Lab 2: Interrupt in AVR ATmega328P

AVR Atmega 328P datasheet: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

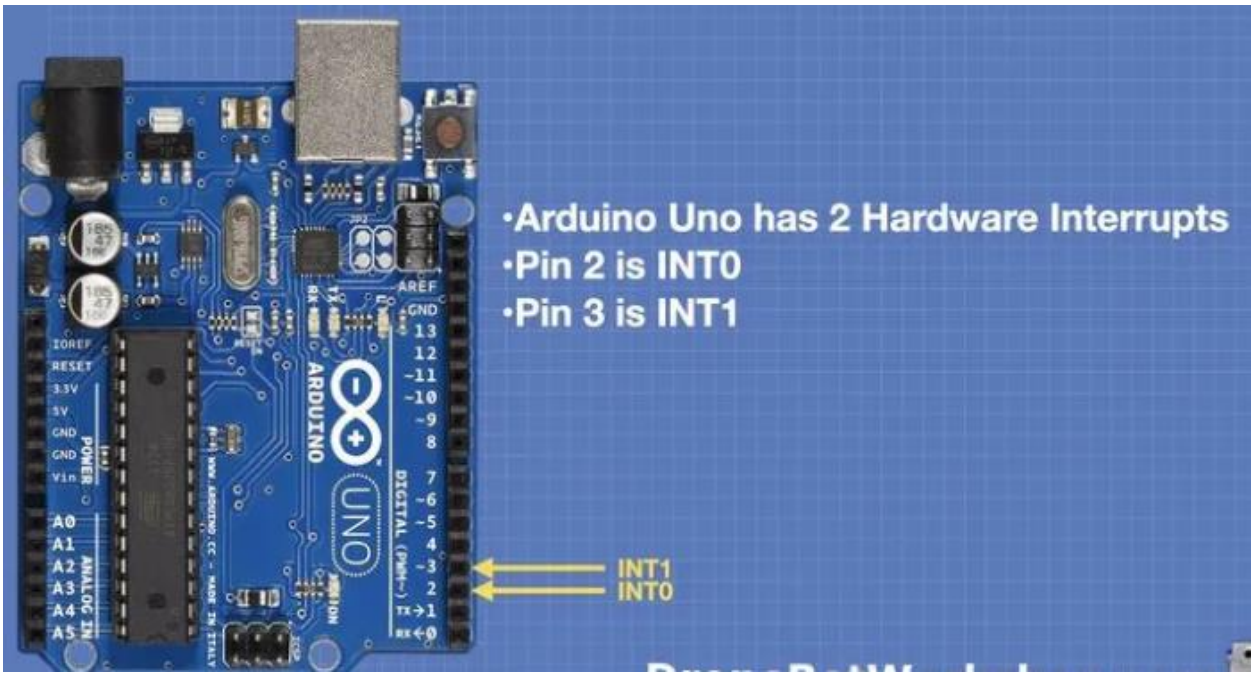
Key registers of AVR Atmega328P

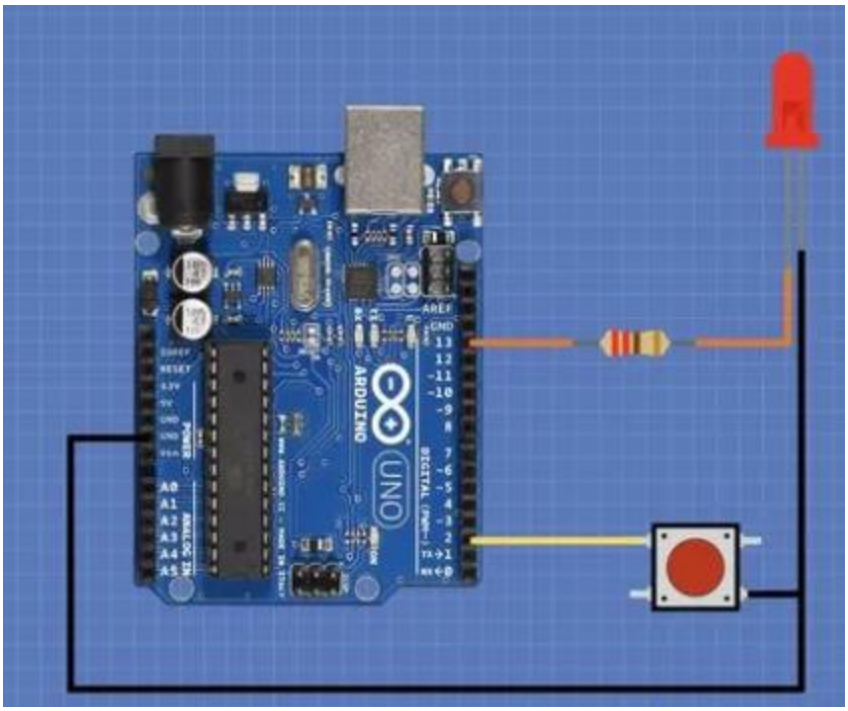
Register	Description
SREG (Status Register)	Global status flags; bit 7 (I-bit) enables global interrupts. Must be set via sei() to allow any interrupt.
EIMSK (External Interrupt Mask Register)	Enables or disables external interrupts. Bit0 = INT0, Bit1 = INT1.
EIFR (External Interrupt Flag Register)	Flags that indicate which external interrupt triggered (read/clear with 1).
EICRA (External Interrupt Control Register A)	Configures trigger type for INT0 and INT1: low level, any change, falling edge, rising edge.
PCICR (Pin Change Interrupt Control Register)	Enables pin-change interrupt groups (PCIE0, PCIE1, PCIE2).
PCMSKx (Pin Change Mask Register)	Selects which pins within a PCINT group can trigger an interrupt. PCMSK0 → PORTB, PCMSK1 → PORTC, PCMSK2 → PORTD.
TIMSKx (Timer Interrupt Mask Register)	Enables specific timer interrupts (overflow, compare match, etc.).
TIFRx (Timer Interrupt Flag Register)	Interrupt flag register for timers. Flags are cleared by writing 1.
ADCSRA (ADC Control and Status Register A)	Contains ADIE bit to enable ADC conversion complete interrupt.
sei()/cli()	Assembly instructions (Set/Clear Interrupts) to enable or disable global interrupts.

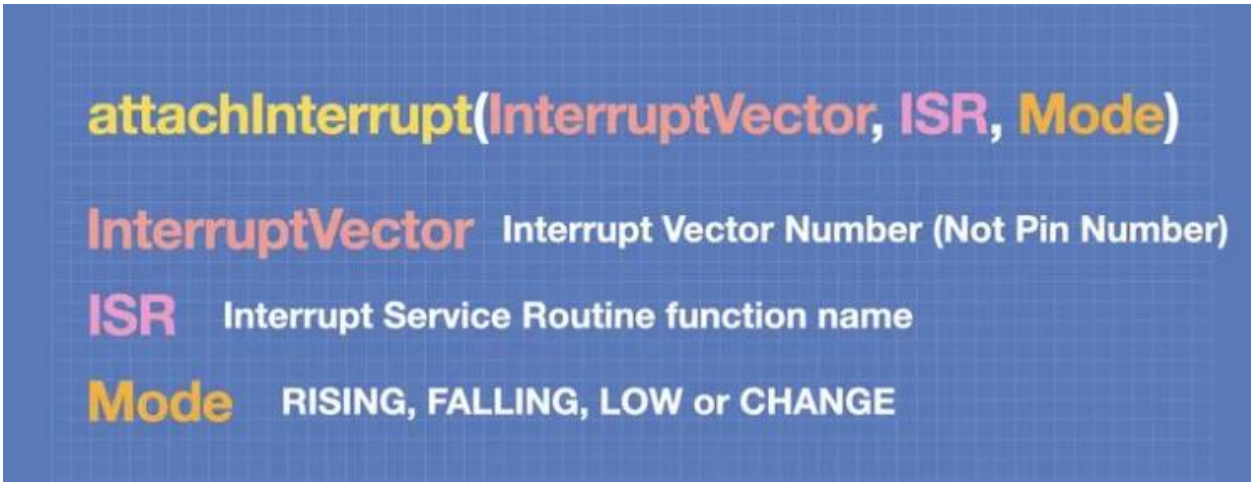
Arduino programming

#	Exercise	Focus	Arduino Function	Bare-Metal Register
1	External Interrupt (Button on INT0)	Edge triggering	`attachInterrupt()`	EICRA, EIMSK, EIFR
2	Pin Change Interrupt	Any pin change	None (custom ISR)	PCICR, PCMSKx
3	Timer Interrupt	Periodic trigger	`TimerOne`/`millis()`	TCCR1x, TIMSK1
4	ADC Interrupt	End-of-conversion ISR	`analogRead()` w/ ISR	ADCSRA, ADMUX, ADIE
5	Latency measurement	Measure ISR timing	Digital toggling	PORT toggling before/after ISR

Task: Use the **push button** on pin **D2 (INT0)** to toggle an LED in which INT0 is with an interrupt and triggered on a falling edge. Write two separate codes, including one with Arduino programming and one with bare metal programming for the case above. Explain the value of register: EIFR in cases of a normal case and an interrupt case







- **Interrupt Vector** – the interrupt that you wish to use. Note that this is the internal Interrupt Vector number and NOT the pin number.
- **ISR** – The name of the Interrupt Service Routine function that you are gluing to the interrupt.
- **Mode** – How you want to trigger the interrupt.

For Mode, there are four selections:

- **RISING** – Triggers when an input goes from LOW to HIGH.
- **FALLING** – Triggers when an input goes from HIGH to LOW.
- **LOW** – Triggered when the input stays LOW.

- **CHANGE** – Triggers whenever the input changes state from HIGH to LOW or LOW to HIGH.
-

Note that for Arduino programming: using attachInterrupt(digitalPinToInterrupt(buttonPin), toggleLED, FALLING);
For bare metal programming using **EICRA**, **EIMSK**, and **SREG**

Understanding SREG:

Bit	Name	Meaning
7	I	Global Interrupt Enable
6	T	Bit Copy Storage
5	H	Half Carry Flag
4	S	Sign Flag
3	V	Two's Complement Overflow Flag
2	N	Negative Flag
1	Z	Zero Flag
0	C	Carry Flag

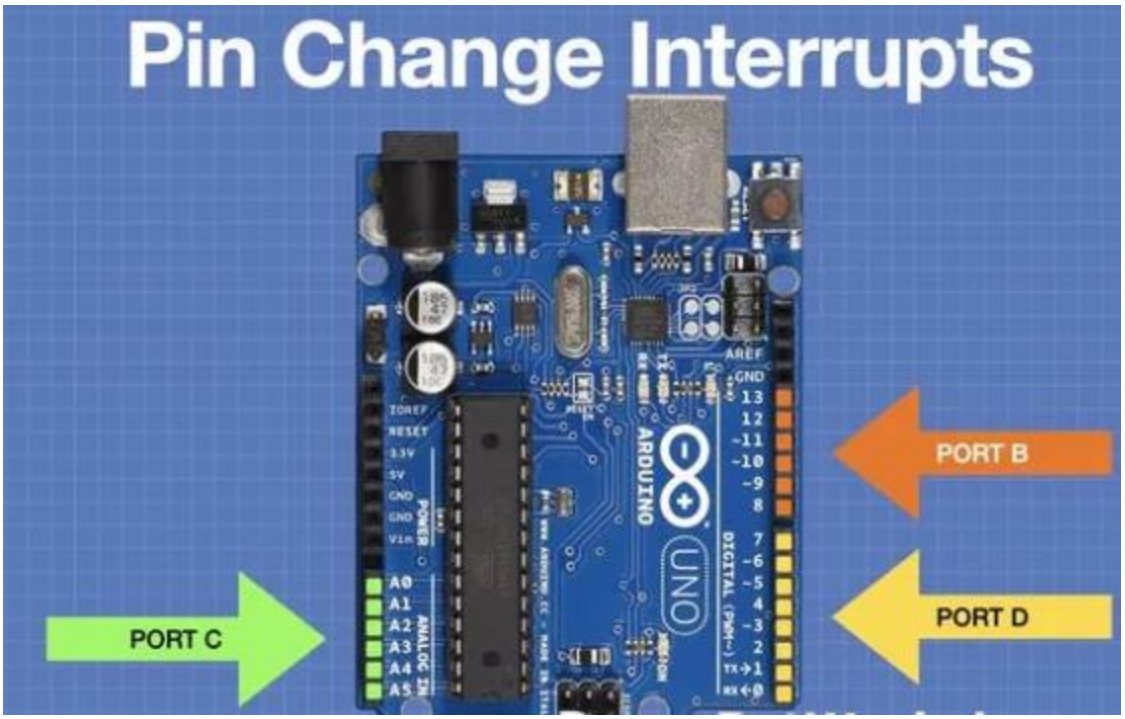
`cli()` and `sei()` are macros that clear and set the **I bit** of SREG, respectively.

Exercise 3: Interrupt when pin changes

Make 2 programs (one with Arduino programming and one with bare metal programming) for triggering an interrupt when pin 8 of Arduino changes from HIGH to LOW (Note that this pin 8 is with a pull-up resistor)

In Arduino programming, using pinMode(pin, INPUT_PULLUP); for setting pull-up

In bare metal programming, using PORTB for setting up pull-up



Register	Bits	Description
PCICR	PCIE2..0	Enable pin change interrupt groups (B=0, C=1, D=2)
PCMSK0	PCINT7..0	Select pins on PORTB (PB0–PB7)
PCMSK1	PCINT14..8	Select pins on PORTC
PCMSK2	PCINT23..16	Select pins on PORTD
PCIFR	PCIFx	Flag cleared by writing 1

Using these registers PCICR, PCMSK0, and sei() for setting interrupt

Exercise 4: using Timer Interrupt

Toggle an LED every 1 second using Timer1 Compare Match A interrupt. Write a bare metal program for this

Using the below registers for triggering timer 1 interrupt

Register	Bits	Function
TCCR1A/B	WGM12	CTC mode selection
CS12..0	Prescaler bits (1024 in this example)	
OCR1A	Compare match value	

Register	Bits	Function
TIMSK1	OCIE1A bit enables interrupt	
TIFR1	OCF1A flag cleared on ISR execution	

```
TCCR1B = (1 << WGM12) | (1 << CS12) | (1 << CS10);
```

Note:

TCCR1B is the Timer/Counter1 Control Register B. It controls how Timer1 counts and operates. Let's analyze it

Timer mode: WGM12

- WGM12 is Waveform Generation Mode bit 12**.**
- Timer1 has multiple modes (normal, CTC, PWM, etc.), selected using WGM13:0 (4 bits across TCCR1A and TCCR1B):

Mode	WGM13	WGM12	WGM11	WGM10	Mode Name
4	0	1	0	0	CTC (Clear Timer on Compare Match)

- Setting (1 << WGM12) means CTC mode.
  - In CTC mode, the timer counts from 0 to the value in OCR1A.
  - When the timer reaches OCR1A, it **resets to 0** and optionally triggers an interrupt.

Prescaler: CS12 and CS10

- CS12 and CS10 are Clock Select bits for Timer1.
- These determine the prescaler, i.e., how much the CPU clock is divided for the timer:

CS12	CS11	CS10	Prescaler
1	0	1	1024

- (1 << CS12) | (1 << CS10) sets CS12=1, CS11=0, CS10=1 → prescaler = 1024.
- This means the timer clock is F_CPU / 1024.
  For example, if F_CPU = 16 MHz, timer ticks every:

  Tick Time=1024/(64*10^6) = 64 microseconds

```
OCR1A = 15625;
```

OCR1A is the Output Compare Register A for Timer1.

- In CTC mode, the timer counts from 0 up to `OCR1A`.
- When the count matches `OCR1A`, the compare match event occurs and the timer resets.

Calculating 1 Hz

We want 1 Hz (1 event per second).

- Timer tick time = `64 µs` (from prescaler calculation).
- Number of ticks for 1 second:

Ticks=1/64 microseconds = 15625

That matches `OCR1A = 15625`.

So every 15625 timer ticks, the compare match triggers, which is once per second.

```
TIMSK1 = (1 << OCIE1A);
```

`TIMSK1` is the Timer/Counter1 Interrupt Mask Register.

- `OCIE1A` = Output Compare A Match Interrupt Enable.
- `(1 << OCIE1A)` enables an interrupt whenever Timer1 matches OCR1A.

This means every second (as per OCR1A calculation), an ISR (interrupt service routine) will run.

Putting it all together

1. **Set Timer1 to CTC mode** → timer resets at OCR1A.
2. **Set prescaler to 1024** → timer ticks slower, allowing long delays with 16-bit counter.
3. **Set OCR1A = 15625** → compare match occurs every 1 second.
4. **Enable compare interrupt** → ISR executes on each 1-second interval.