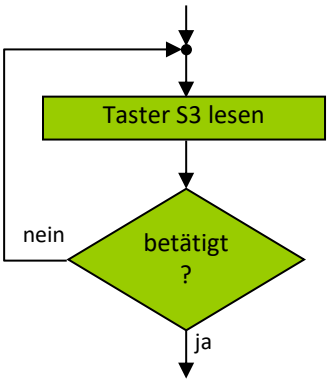
1. Port-Polling



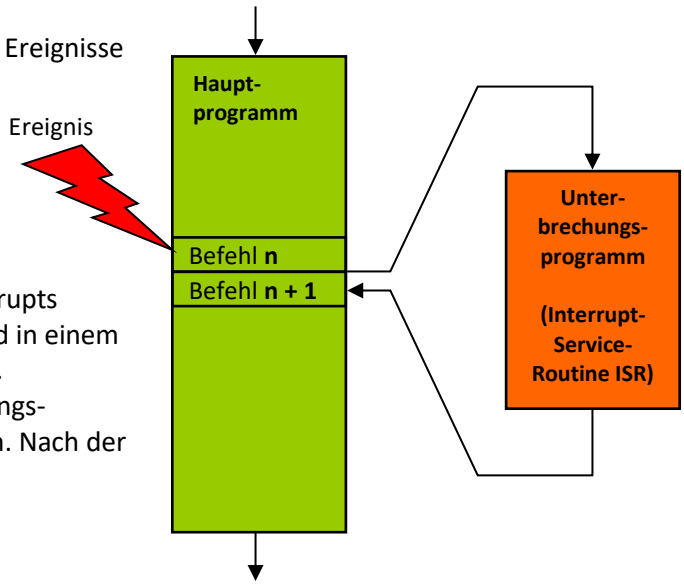
Beim Portpolling wird ein Eingangspin des Controllers in einer Schleife zyklisch abgefragt (Polling-Schleife). Dabei handelt es sich um eine reine Softwarelösung die sehr einfach zu realisieren ist. Z.B.:

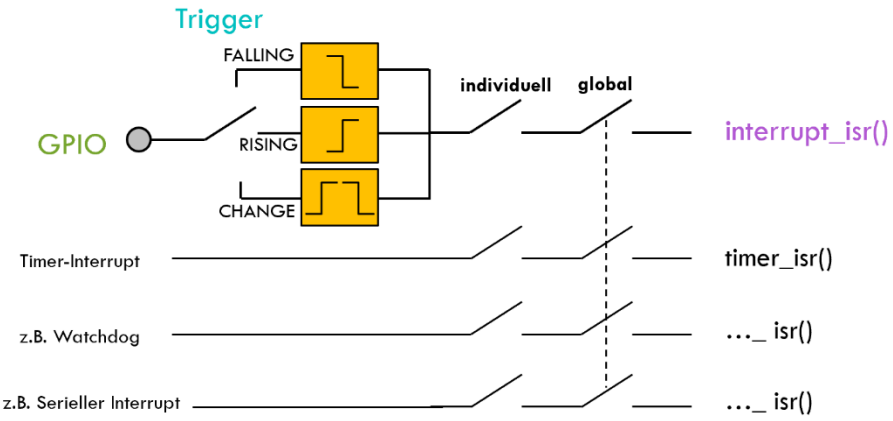
```
bool state
do
{
 state = digitalRead(tasterS3);
} while (state != LOW);
```

Nachteilig ist allerdings, dass die Eingabeleitung nur an jeweils einer Programmstelle abgefragt wird. Befindet sich die Programmabarbeitung nicht in der Polling-Schleife, findet keine Reaktion auf ein externes Ereignis statt. Andererseits führt das "Warten" in der Abfrageschleife dazu, dass der Controller keine weiteren Aufgaben durchführen kann. Insbesondere zeitkritische Funktionen (z.B. Not-Aus) lassen sich mit dieser Abfrageart kaum realisieren.

2. Interrupts

Interrupts bieten die Möglichkeit, auf bestimmte Ereignisse schnell und sicher zu reagieren, ohne diese ständig im Programm abfragen zu müssen.
Das Interrupt-System unterbricht direkt den normalen Programmablauf.
Wird ein Ereignis vom Interrupt-System erkannt, wird der aktuelle **Befehl n** fertig ausgeführt und anschließend auf die **Einsprungadresse** des Interrupts verzweigt. Die Rückkehradresse (**Befehl n+1**) wird in einem speziellen Speicherbereich (Stack) abgespeichert.
Auf der Einsprungadresse muss das Unterbrechungs-programm (Interrupt-Service-Routine **ISR**) stehen. Nach der Abarbeitung der ISR wird zurück zum **Befehl n+1** gesprungen.





Damit Interrupts erkannt werden, müssen sie freigegeben werden. Alle Interrupt-Quellen eines Mikrocontrollers haben je eine individuelle und eine gemeinsame (globale) Freigabe. Externe Interrupts erkennen bestimmte Signale an einem speziellen Eingangspin. Im einfachsten Falle sind dies Flanken (0→1, 1→0) oder Pegel (LOW/HIGH).

Beim ESP32 kann jeder GPIO als externer Interrupteingang verwendet werden.

Mit den folgenden Befehlen wird die Verwendung eines Pins als externer Interrupt gesteuert:

```
attachInterrupt ( GPIO, isr, Trigger )      // Freigeben eines Interrupts
dettachInterrupt( GPIO )                    // Sperren eines Interrupts
```

Parameter	Beschreibung	Werte
GPIO	Pinnummer	Alle GPIOs
Trigger	Interruptereignis	RISING = Steigende Flanke (0→1) FALLING = Fallende Flanke (1→0) CHANGE = Beide Flanken (1→0, 0→1) LOW = Low-Pegel (0)[1]
isr	Name der Interrupt Service Routine (ohne Klammern)	**Bsp.**: Interrupt-Freigabe attachInterrupt(S3,toggle_isr,RISING);

Ein Interrupt wird analog einer Funktion definiert und deklariert. Er bekommt aber niemals Parameter übergeben und gibt auch keinen Wert zurück. Mit dem Hauptprogramm kann er nur über globale Variablen Daten austauschen. Die Deklaration einer Interrupt-Service-Routine (ISR) lautet folgendermaßen:

```
void IRAM_ATTR isr_name ( void );
```

Der Speicherqualifizierer **IRAM_ATTR** gibt an, dass die ISR vom C-Startcode in den internen RAM verschoben wird. Dies bewirkt eine deutlich schnellere Ausführung als ein Aufruf im Programmspeicher (Flash-ROM). Die ISR wird aufgerufen, wenn die Interrupt-Hardware das konfigurierte Ereignis erkennt. Da die sie wie eine Funktion deklariert/definiert ist, darf sie prinzipiell auch wie eine Funktion aufgerufen werden.

Globale Variablen, die in der ISR geändert werden, sollten mit dem Schlüsselwort **volatile** deklariert werden. Dies verhindert, dass der Compiler die Variable wegoptimiert und unerwünschte Effekte auftreten.

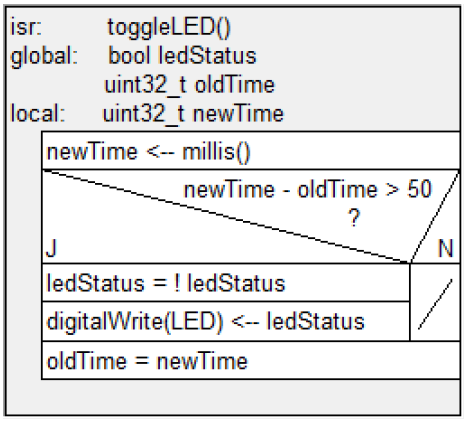
```
const int LED = 32;
const int S3  = 19;

volatile bool LedStatus = false;

void IRAM_ATTR toggle_isr(void)
{
  LedStatus = !LedStatus;
  digitalWrite(LED,LedStatus);
}

void setup()
{
  pinMode(LED,OUTPUT);
  pinMode(S3,INPUT_PULLUP);
  attachInterrupt(S3,toggle_isr,FALLING);
}

void loop() {}
```

Arbeitsauftrag

1. Erstellen Sie den abgebildeten Code zum Testen des externen Interrupts an Pin 5 (S3). Beachten Sie, dass hier in der loop() keine Anweisungen ausgeführt werden.

   Sie werden feststellen, dass das auch hier das togglen der LED aufgrund von Tasterprellen nicht immer funktioniert, da jeder „Preller“ auch eine fallende Flanke hat.

2. Ergänzen Sie eine Tasterentprellung mit **millis()**. Denken Sie daran, die Variable **oldTime** als **volatile** zu deklarieren.



3. Vermutlich werden immer noch Fehler auftreten. Insbesondere bei längerem drücken des Tasters. Überlegen Sie, woran das liegen könnte. (Hinweis: Der Interrupt wird nur bei fallender Flanke ausgelöst und nicht so lange die Taste betätigt ist.)
   Probieren Sie, ob sich das Verhalten verbessern lässt, wenn Sie die Impulsdauer von 50 auf 250 erhöhen.

[1] Pegel-Interrupts werden nicht unterstützt.