# Kap.2
# Befehlsschnittstelle

## Prozessoren,
## externe Sicht

⌘**2.1** elementare Datentypen, Operationen

⌘**2.2** logische Speicherorganisation

⌘**2.3** Maschinenbefehlssatz

⌘**2.4** Klassifikation von Befehlssätzen

⌘**2.5** Ausnahmesituationen bzw. Exceptions

⌘**2.6** Prozesse

# 2.5 Ausnahmesituationen bzw. Exceptions

- Situationen während der Bearbeitung von Programmen, die besondere Behandlung erfordern
- Folgende Mechanismen lassen sich unter dem Begriff „Exceptions“ zusammenfassen:
  - Softwareinterrupts
  - Traps
  - (Hardware-)Interrupts

# Prozessorinterne Exceptions

- **Definition**
  Ein **prozessorinternes** Ereignis, welches den normalen Programmablauf unterbricht, heißt **prozessorinterne Exception.**

- Man unterscheidet verschiedene Arten von **prozessorinternen Exceptions**:
  - **Softwareinterrupts:** Über Softwareinterrupts lassen sich (z.B. auch im User-Modus) Betriebssystemroutinen aufrufen. In ihrer Wirkung entsprechen sie Unterprogrammaufrufen. Der aufgerufene Softwareinterrupt wird nicht direkt durch Angabe der Adresse des Programmcodes, sondern durch eine Interruptnummer spezifiziert.
  - **Traps:** Ausnahmesituationen, die durch prozessorinterne Fehler hervorgerufen werden.

# Traps

- **Beispiele:**
  - arithmetischer Überlauf
  - Division durch Null
  - ungültiger Befehlscode
  - Seitenfehler beim Speicherzugriff eines mit Paging arbeitenden Prozessors
  - ...
- **Ablauf:**
  - Schreibe Rücksprungadresse auf Stack oder in dafür vorgesehenes Register
  - Schreibe Registerinhalte auf Stack
  - Rufe Betriebssystemroutine zur Fehlerbehandlung auf, die
    - das aktuelle User-Programm mit Fehlermeldung abbricht (z.B. bei Division durch Null), d.h. nicht mehr zurückspringt, oder
    - die Fehlersituation behandelt (bei Seitenfehler z.B. Laden einer neuen Seite in den Hauptspeicher oder Fehlermeldung bei Overflow), die Registerinhalte wieder rekonstruiert und zurückspringt

# Alternativen zu Traps

- Je nach Prozessor werden nicht bei allen prozessorinternen Fehlersituationen Traps ausgelöst.
- Z.B. bei arithmetischen Überläufen wird häufig auch wie folgt vorgegangen:
  - Ein bestimmtes *Flag* im *Condition-Code-Register* wird gesetzt.
  - Der Programmierer muss selbst den Inhalt des Flags im folgenden Programm auswerten und entsprechend reagieren.

# Interrupts (1)

## Definition

Ein **prozessorexternes** Ereignis, welches den normalen Programmablauf unterbricht, heißt **Hardwareinterrupt** oder kurz **Interrupt.**

Interrupts werden durch asynchron zum Prozessor laufende externe Geräte erzeugt und dienen zur Kommunikation mit dem Prozessor.

## Beispiele

- Eingabe über Tastatur/Maus
- Diskettenlaufwerk / Festplatte meldet, daß Eingabedaten vorhanden sind (die vorher durch ein Kommando an das externe Gerät angefordert wurden).
- Externes Gerät (Festplatte,Netzwerk) meldet, daß Datenübertragung beendet.
- Externes Gerät meldet Fehler bei Datenübertragung.
- ...

# Interrupts (2)

Interrupts werden mit densselben Mechanismen behandelt wie prozessorinterne Exceptions.

**Beispiel**

Der Nutzer drückt eine Taste der Tastatur

- CPU unterbricht aktuelles Programm
- Schreibe Rücksprungadresse auf Stack oder in dafür vorgesehenes Register
- Schreibe Registerinhalte auf Stack
- Rufe Betriebssystemroutine auf, die folgendes tut:
  - Tastaturcode wird überprüft
  - weitere Aktionen (z.B. nach Ctrl-Alt-Del) oder Speicherung im Tastaturpuffer
- Tätigkeit wird an unterbrochener Stelle im alten Zustand fortgesetzt

# Exkurs: Alternativen zu Interrupt-driven IO

## ⌘ Alternative 1: Polling

- Um anliegende Eingaben oder Rückmeldungen externer Geräte zu erkennen, führt der Prozessor in regelmäßigen Abständen Softwareinterrupts aus, um den Status der externen Geräte der Reihe nach abzufragen.
- **Nachteil**: Hoher Overhead durch „unnötige Abfragen“

## ⌘ Alternative 2: DMA = Direct Memory Access

- Kommunikation direkt zwischen Hauptspeicher und externem Gerät.
- **Vorteil**: Entlastung des Prozessors, speziell bei blockweiser Datenübertragung
- Kommunikation ausschließlich über einen **DMA-Controller**
- DMA-Controller übernimmt Datenübertragung zwischen Speicher und externem Gerät, benötigt dabei den Systembus.
- CPU kann während der Übertragung weiterarbeiten, solange der Systembus nicht benötigt wird.

# Prozessorinterne Exceptions und Interrupts (1)

| Ereignis | ausgelöst | Bezeichnung |
|---|---|---|
| arithmetischer Überlauf | intern | Trap |
| undefinierter Opcode | intern | Trap |
| I/O-Request | extern | Interrupt |
| Hardware-Fehlfunktion | intern/extern | Trap oder Interrupt |

# Prozessorinterne Exceptions und Interrupts (2)

**Wesentlicher Unterschied**

- Prozessorinterne Exceptions sind **synchron** zum ausgeführten Programm,
- Interrupts hingegen **asynchron**

- Prozessorinterne Exceptions sind reproduzierbar, indem bei gleichem Systemstatus das Programm mit den gleichen Eingaben wiederholt gestartet wird.

# 1. Möglichkeit zur Behandlung von Exceptions

- Bei der Behandlung von Exceptions muß zunächst die **Ursache der Exception** bestimmt werden.
- **1. Möglichkeit:**
  - Es wird eine spezielle Ausnahmeroutine, die **Unterbrechungsbehandlungsroutine** oder auch **Interrupthandler** genannt, aufgerufen.
  - Rücksprungadresse und Registerinhalte werden wie schon erwähnt gesichert.
  - Bei Auftreten einer Exception wurde ein bestimmtes Bit (*Flag*) im Causeregister gesetzt. Dieses kann vom Handler abgefragt werden.
  - Der Interrupthandler springt aufgrund der festgestellten Ursache zu dem entsprechenden Code, der die Exception behandelt.
  - Wenn User-Programm nicht vom Interrupthandler abgebrochen: Rekonstruktion der Registerinhalte und Rücksprung (**RTI** (**R**eturn-**f**rom-**I**nterrupt))

# 2. Möglichkeit zur Behandlung von Exceptions

## ⌘ 2. Möglichkeit (Vectored Interrupt):

- Die Ursache der Exception wird der CPU von der Hardware (von einem externen Gerät bei Interrupt, von der CPU-Steuereinheit bei Trap) als Index einer **Interrupttabelle** bereitgestellt.
- Die Indizes werden als **Interrupt-Vektornummern** (**IVN**) bezeichnet.
- Die Tabelle liegt oftmals auf den ersten Adressen des Hauptspeichers
- In dieser **Interrupttabelle** sind die Adressen für die zu speziellen Interrupts gehörenden Routinen (**ISR**: **I**nterrupt-**S**ervice-**R**outine) gespeichert.
- Es wird die entsprechende Spezialroutine aufgerufen.

# MIPS:
# Exception-Codes im Cause-Register

- CPU verfügt über spezielle Register für die Ausnahmebehandlung

| Register | Nr. | Erläuterung |
|---|---|---|
| Status | 12 | Interrupt Maske und Enable Bits |
| Cause | 13 | Exception Type |
| EPC | 14 | Adresse des Befehls, der die Exception auslöste |

Im Falle einer Exception erfolgt ein Sprung an eine fest definierte Adresse, an der der Interrupthandler steht. Der Exception Code dient zur Feststellung der Ursache der Exception und kann nun dem Cause-Register entnommen werden (Bit 5-2).

# IBM PC Interrupt-Tabelle

| INT | Function |
|---|---|
| 0 | Timer (55 ms – Intervall) |
| 1 | Keyboard |
| 3 | COM2 and COM4 (serieller Port) |
| 4 | COM1 and COM3 (serieller Port) |
| 5 | LPT2 (paralleler Port) |
| 6 | Floppy Disk |
| 7 | LPT1 (paralleler Port) |
| 8 | Real Time Clock (CMOS Clock) |
| C | PS2 – Mausport |
| D | Numeric Coprocessor Error |
| E | IDE0 – Festplatte / CDROM / ... |
| F | IDE1 – Festplatte / CDROM / ... |

(Hier nur Indizes für (Hardware-)Interrupt angegeben!)

# Maskierbare / Nichtmaskierbare Interrupts

Prozessoren unterscheiden zwischen

**maskierbaren** und **nichtmaskierbaren Interrupts**

- **maskierbarer Interrupt**
  - wird nur dann ausgeführt, wenn das **I**nterrupt **E**nable Bit (**IE**) in einem dafür vorgesehenen Register gesetzt ist. (MIPS: Bits 8-15 im Status-Register)
  - Der Nutzer kann den Registerinhalt ändern.
- **nichtmaskierbarer Interrupt (NMI):**
  - wird auf jeden Fall ausgeführt. Bei NMI handelt es sich üblicherweise um Ausnahmesituationen, die die Funktionalität des Systems gefährden (Bsp.: “Notprogramm” bei Zusammenbruch der Betriebsspannung).

# Interrupt in Interrupt-Service-Routine

**Szenario**

Es besteht eine gewisse Wahrscheinlichkeit, daß innerhalb einer ISR (z.B. durch ein I/O-Device verursacht) ein weiteres I/O-Device seinen Interrupt auslöst!

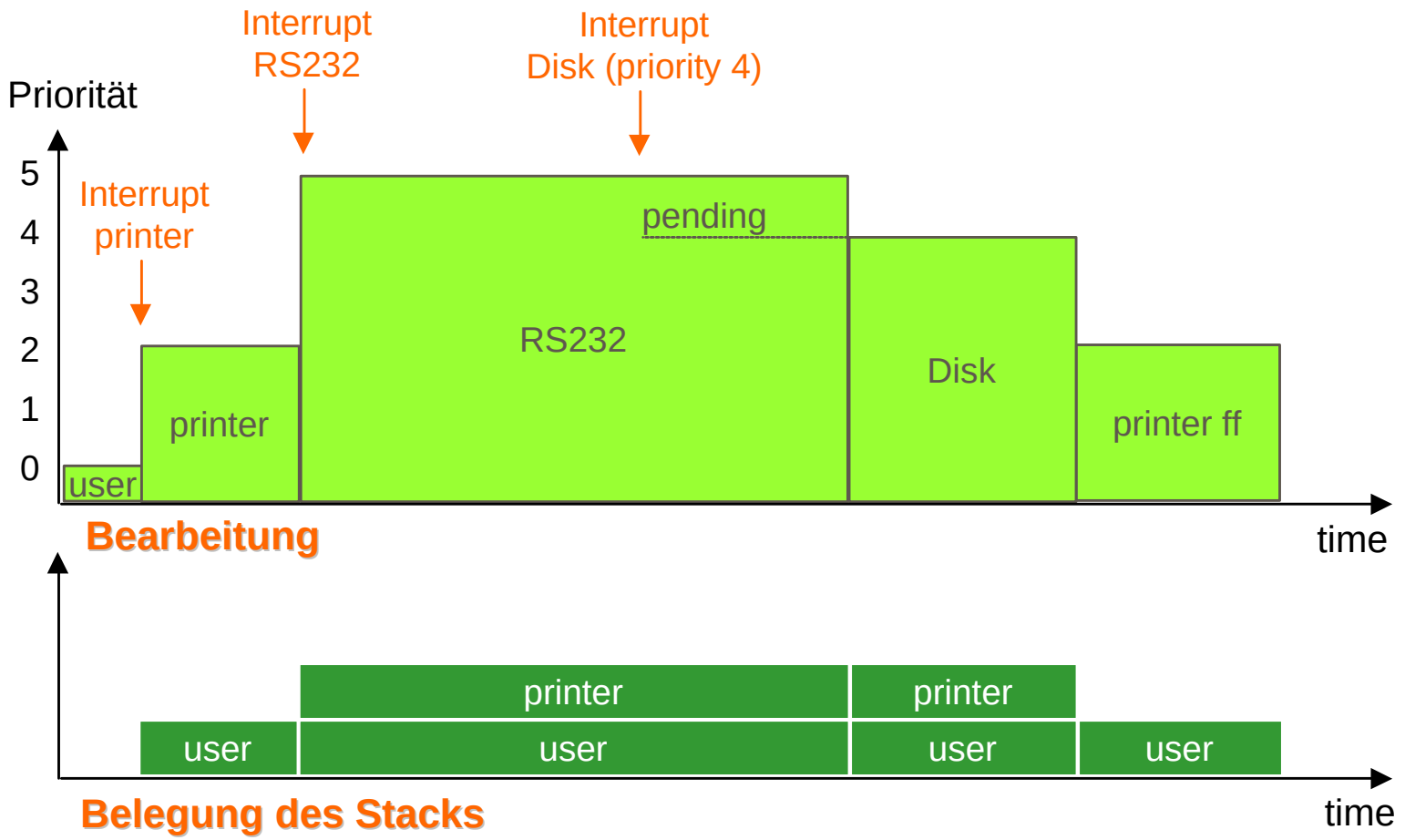
**Mögliche Lösungen**

- Erster Befehl in der ISR ist Ausmaskierung aller Interrupts
  - keine weiteren Interrupts möglich
- Vergabe von Interruptprioritäten derart, daß ein kritisches I/O-Device eine höhere Priorität, als ein weniger kritisches Device hat.

# Interrupt-Priorität

- Falls ein Device mit Priorität *n* einen Interrupt auslöst, so startet die ISR mit Priorität *n* . Wird ein Interrupt mit einer Priorität *m>n* ausgelöst, so wird die aktuelle ISR unterbrochen und eine neue ISR der Priorität *m* sofort gestartet.
- Der Versuch eines Gerätes mit einer niedrigeren Priorität als *n* die ISR zu unterbrechen, wird ignoriert und der Interruptcode wird zur späteren Ausführung zwischengespeichert (**pending Interrupt**).
- Nach Abarbeitung der ISR geht das System auf die user-Priorität (**userpriority 0**) zurück.

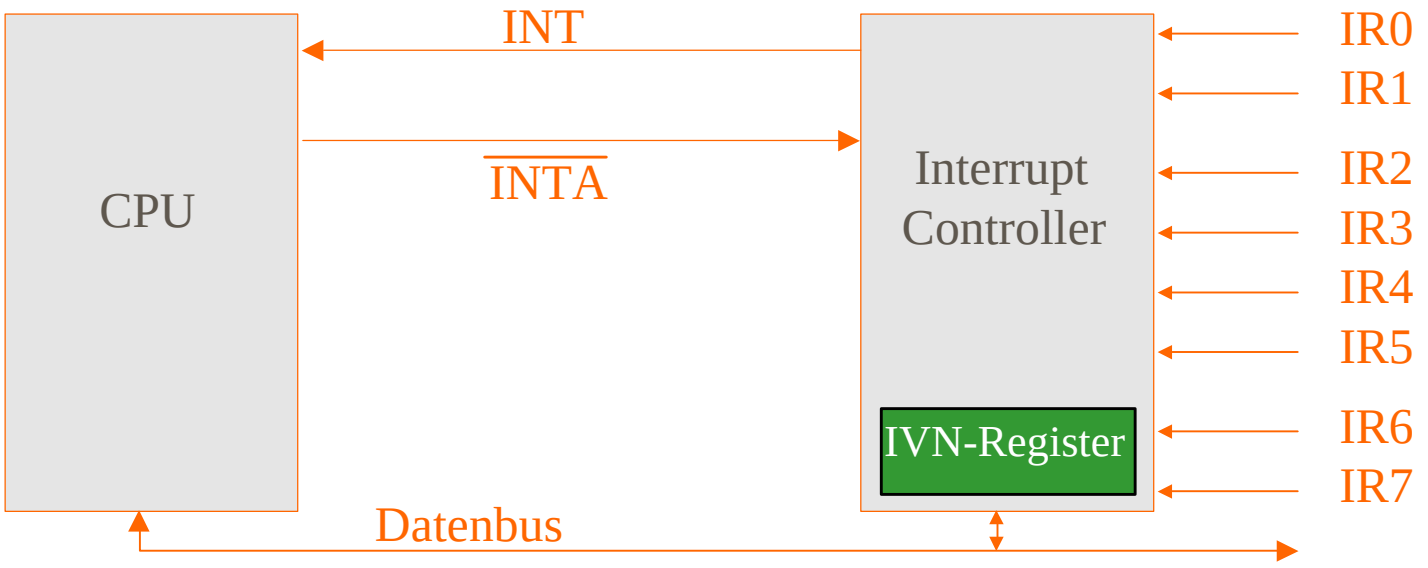
# Interrupt-Prioritäten: Illustration



# Interrupt-Prioritäten

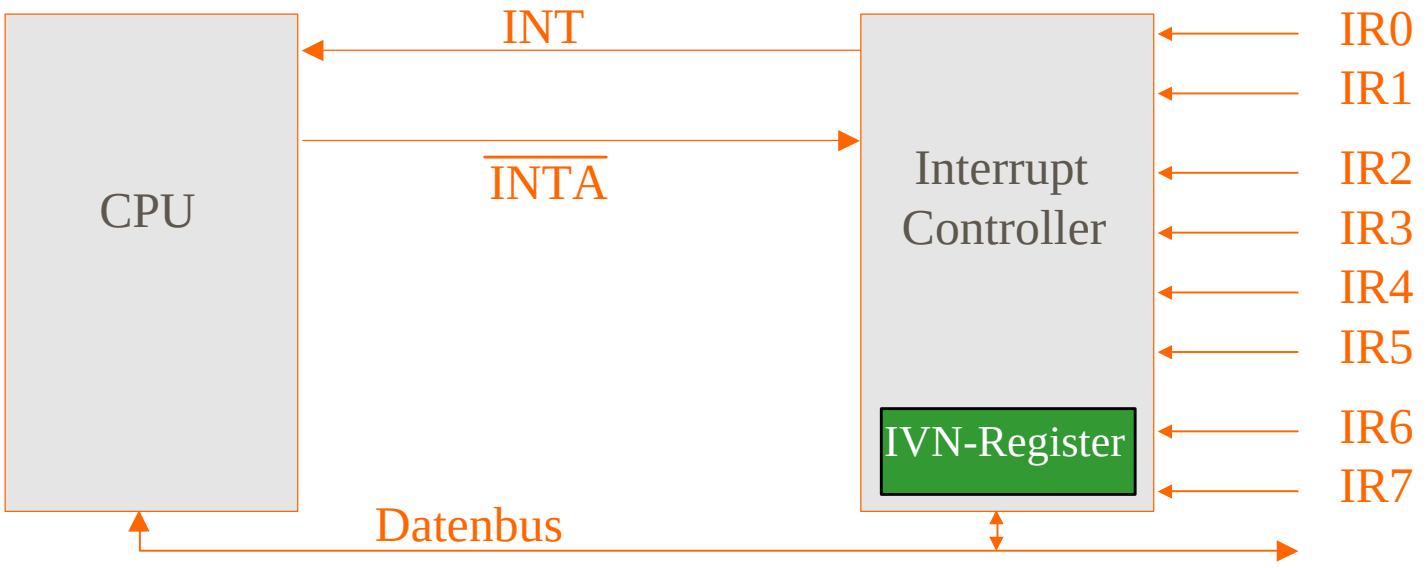
- Prozessoren mit nur einem **Interruptlevel** (x86) ist es selbst nicht möglich, verschiedenen Geräten verschiedene Prioritäten zuzuordnen.
- Abhilfe schafft in diesem Fall ein **Interrupt-Controller**, der extern mit dem Prozessor verbunden wird (z.B. 8259A).
- Dieser externe Controller verwaltet die Prioritäten. Für den Fall dass keine ISR aktiv ist oder ein Interrupt eine laufende ISR unterbrechen darf, signalisiert er dem Prozessor einen Interrupt, falls ein solcher bei ihm anliegt
- Da er vor dem Auslösen eines weiteren Interrupts mit niedriger Priorität wissen muss, ob die ISR beendet wurde, benötigt er eine Rückmeldung vom Prozessor.

# Interrupt-Controller



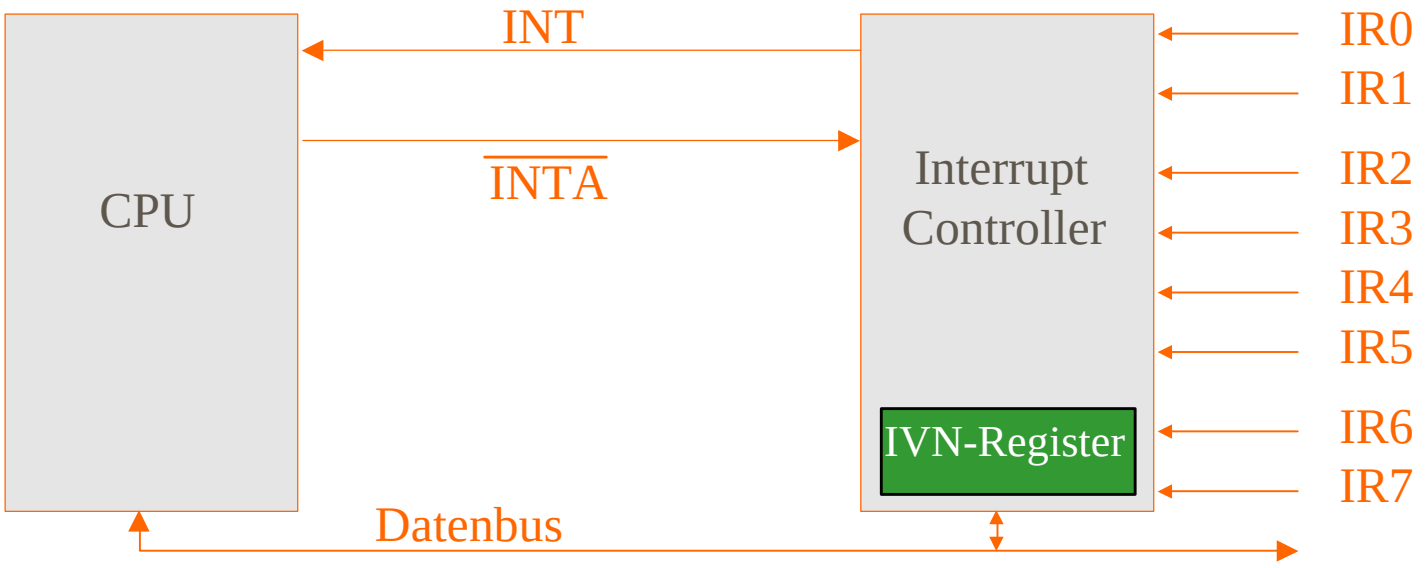

- I/O-Device *j* teilt dem Controller durch Setzen der Signalleitung IR*j* mit, daß eine Ausnahmesituation vorliegt.
- Die Kennung des Interrupts wird in das Register **IVN** eingetragen.
- Der Controller gibt die Meldung durch Setzen des Signals INT an die CPU weiter.
- Ist die CPU bereit, so teilt sie dies dem Controller durch Setzen des Signals INTA mit.
- Der Interrupt-Controller legt den Inhalt seines IVN-Registers auf den Datenbus.

# Interrupt-Controller ff




- Der Interrupthandler der CPU nimmt den Inhalt des IVN-Registers vom Bus und rettet ihn in einem eigenen Register. Hiermit weiß der Prozessor, welches periphere Gerät den Interrupt ausgelöst hat.
- Vor der Abarbeitung des Interrupts rettet die CPU den Befehlszähler und Statusregister auf dem Stack.
- Die CPU schlägt in der Interruptvektortabelle nach, um mit Hilfe der IVN die Startadresse der entsprechenden Interrupt-Serviceroutine (ISR) zu finden.

# Interrupt-Controller ff




- Die CPU führt die ISR aus. Falls sie ISR Register verwendet, so muß sie deren Inhalt vorher auf dem Stack sichern.
- Vor Beendigung der ISR holt diese die geretteten Registerinhalte vom Stack.
- Die CPU stellt den ehemaligen Zustand (Befehlszähler und Statusregister) wieder her.
- Dem Interrupt-Controller wird die Abarbeitung des Interrupts mitgeteilt.