

# INTERRUPT

Reagieren auf unerwartete Ereignisse



# INTERRUPT

Der  $\mu$ Controller muss schnell und unmittelbar jederzeit bereit sein auf folgende Ereignisse zu reagieren:

- Not-Aus-Taster
- Lichtschranke
- Andere Signale mit höchster Priorität



# INTERRUPT

Funktionsprinzip:

Wann geht's los?



# INTERRUPT

Funktionsprinzip:



Das  
Hauptprogramm  
läuft in der  
Endlosschleife



# INTERRUPT

Funktionsprinzip:



Hauptprogramm

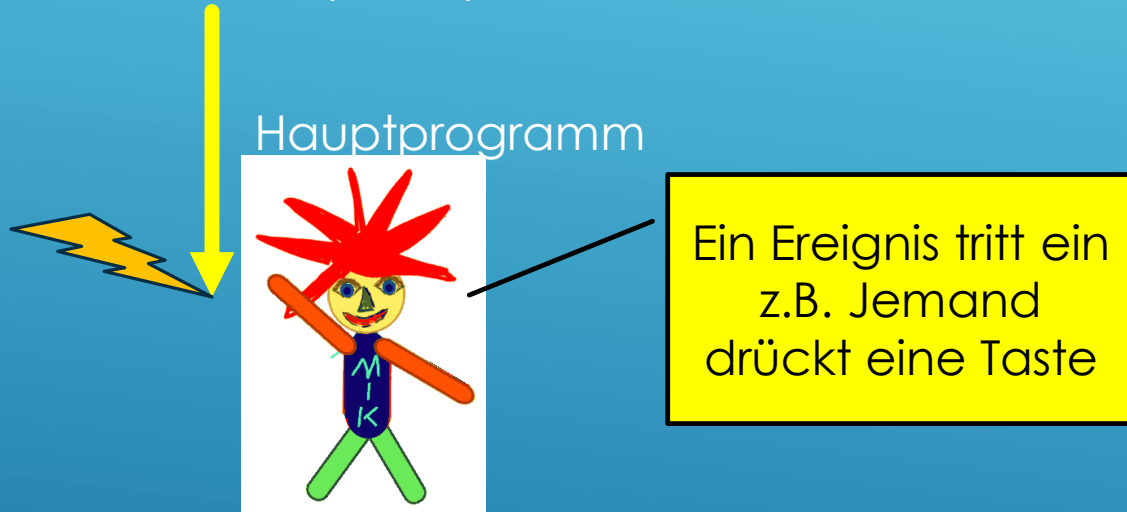


Das  
Hauptprogramm  
läuft in der  
Endlosschleife



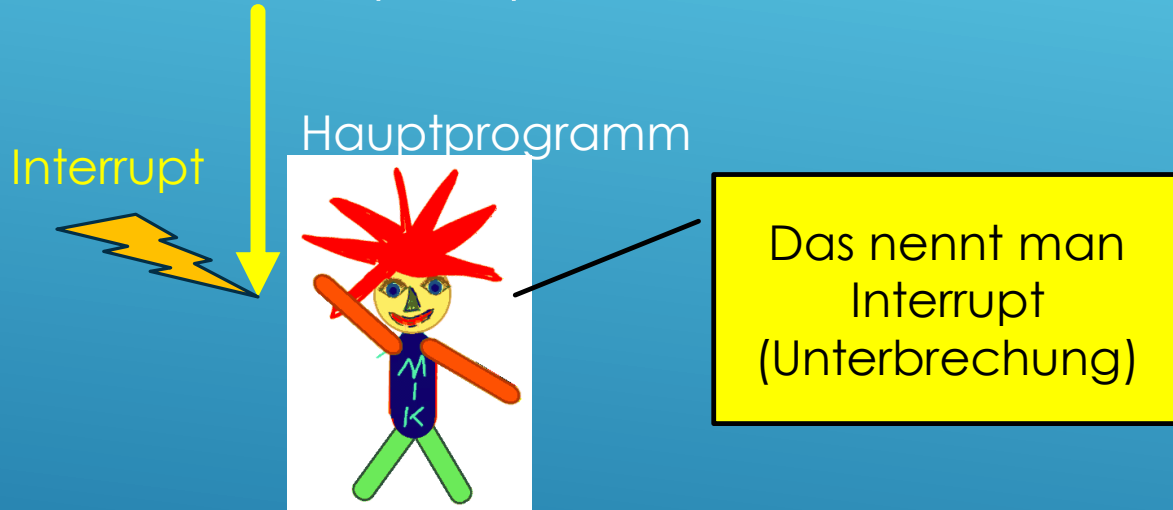
# INTERRUPT

Funktionsprinzip:



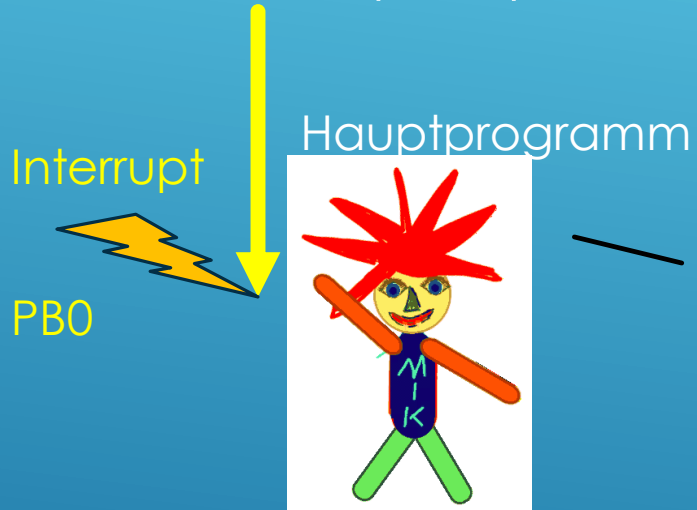
# INTERRUPT

Funktionsprinzip:



# INTERRUPT

Funktionsprinzip:



Beim Interrupt handelt es sich um ein Ereignis wie z.B. Ein Tastendruck auf PB0, verursacht vielleicht durch eine Lichtschranke.





# INTERRUPT

Funktionsprinzip:



Beim Interrupt handelt es sich um ein Ereignis wie z.B. Ein Tastendruck auf PBO, verursacht vielleicht durch eine Lichtschranke.



Der Prozessor verzweigt automatisch in ein Unterprogramm. Die Interrupt Service Routine (ISR)



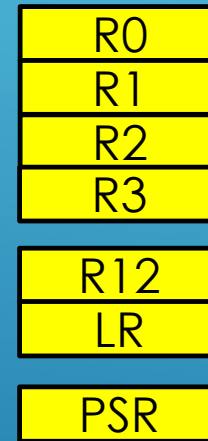
# INTERRUPT

Funktionsprinzip:



Beim Interrupt handelt es sich um ein Ereignis wie z.B. Ein Tastendruck auf PB0, verursacht vielleicht durch eine Lichtschranke.

Die Register R0..R3, R12, LR und PSR werden in einem speziellen Bereich des RAM, dem sogenannten Stack gesichert

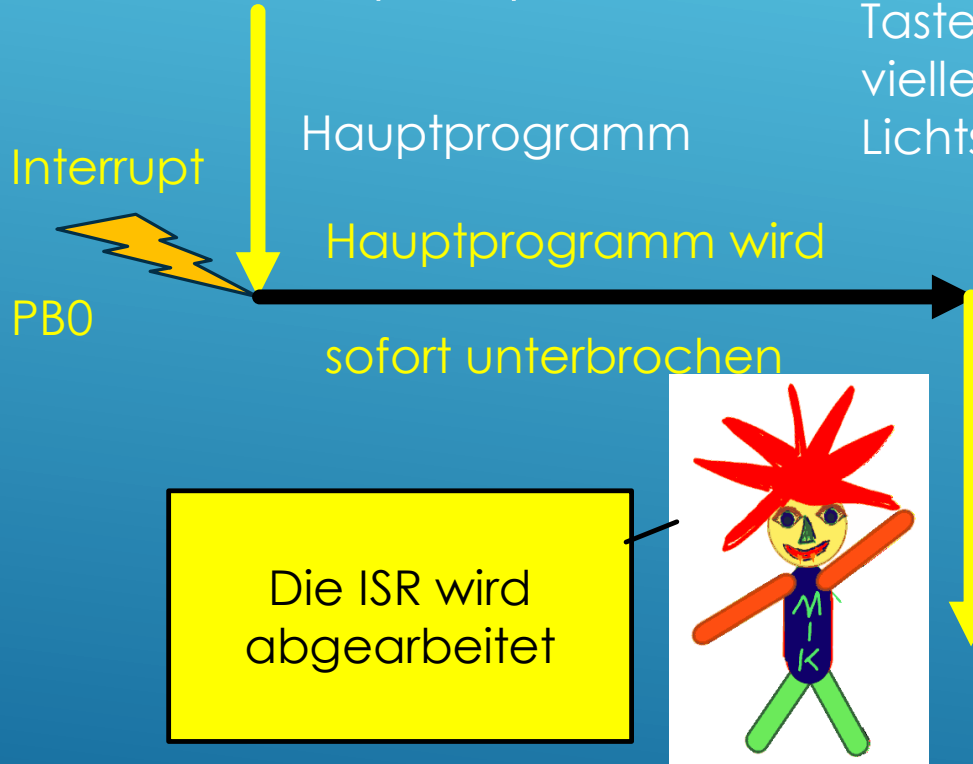


Stack (RAM)



# INTERRUPT

Funktionsprinzip:



Beim Interrupt handelt es sich um ein Ereignis wie z.B. Ein Tastendruck auf PB0, verursacht vielleicht durch eine Lichtschranke.

Ein Unterprogramm, die sogenannte Interrupt Service Routine kurz ISR wird automatisch gestartet und führt die programmierten Instruktionen aus.



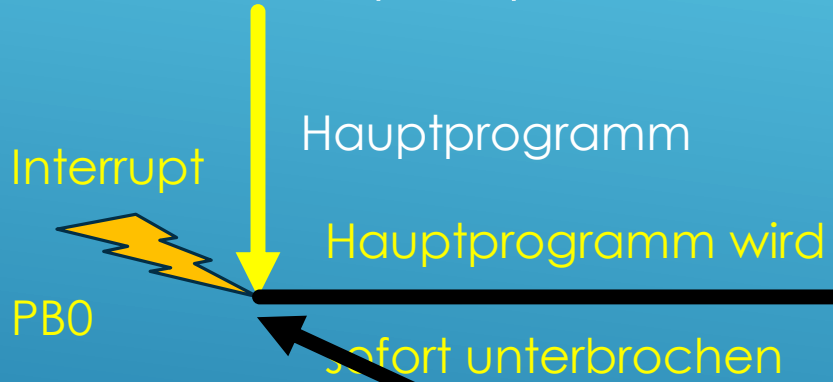
# INTERRUPT

Funktionsprinzip:



# INTERRUPT

Funktionsprinzip:

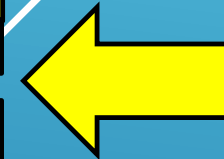
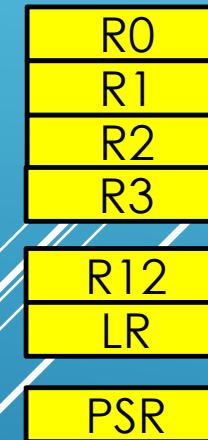


Beim Interrupt handelt es sich um ein Ereignis wie z.B. Ein Tastendruck auf PBO, verursacht vielleicht durch eine Lichtschranke.

Ein Unterprogramm, die sogenannte Interrupt Service Routine kurz ISR wird automatisch gestartet und führt die programmierten Instruktionen aus.



Die auf dem Stack (Bereich im RAM) gesicherten Register R0..R3, R12, LR, PSR werden zurückgeholt



Stack (RAM)



# INTERRUPT

Funktionsprinzip:



# INTERRUPT

Voraussetzung: Der Portpin muss als Eingang (gegebenenfalls mit Pull) eingestellt sein.

1. NVIC alle Interrupts einschalten

```
ldr    R4,=nvic
ldr    R3,=0xFFFFFFFF
str    R3,[R4,isrEnableReg0]
str    R3,[R4,isrEnableReg1]
```



Das ist zu tun

2. Interruptquelle auswählen

```
ldr R4,=SysCFG
ldr  R3,=0b0001000100010001 //EXTI 3..0 GPIOB: PB3, PB2, Pb1, PB0
str  R3,[R4,SYSCFG_EXTICR1]
```

3. Steigende Flanke

```
ldr    R4,=EXTI    //external Interrupt Controller
mov    R3,Bit0     //Bit0 = PB0 mit steigender Flanke
str    R3,[R4,RTSR] //RTSR = rising Transitions Register
```



# INTERRUPT

Voraussetzung: Der Portpin muss als Eingang (gegebenenfalls mit Pull) eingestellt sein.

## 4. Interrupt freigeben

```
ldr    R4,=EXTI    //external Interrupt Controller
mov    R3,Bit0     //Bit0 = PB0 Interrupt freigeben
str     R3,[R4,IMR] //IMR = Interrupt Mask Register
```





# INTERRUPT

Voraussetzung: Der Portpin muss als Eingang (gegebenenfalls mit Pull) eingestellt sein.

## 5. ISR (nach der Endlosschleife)

```
.global meinInterrupt
```

```
meinInterrupt:
```

```
//Pendingbit zurücksetzen
```

```
ldr    R4,=EXTI
```

```
mov    R3,Bit0    //Bit0 gehört zum externen Interrupt 0
```

```
str    R3,[R4,PR] //Zum Rücksetzen des Pendingbits muss eine 1  
                // in das entsprechende Bit im Pending  
                //Register (PR) eingetragen werden
```

```
//ISR Code
```

```
bx     lr          //Return
```



# INTERRUPT

Voraussetzung: Der Portpin muss als Eingang (gegebenenfalls mit Pull) eingestellt sein.

7. ISR in Vektortabelle in Startup\startup\_stm32l152retx.s eintragen:

```
/*  
g_pfnVectors:  
    .word _estack  
    ...  
    .word RCC_IRQHandler  
    .word meinInterrupt+1 //EXTI0_IRQHandler  
    .word EXTI1_IRQHandler  
    ...  
*/
```



Startadresse  
der ISR **+1**



# INTERRUPT

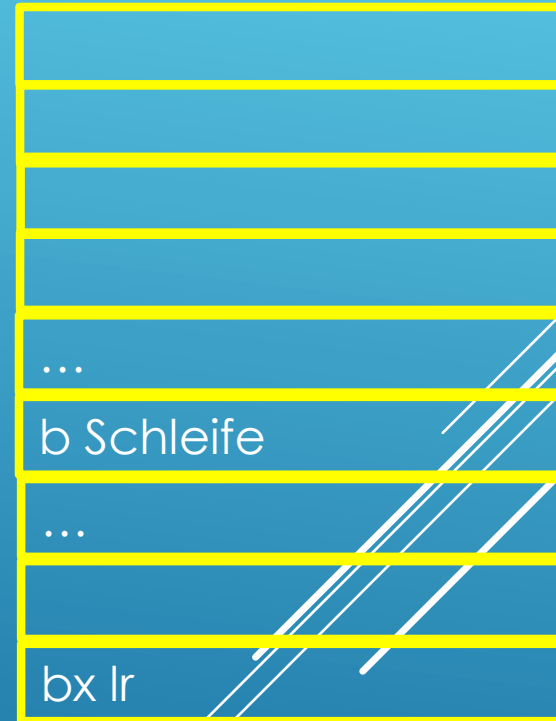
Anordnung der ISR im Codespeicher:

Die Interrupt-Service-Routine wird dem Interrupt zugeordnet, indem in einer Sprungtabelle, der Interruptvectortabelle die Anfangsadresse der ISR eingetragen wird.

main:

Schleife:

ISR:



Hauptprogramm

ISR endet immer mit bx lr  
Return from Interrupt



# INTERRUPT

Jetzt probieren wir das aus!!!

