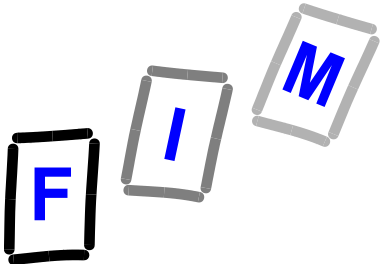
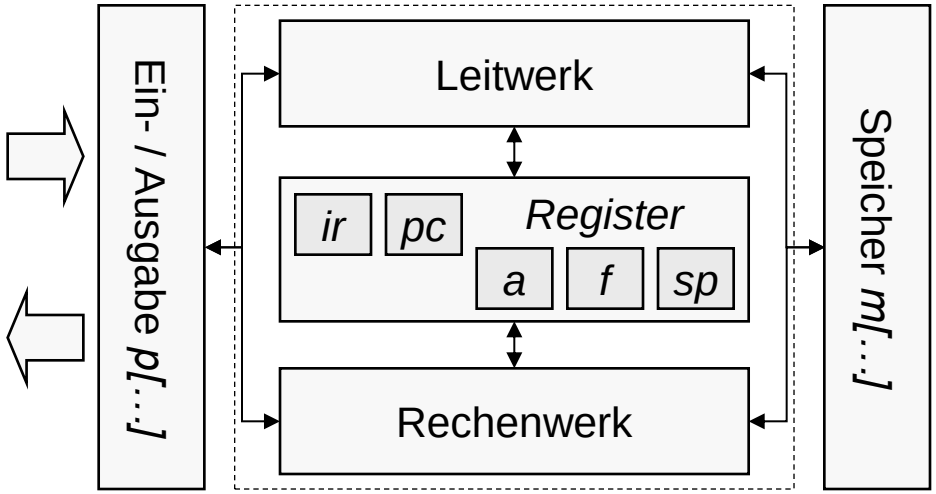


Betriebssysteme

Kap B: Hardwaremechanismen

Beispielprozessor

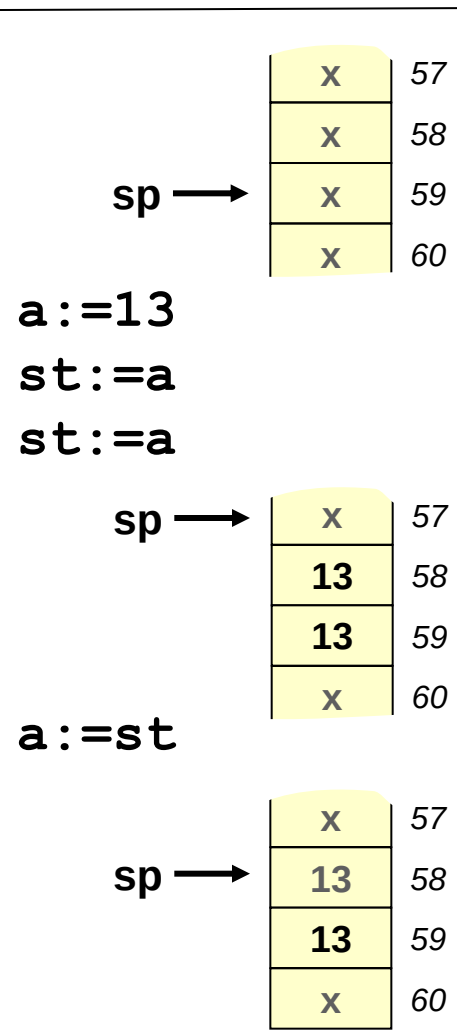


Spezielle Register

- **Flagregister *f***
  - ➔ ***f.i*: Interrupt-Enable-Flag**
  - ➔ ***f.z*: Zero-Flag**

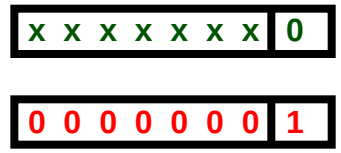
- **Stack *st***
  - ➔ **Lage durch den Stackpointer *sp* beschrieben**
  - ➔ **Einfügen: *sp* wird dekrementiert**
    Löschen: *sp* wird inkrementiert
  - ➔ **Inkrement/Dekrement implizit und damit in der Notation unsichtbar**
  - ➔ **Beispiele:**

PUSH	`st:=a`	`m[sp]:=a;sp:=sp-1`
POP	`a:=st`	`sp:=sp+1;a:=m[sp]`



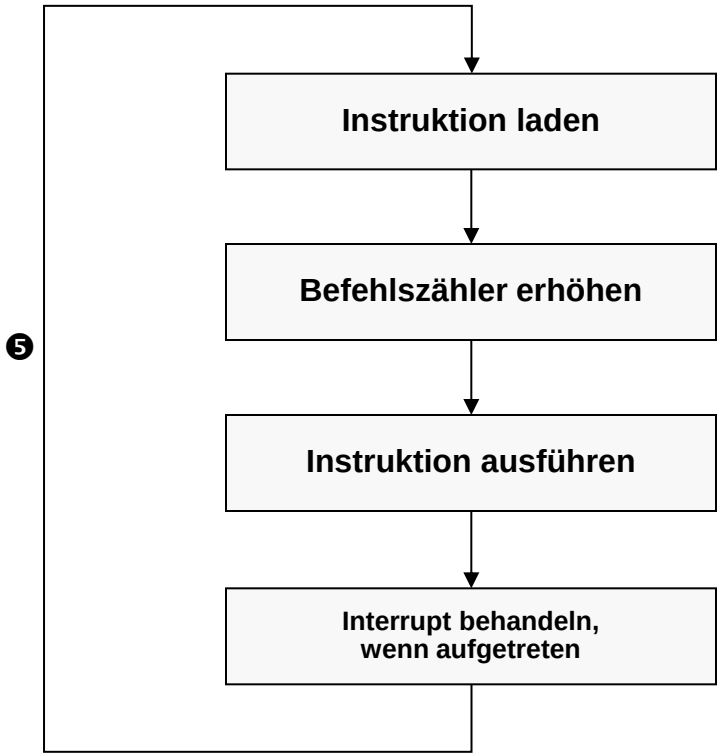
Code B-1 Beispiel für Assembler Pseudocode erste Instruktion auf Adresse 100

100	a:=p[0]	// Inhalt von Port 0 nach a
101	f.z:=((a&1)==0)	// wenn least signifikant Bit 0 von a gleich 0, dann setze Zero-Flag f.z auf 1, ansonsten f.z:=0
102	If (f.z) pc:=100	// wenn f.z gesetzt ist, d.h. f.z=1, dann führe die nebenstehende Operation aus; pc := 100 entspricht ?????
103	m[5]:=a	// Inhalt von a nach Speicherstelle 5, d.h. nach m[5]



a&1 ist false (0), daher a&1==0 → true (1)

Von-Neumann-Zyklus (John von Neumann 1903-1957)



❶ *Befehl einlesen*

❷ *pc auf nächsten Befehl*

❸ *Befehl ausführen*

❹ *Interrupt-Behandlung*

Moderne Hardware benutzt zB. „Pipelining“ um Instruktionen in kleinere Teile zu unterteilen und damit versetzt parallel auszuführen, oder mehrere Instruktionen werden physisch parallel ausgeführt (mehrere Recheneinheiten)

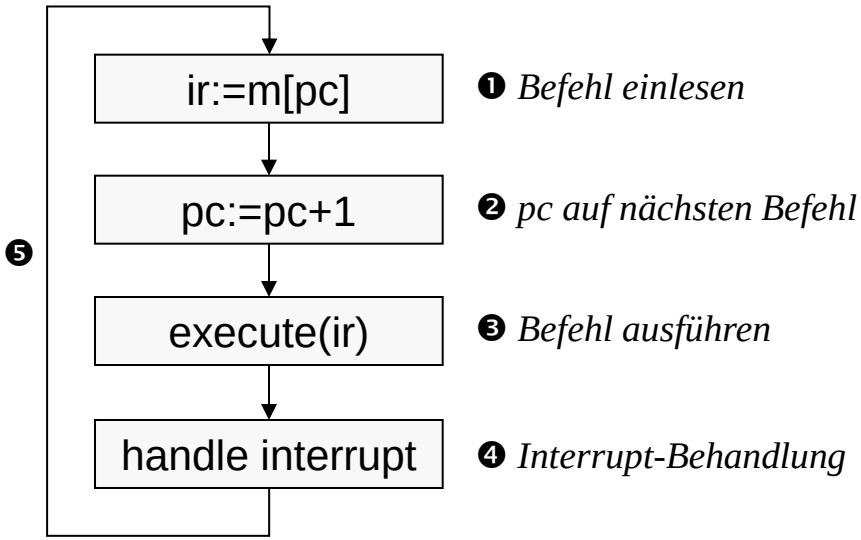
Wichtige Gemeinsamkeit von Optimierungen:
Es muss immer exakt dasselbe herauskommen, wie wenn dieses einfache Modell ausgeführt worden wäre

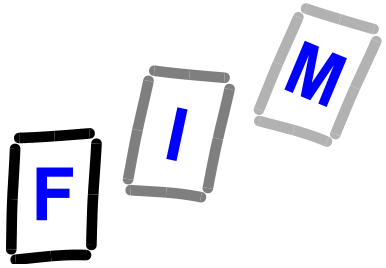
Und genau darin liegt ein großer Aufwand für die Optimierungen!

Von-Neumann-Zyklus in Pseudocode

Anfangswert von pc ist maschinenabhängig, zB.: pc:=100

Wichtig: Unmittelbar nach dem Einlesen der Instruktion (Befehl) wird pc erhöht und erst nachher die Instruktion ausgeführt!





Laden des OS nach dem Einschalten des Computers

- **Beachte: Das OS (BS) ist selbst ein Programm**

- **Frage 1: Wie erreicht man, dass ein bestimmtes Programm ausgeführt wird ?**
  - ➔ **Z.B. unmittelbar nach dem Einschalten:**
    Nach der Basis-Initialisierung der Hardware
    POST (Power On Self Test)
    ein Programm, das persistent im BIOS gespeichert ist

- **Frage 2: Wie wird das auf dem Massenspeicher (Disk) gespeicherte OS geladen und zur Ausführung gebracht ?**
  - ➔ **„BS–Boot“ : Starten des BS-Ladeprogrammes**

Basis-“Trick“

- **Nach dem Einschalten beginn der „von Neumann“-Zyklus zu laufen**

- **Hardwaremäßig ist der pc (Program Counter) auf die erste ausführbare Instruktion jenes Programmes zu setzen, das als erstes auszuführen ist**
  - ➔ **Konsequenz:**
    - » **Zunächst pc:=Adresse der ersten Instruktion der Befehlsfolge, die für die Hardware-Initialisierung zuständig ist**
    - » **Nach der Hardwareinitialisierung – als letzter Befehl derselben – wird der pc auf die erste ausführbare Instruktion (z.B.) des Programmes „POST“ gesetzt**
  - ➔ **Welches diese Adresse ist → Siehe CPU-Dokumentation!**

Beispiel: ARM-Core des Raspberry Pi

Reset

When Reset is de-asserted:

```
/* Stay in secure state */
R14_svc = UNPREDICTABLE value
SPSR_svc = UNPREDICTABLE value
CPSR [4:0] = 0b10011 /* Enter supervisor mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [6] = 1 /* Disable fast interrupts */
CPSR [7] = 1 /* Disable interrupts */
CPSR [8] = 1 /* Disable imprecise aborts */
CPSR [9] = Secure EE-bit /* store value of Secure Control Register bit[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0000
else
    PC = 0x00000000
```

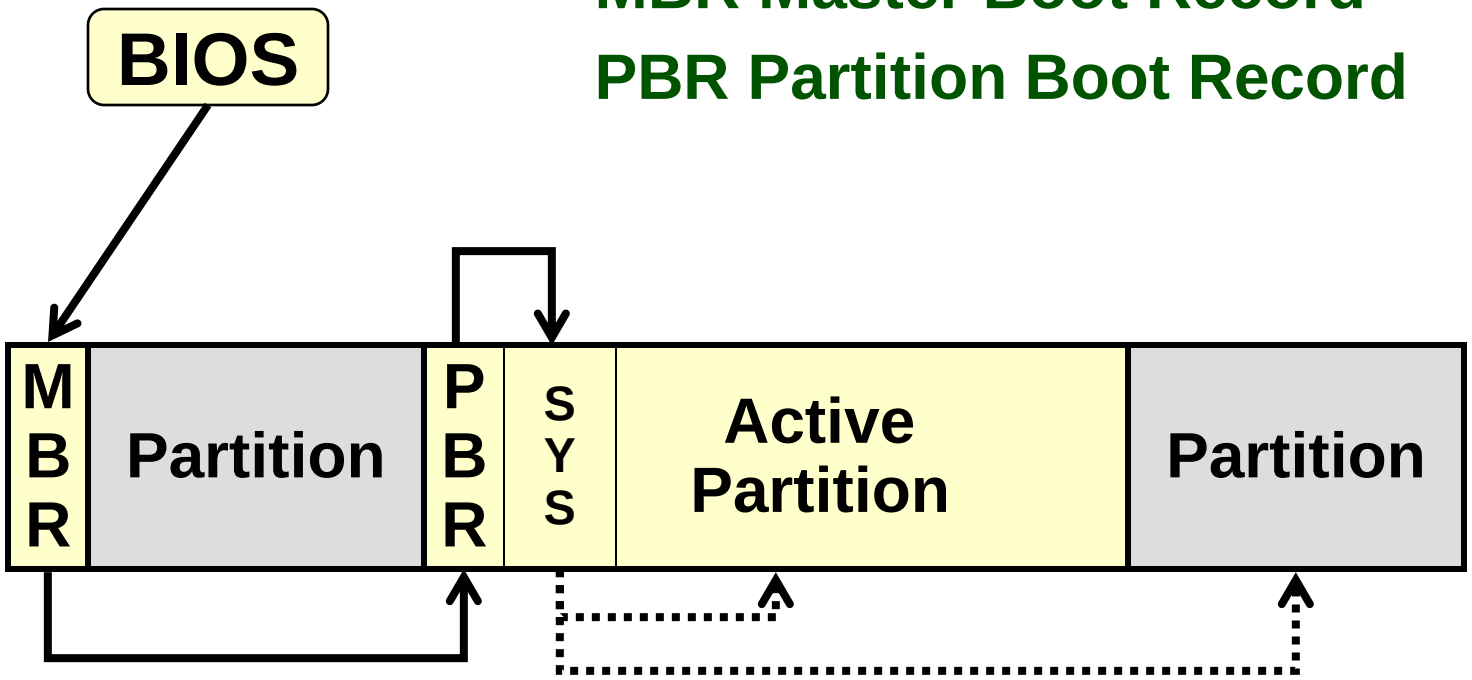
- **Je nach Konfiguration (CPU-Kern extern!) beginnt die Ausführung nach einem Reset bei 0x00000000 oder 0xFFFF0000**

Quelle: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf

Disk-Partition und OS-BOOT

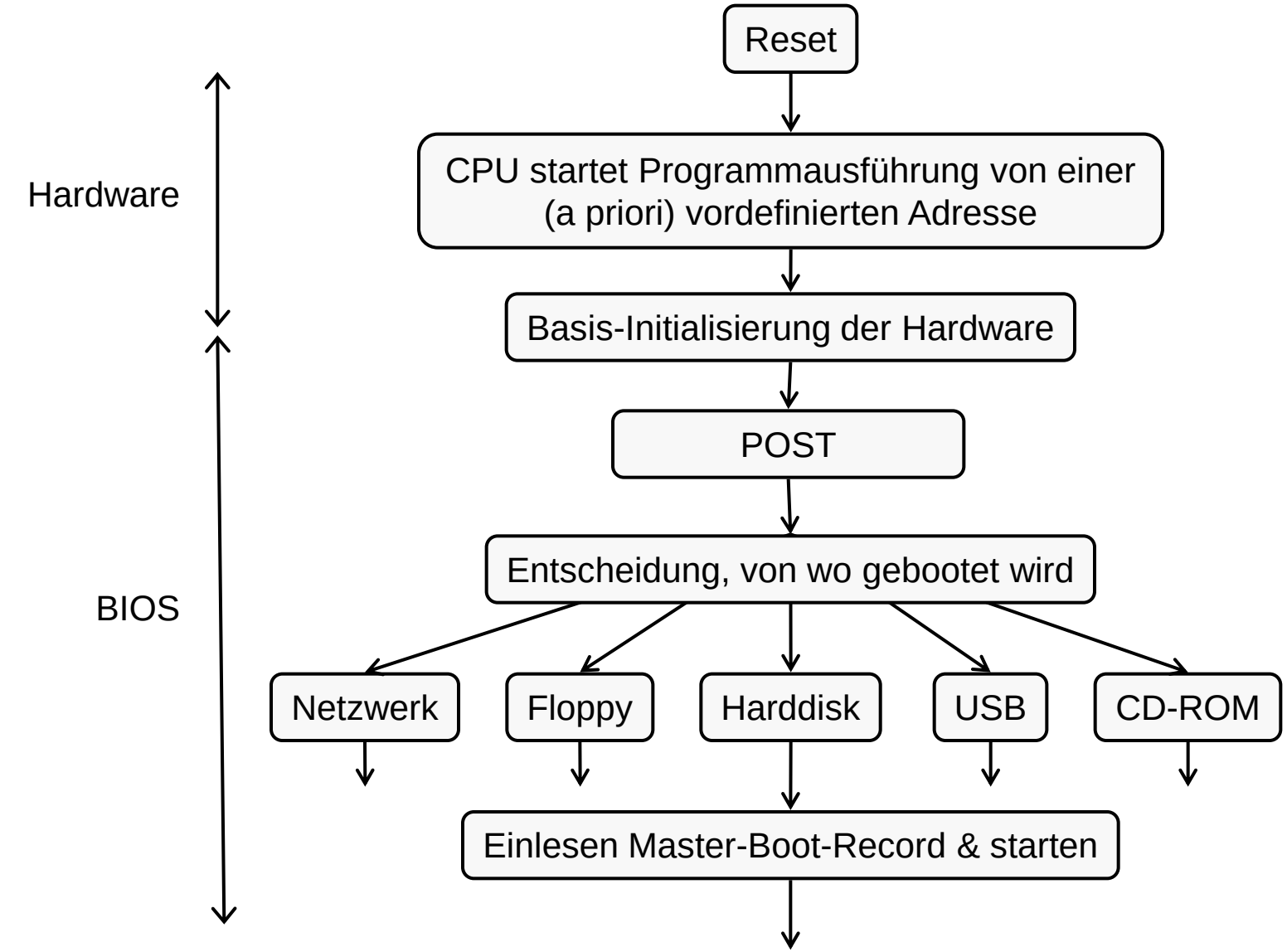
MBR Master Boot Record

PBR Partition Boot Record

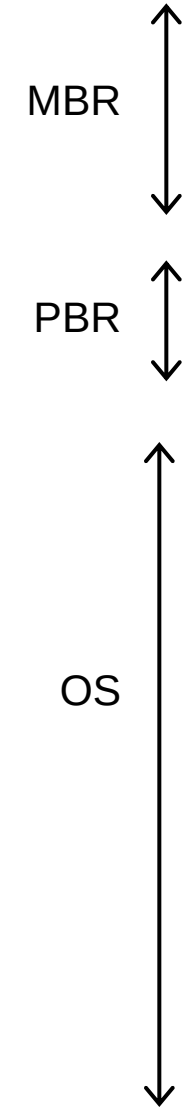


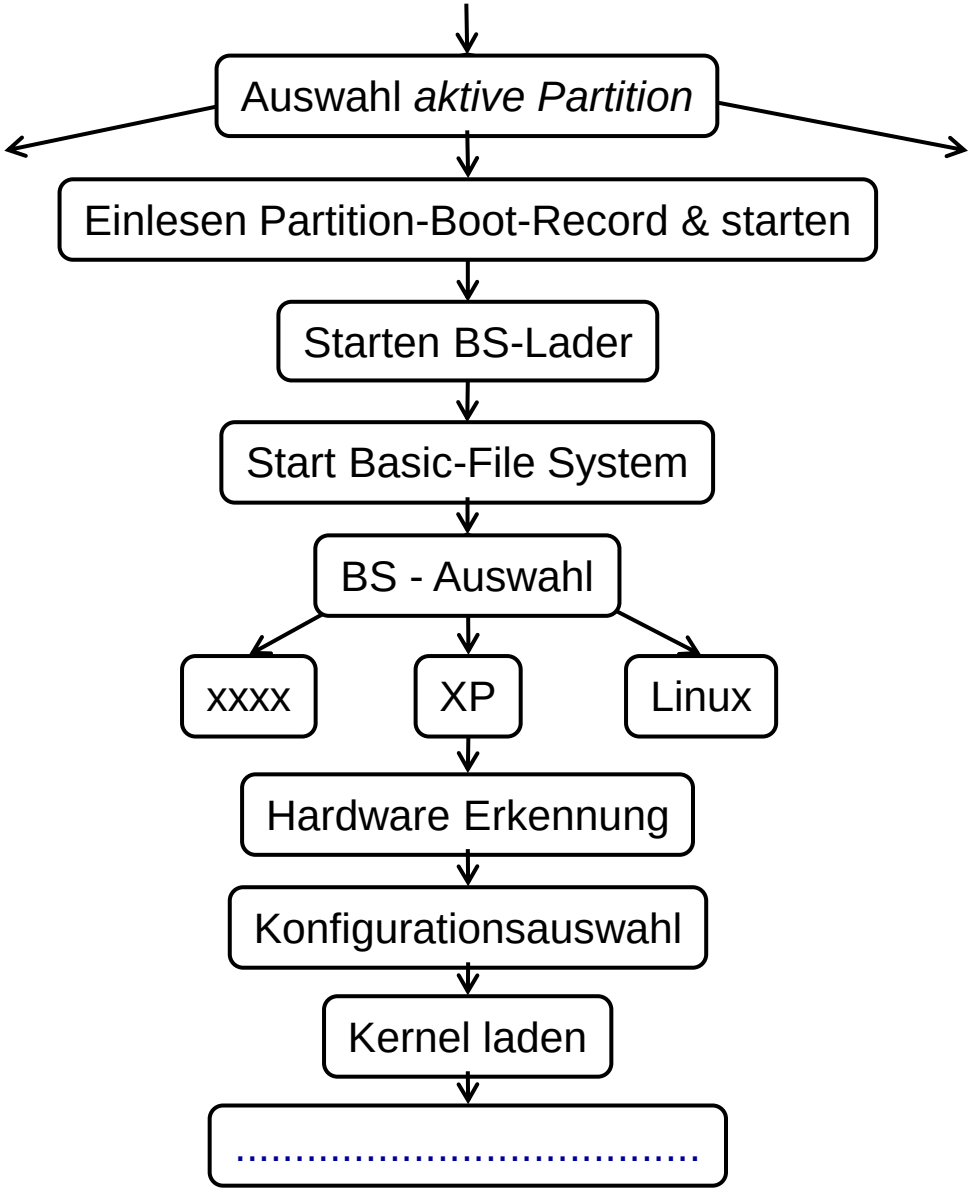
(Nur schematisch)

OS BOOT [1]


Hardware
BIOS
Reset
CPU startet Programmausführung von einer (a priori) vordefinierten Adresse
Basis-Initialisierung der Hardware
POST
Entscheidung, von wo gebootet wird
Netzwerk
Floppy
Harddisk
USB
CD-ROM
Einlesen Master-Boot-Record & starten

OS BOOT [2]


MBR
PBR
OS


Auswahl *aktive Partition*
Einlesen Partition-Boot-Record & starten
Starten BS-Lader
Start Basic-File System
BS - Auswahl
xxxx
XP
Linux
Hardware Erkennung
Konfigurationsauswahl
Kernel laden
.....

OS BOOT [3]

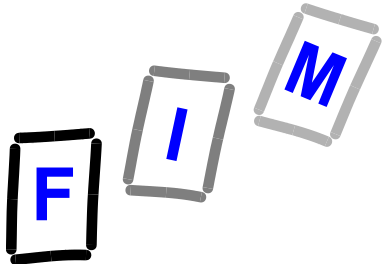
- **Nachdem der „Kern“ (kernel) des BS geladen ist und**
  - ➔ **weitere „anfangs benötigte“ Komponenten des BS geladen worden sind („Dienste“, „Services“, …)**
  - ➔ **Verzweigung zum**
    - » **Kommando –Interpreter**

      Eingabe der Kommandos (Aufruf von Programmen) über Kommandozeilen / Namen der Programme

    oder

    - » **GUI (Graphical User Interface)**

      Eingabe der Kommandos durch Mausklick auf Icons, die für die betreffenden Kommandos stehen (oder Touchscreen; äquivalent)



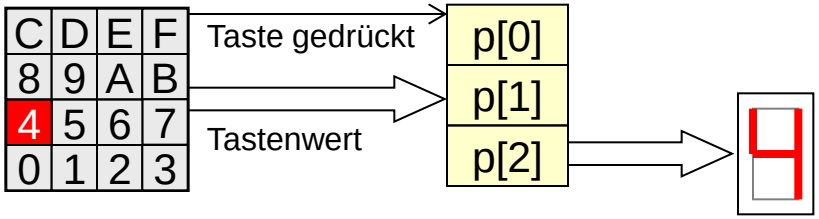
Elementare Mechanismen für Ein/Ausgabe (I/O)

- **Zwei Methoden**
  - ➔ **Polling**
  - ➔ **Interrupt getrieben**

Ein- / Ausgabe

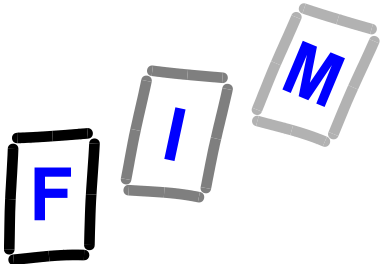
Eingabe

- ➔ Tastatur
- ➔ Schalter
- ➔ Kamera
- ➔ Mikrofon
- ➔ Netzwerk
- ➔ Maus
- ➔ Touchscreen
- ➔ …



Ausgabe

- ➔ LED
- ➔ Relay
- ➔ Bildschirm
- ➔ Sound
- ➔ Netzwerk
- ➔ Braille-Zeile
- ➔ …



☹ Polling ☹

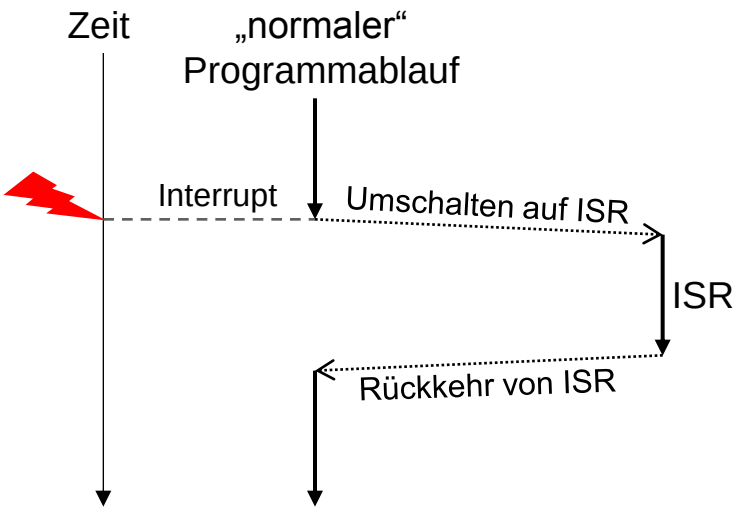
```
// warten, bis Daten vorhanden sind
while (p[0]&1==0) { /* leer!! */ };
// jetzt sind Daten vorhanden: Diese Ausgeben
p[2]:=p[1];
```

```
100   a:=p[0]              // p[0] auslesen und
101   f.z:=(a&1==0)        // testen, ob Bit 0 gesetzt ist
102   if(f.z) pc:=100      // solange testen, bis Daten verfügbar,
                           //  also p[0].0==1
103   a:=p[1]              // Aktion nach Verfügbarkeit der Daten
104   p[2]:=a              // ...
```

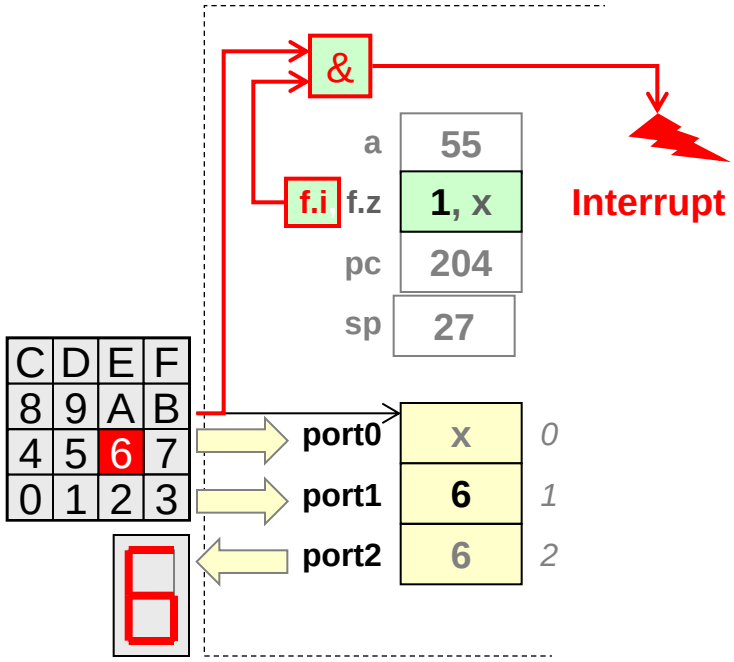
Interrupts

- **Interrupts …**
  - ➔ **unterbrechen das eben laufende Programm**
  - ➔ **werden durch ein externes Interrupt-Signal ausgelöst**
- **Funktion in der Interrupts-Service-Routine (ISR)**

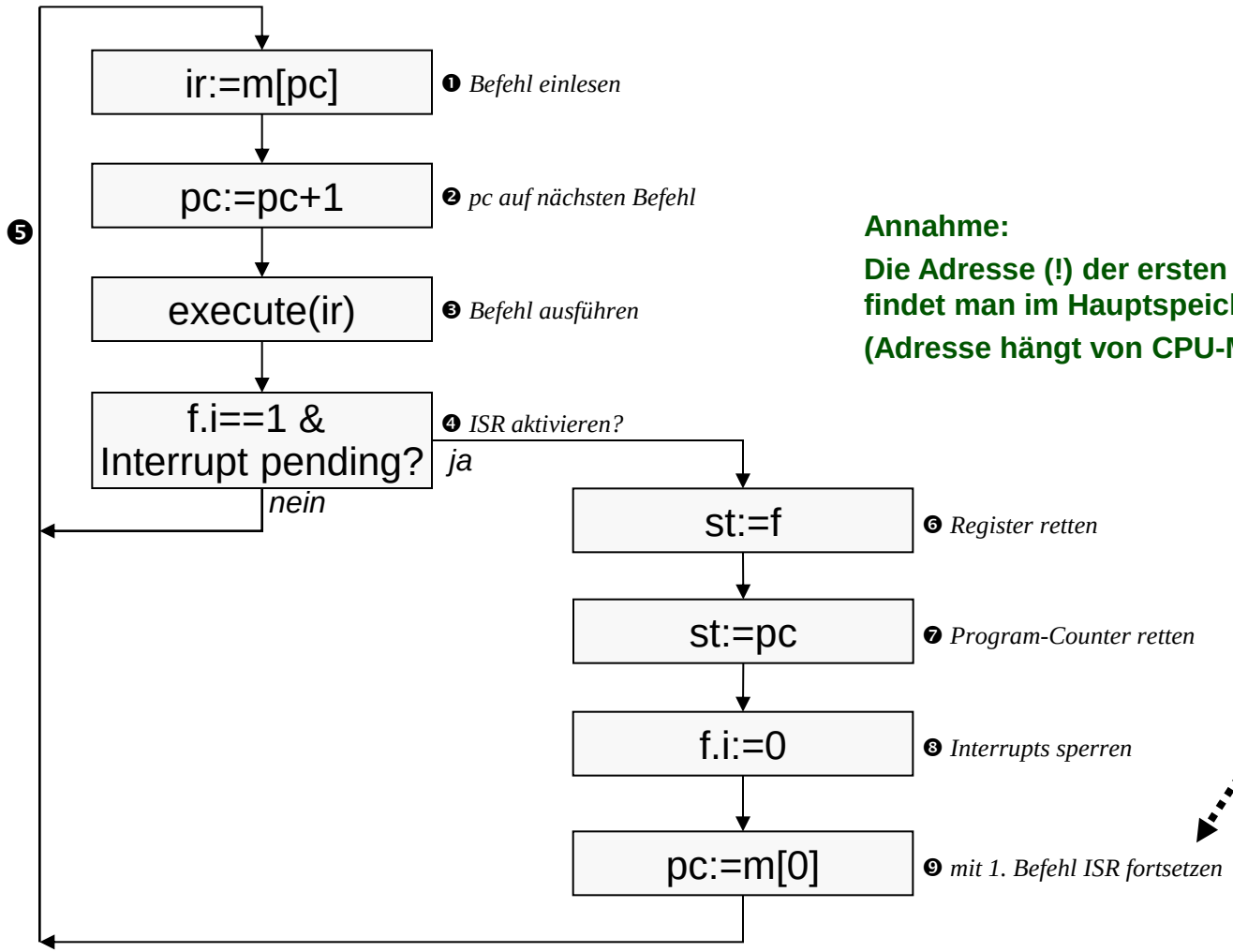
- **Zeitlicher Ablauf:**



Externes Interrupt-Signal



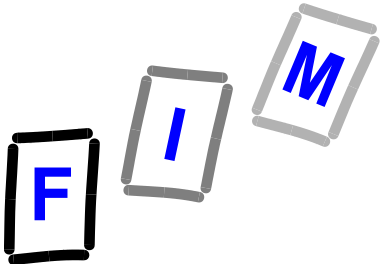
Von-Neumann-Zyklus und Interrupts



Annahme:

Die Adresse (!) der ersten Instruktion der IR findet man im Hauptspeicher an der Stelle m[0]

(Adresse hängt von CPU-Modell/Konfiguration ab)



Rückkehr aus der IR

- **Folgende Schritte sind notwendig**

  ➔ **Beachte: Die benötigten Daten wurden vorher auf den Stack gerettet:**

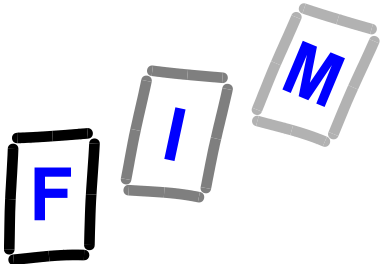
  - » **Die Zustands-Register der CPU**
    - – Im einfachen Beispiel nur das Flag Register f, sowie
    - – der Wert des Program-Counter (PC) zum Zeitpunkt der Unterbrechung

  ➔ **Diese Daten müssen von dem Stack geholt werden**

  ➔ **Die betroffenen Register werden damit auf den Wert zurück-gesetzt, den sie zum Zeitpunkt der Unterbrechung hatten**

  - » **Insbesondere: PC**
  - » **Vor dem Verlassen: f.i =1 (Interrupt Sperre aufheben)**

    Wird durch spezielle Instruktion reti (teilweise) erledigt



Software-Interrupt

- **Interrupts sind asynchrone Ereignisse, die zu unbekanntem Zeitpunkt auftreten**
- **Softwaremäßige Nachbildung**
  - ➔ **SWI: Sprung zur Interrupt-Service Routine**
  - ➔ **Wenn mehrer Serviceroutinen zur Auswahl:**
    - » **SWI[j] mit „Nummer“ j der Service Routine**
  - ➔ **Aktivierung damit deterministisch**
    - » **Vgl.: Subroutine Call**
    - » **Aber: Zustand (Flag-Register werden gerettet)**
    - » **Rücksprung-Adresse detto**
  - ➔ **Zweck: zB „Umschalten“ von Programm zu Betriebssystem**
    - » **Aufruf von Betriebssystem-Funktionen**
    - » **Je nach Hardware auch andere Spezialinstruktionen!**