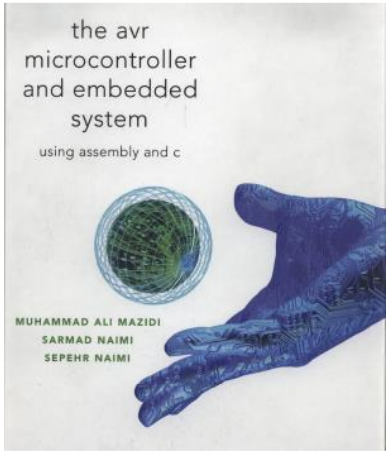
- Es un método de protección del software para detectar anomalías durante la ejecución normal del programa. Por ejemplo, si el programa “se cuelga” en bucle infinito el sistema dejará de responder.
- El watchdog es un temporizador programable que cuenta pulsos de reloj de un oscilador interno



- Si el contador de pulsos llega a un valor prefijado se generará un RESET.
- El programa durante su ejecución debe reiniciar frecuentemente el contador (con la instrucción WRT) para que no se genere dicha señal.

Bibliografía:

- *The AVR microcontroller & Embedded Systems*.
  Mazidi,Naimis, CH10



- Libro Digital de Felipe Espinosa, CH2 y CH4

- Hoja de datos ATMEGA328p
- AVR Studio help

- En arduino : VIDEO: https://www.youtube.com/watch?v=up-goNfJ_EY