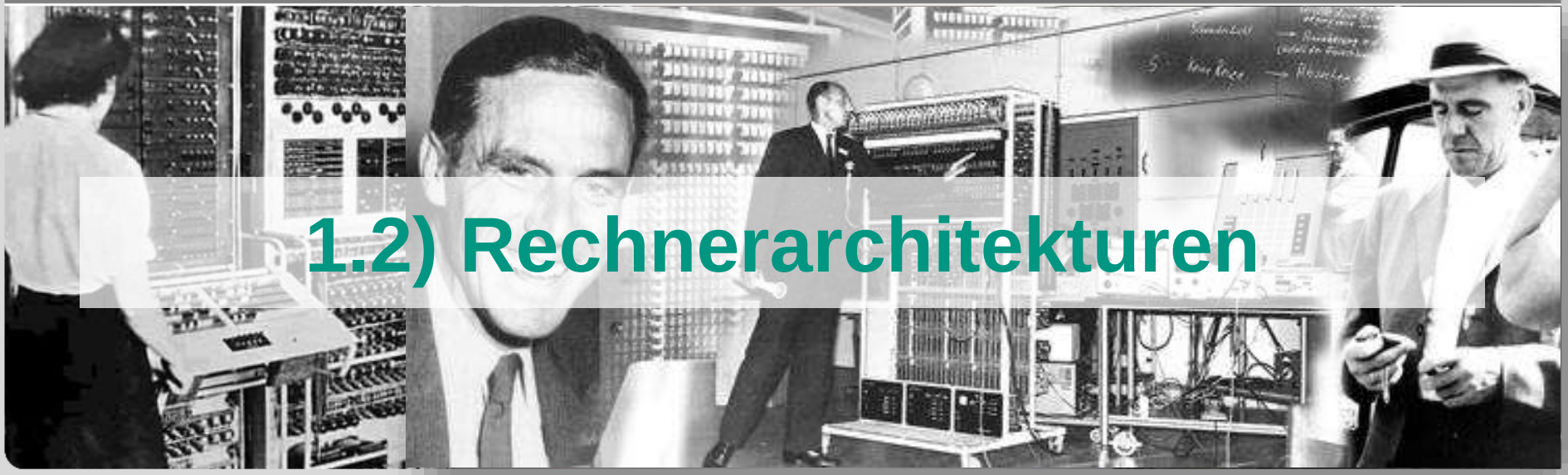
Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Vorlesung Informationstechnik (IT)

Sommersemester 2018



1.2) Rechnerarchitekturen

IT Inhalt

0. Einleitung und Motivation

- Organisatorisches
- Begriffe
- Typische Anwendungen

1. Rechnerarchitekturen

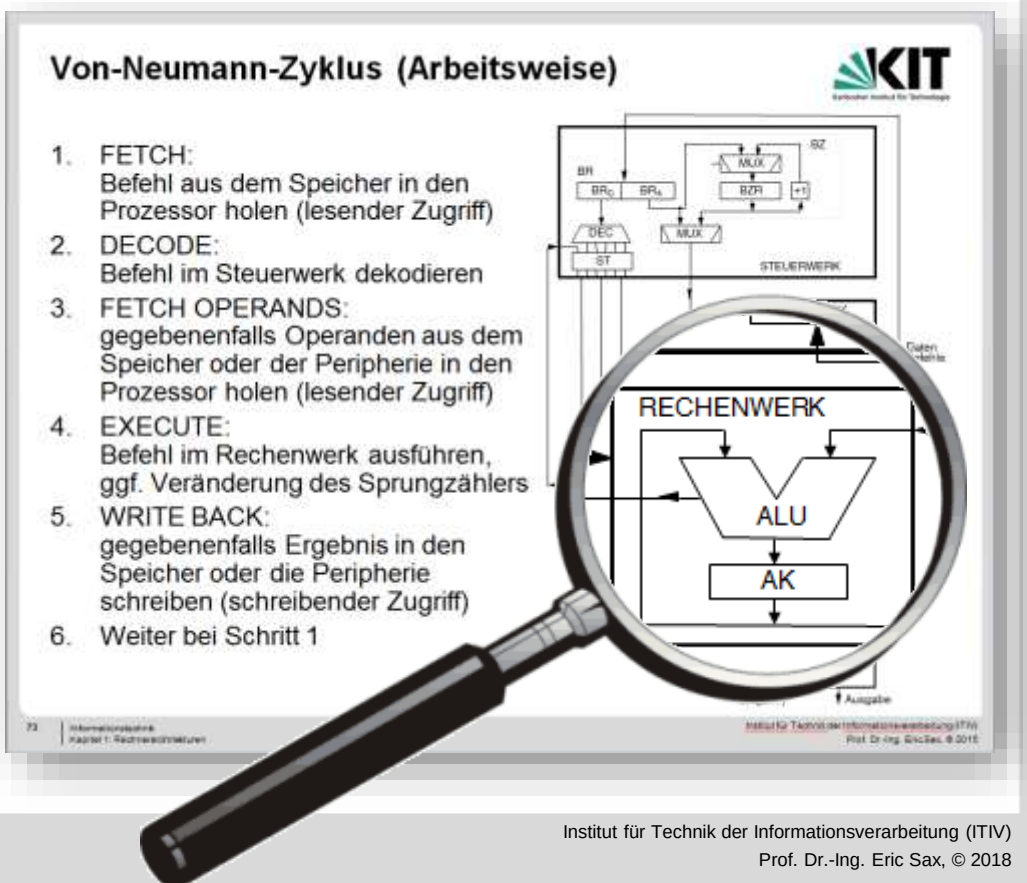
- Einführung
- Allgemeiner Aufbau der internen Hardware Architektur
- Instruction Set Architecture (ISA) am Beispiel AVR8
- Klassifikation von Rechnerarchitekturen
  - Akkumulator-Maschine
  - Stack-Maschine
  - Registersatzmaschine
    - CISC-Architektur
    - RISC-Architektur
- Performanzsteigerung
- ausgewählte Beispiele

Akkumulatormaschine

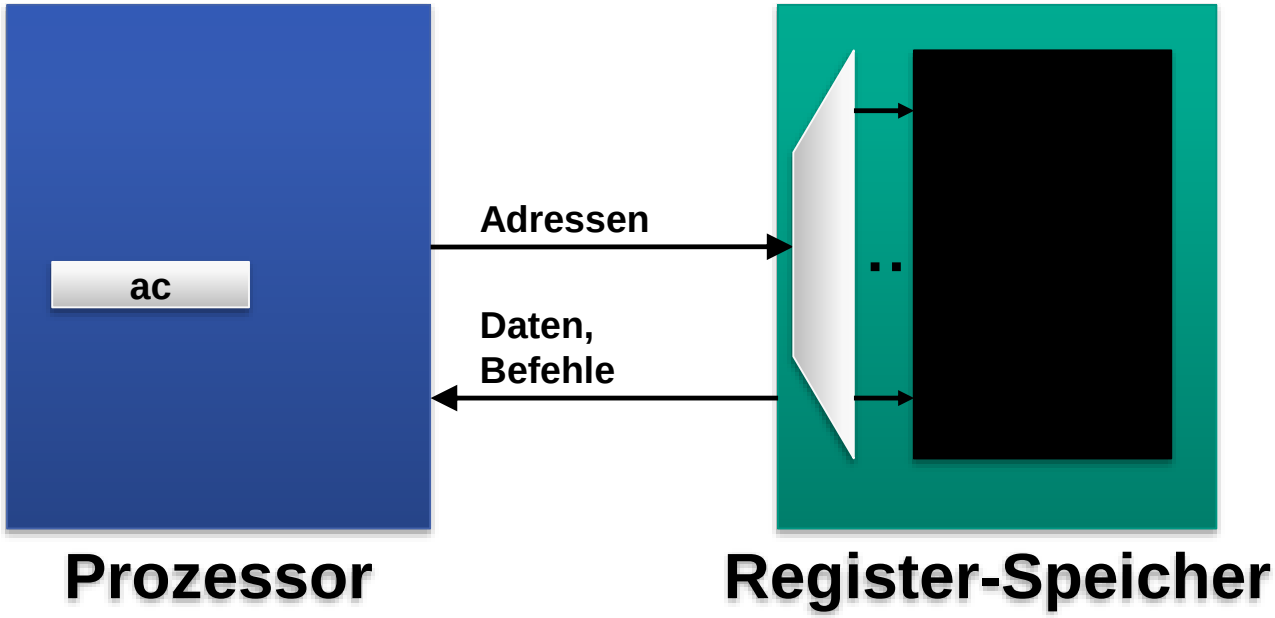
- Die Rechenergebnisse werden in einem speziellen Register, dem Akkumulator (AC oder AK), „akkumuliert“
  - Akkumulator in zentraler Rolle
  - Organisationsprinzip vieler einfacher Rechner
- Typische Befehle:
  - LAC m: lade Wert unter Adresse m in den AC
  - ADD m: addiere den Inhalt von AC mit dem Wert unter Adresse m und speichere das Resultat nach AC
  - SAC m: speichere den Inhalt

Beispiel: *c := a+b*

LAC a
ADD b
SAC c

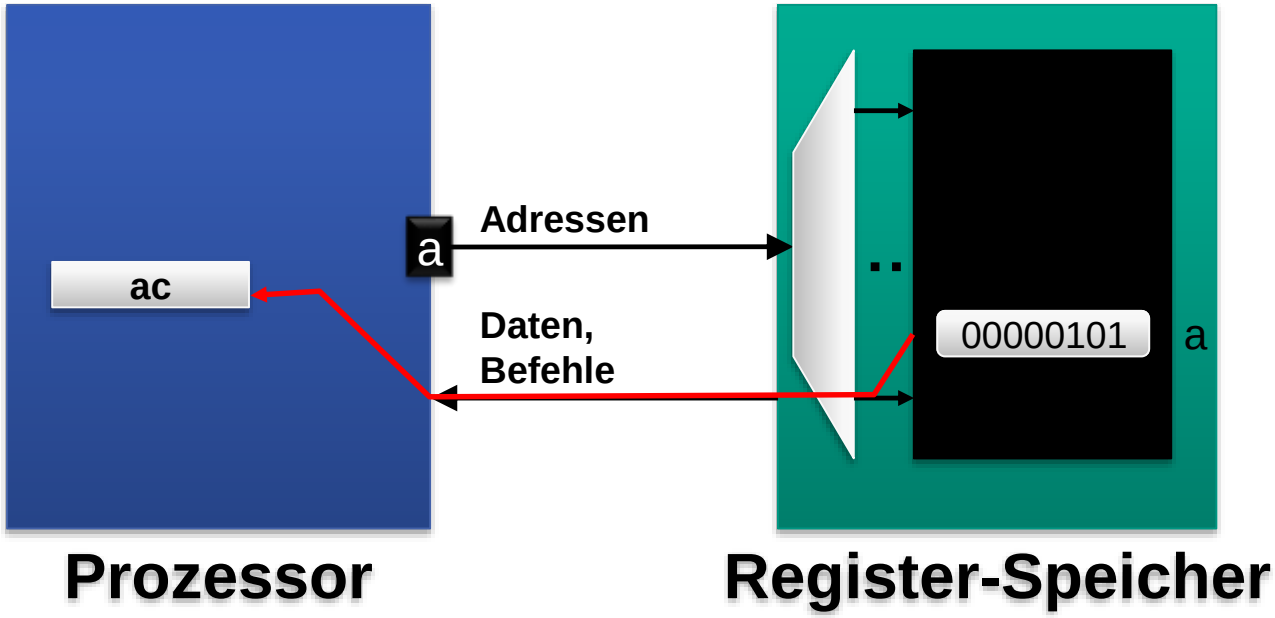


Struktur der Akkumulatormaschine



- Prozessor beinhaltet den Akkumulator ac
- Speicher beinhaltet Adressmultiplexer und die Register

Struktur der Akkumulatormaschine

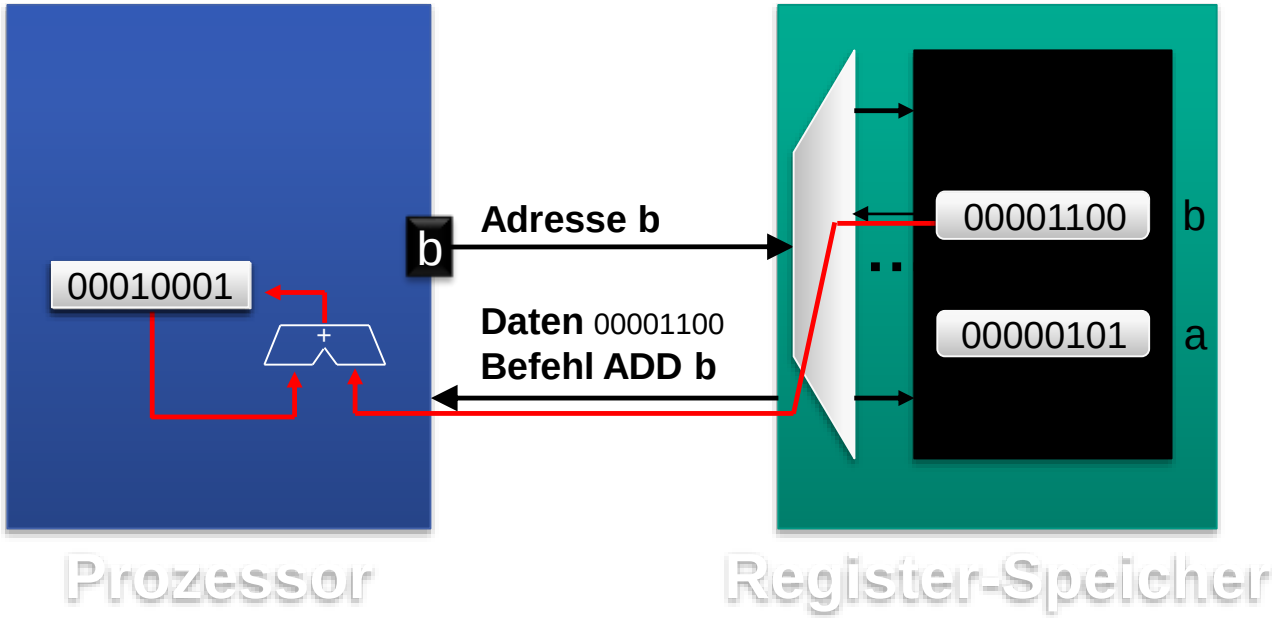


Beispiel: *c := a + b*

LAC a
ADD b
SAC c

- Daten aus Register a in den Akkumulator laden

Befehlsausführung

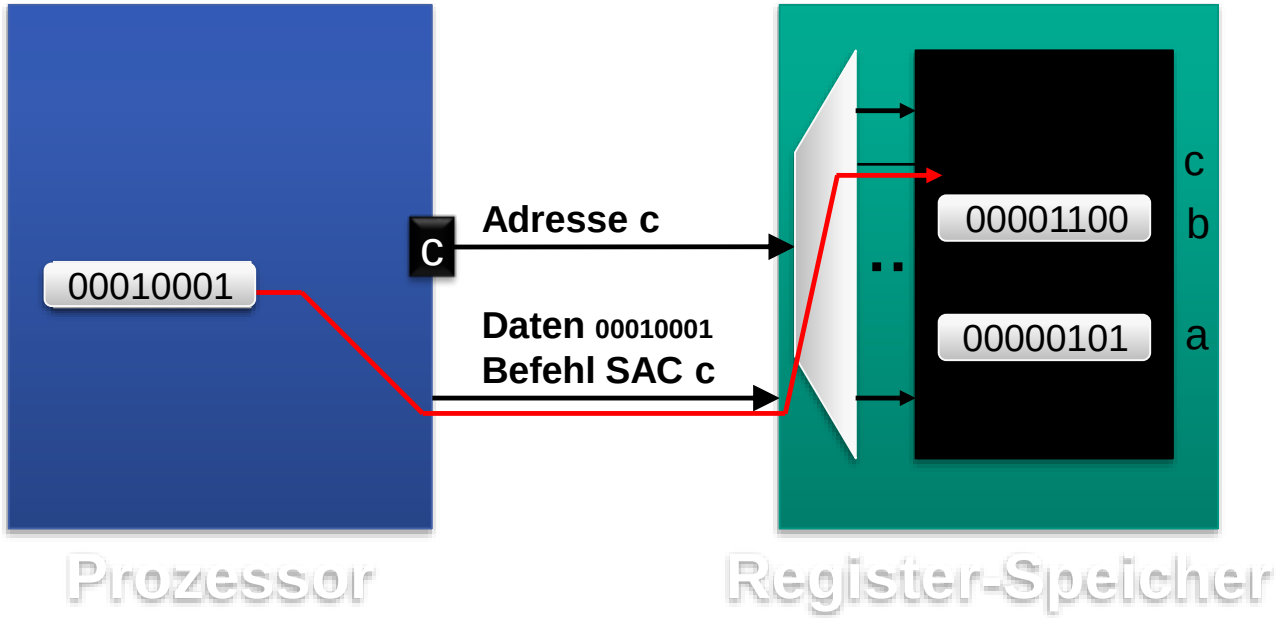


Beispiel: *c := a + b*

LAC a
ADD b
SAC c

- Inhalt des Registers b zum Inhalt des Akkumulators addieren
- Ergebnis in Akkumulator ablegen

Befehlsausführung

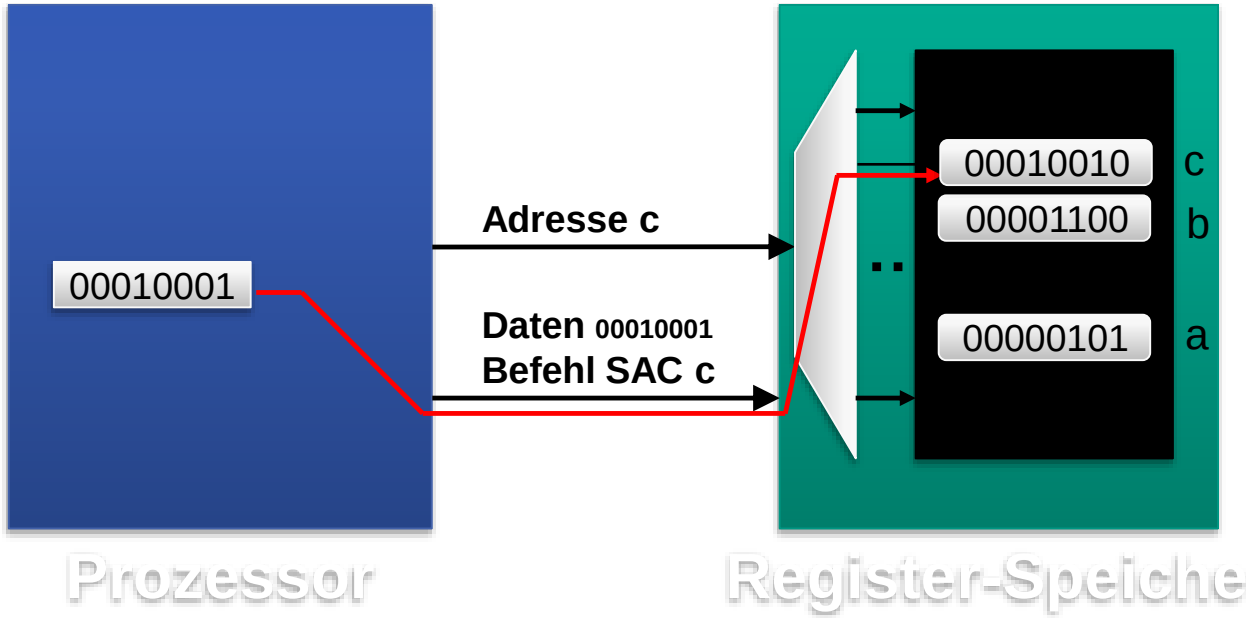


Beispiel: *c := a + b*

LAC a
ADD b
SAC c

- Daten aus Akkumulator in das Register c schreiben

Befehlsausführung



Beispiel: *c := a + b*

LAC a
ADD b
SAC c

- Daten aus Akkumulator in das Register c schreiben

Stackmaschine

- Befehle beziehen sich auf einen Stack (Stapelspeicher) zur Speicherung u. a. von Zwischenergebnissen

- Typische Befehle:
  - PUSH m: lege den Wert unter der Adresse m auf den Stack
  - ADD: addiere die beiden obersten Werte des Stacks, entferne sie und lege die Summe als oberstes Elemente auf den Stack
  - STORE m: speichere das oberste Element auf dem Stack nach der Adresse m

Beispiel: c := a+b

```
PUSH a
PUSH b
ADD
STORE c
```

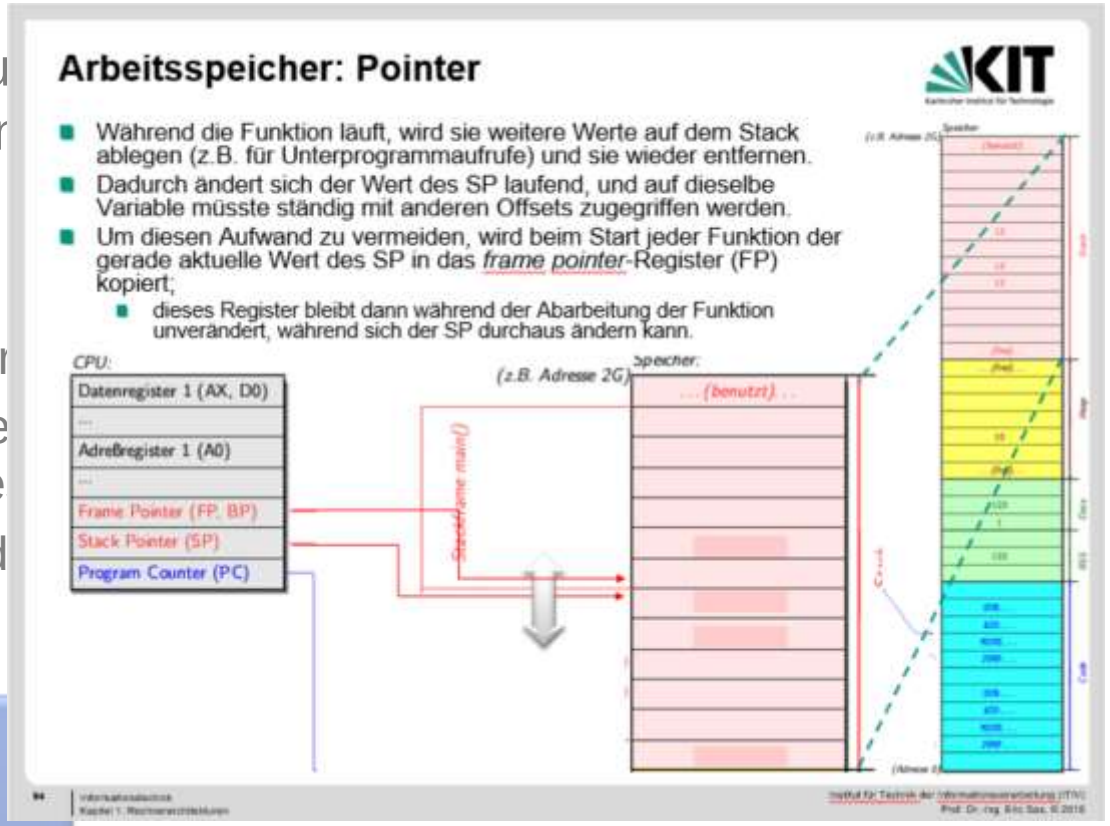
Stackmaschine

- Befehle beziehen sich au
  zur Speicherung u. a. vor

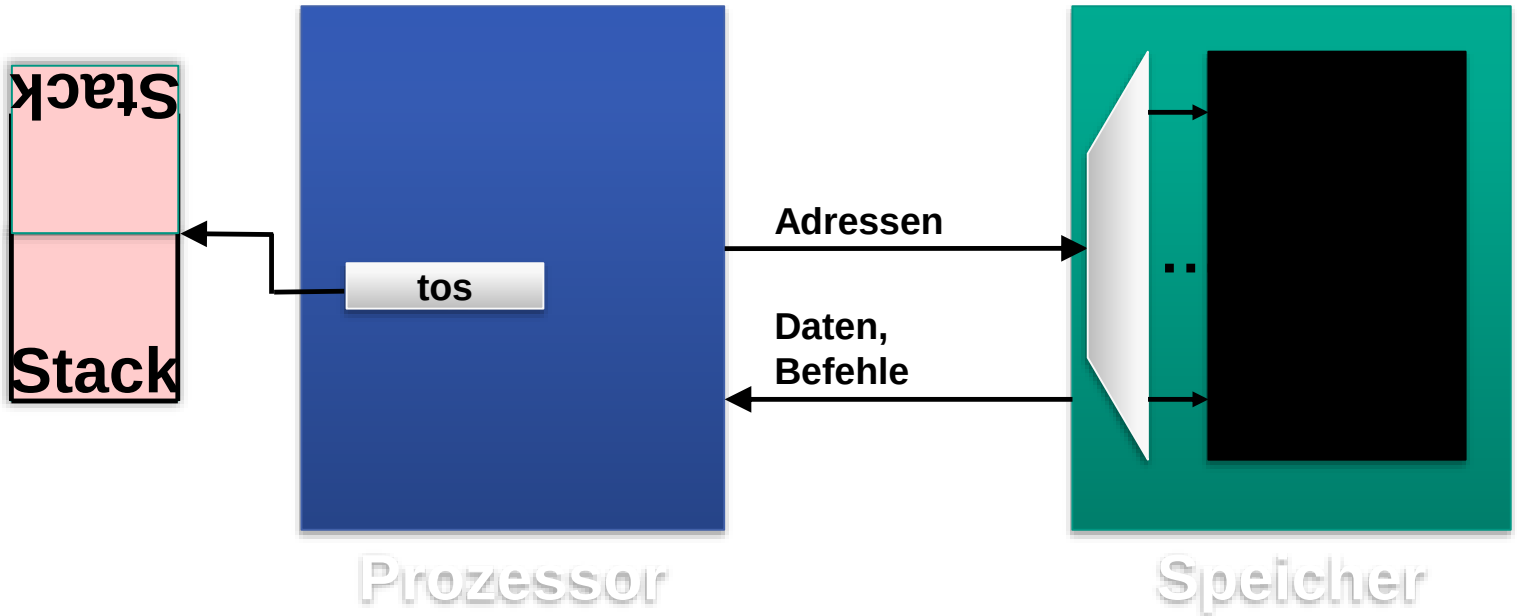
- Typische Befehle:
  - PUSH m: lege den Wer
  - ADD: addiere die beide
    die Summe als oberste
  - STORE m: speichere d
    Adresse m

Beispiel: c := a+b

```
PUSH a
PUSH b
ADD
STORE c
```



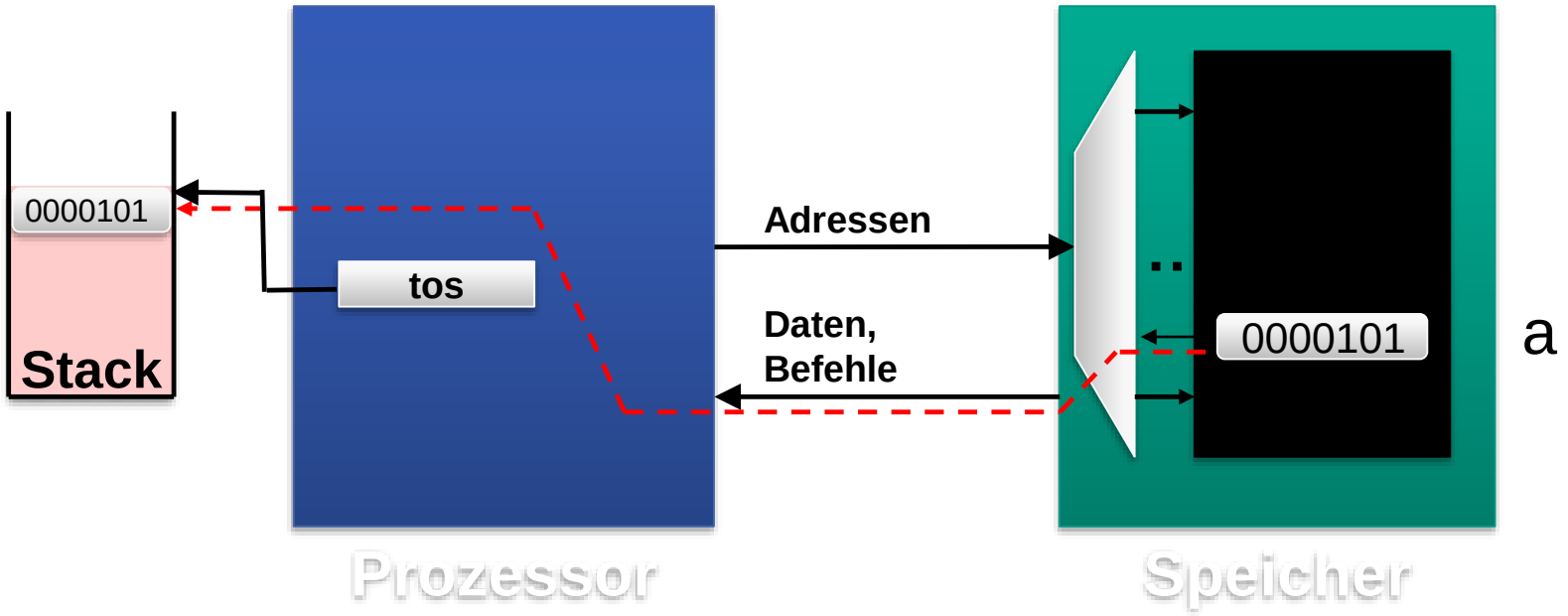
Struktur der Stackmaschine



- Der Prozessor hat Zugriff auf einen Zwischenspeicher mit *LIFO**-Prinzip, den „*Stack*“ oder Stapelspeicher
- Der Prozessor verwaltet einen Verweis auf das oberste Stack-Element: *tos – top of stack (*s. zuvor*: Stack Pointer)*

**LIFO = Last in, first out*

Befehlsausführung

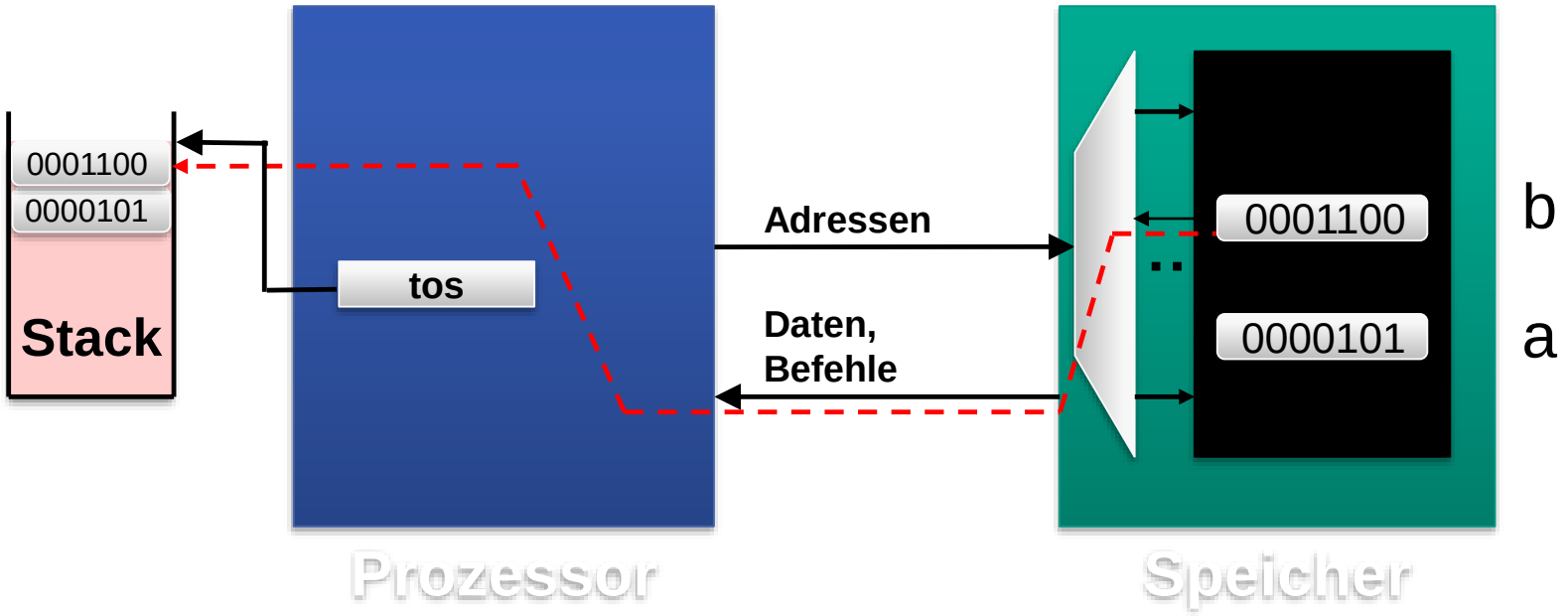


Beispiel: *c := a + b*

- Daten aus Register a auf den Stack legen

PUSH a
PUSH b
ADD
STORE c

Befehlsausführung

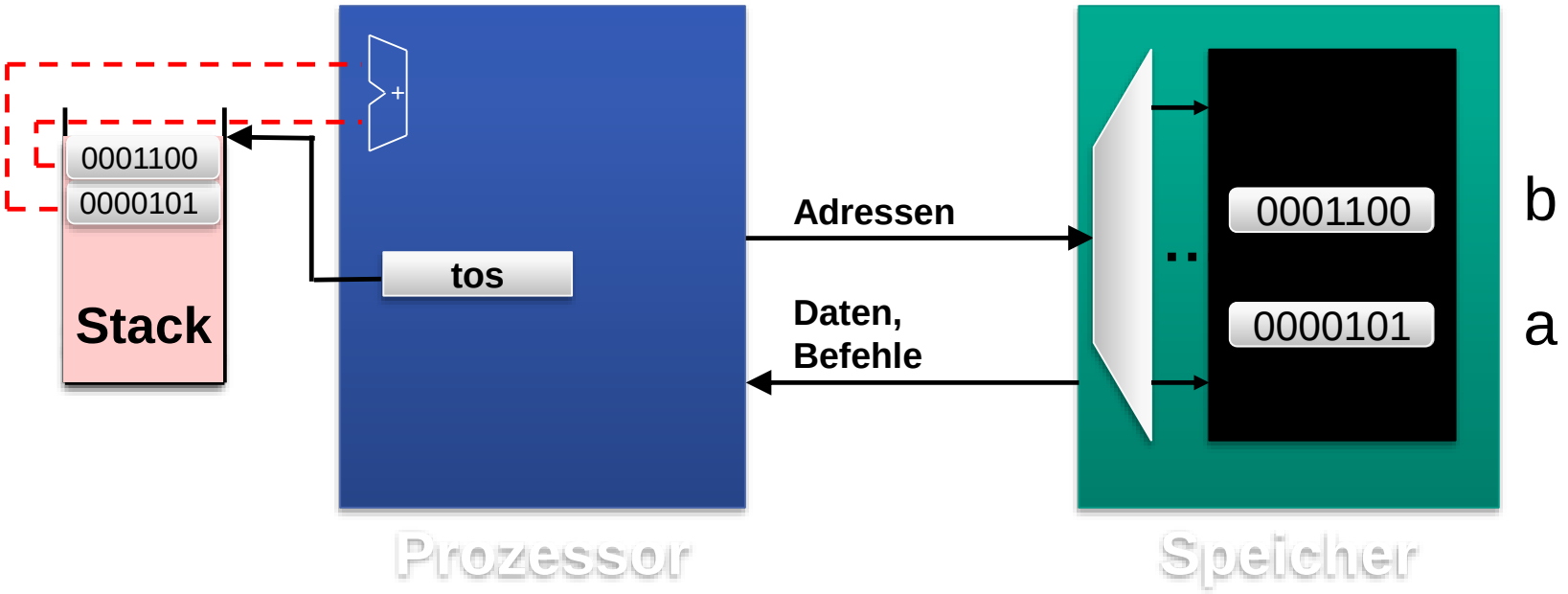


Beispiel: *c := a + b*

- Daten aus Register b auf den Stack legen

PUSH a
PUSH b
ADD
STORE c

Befehlsausführung

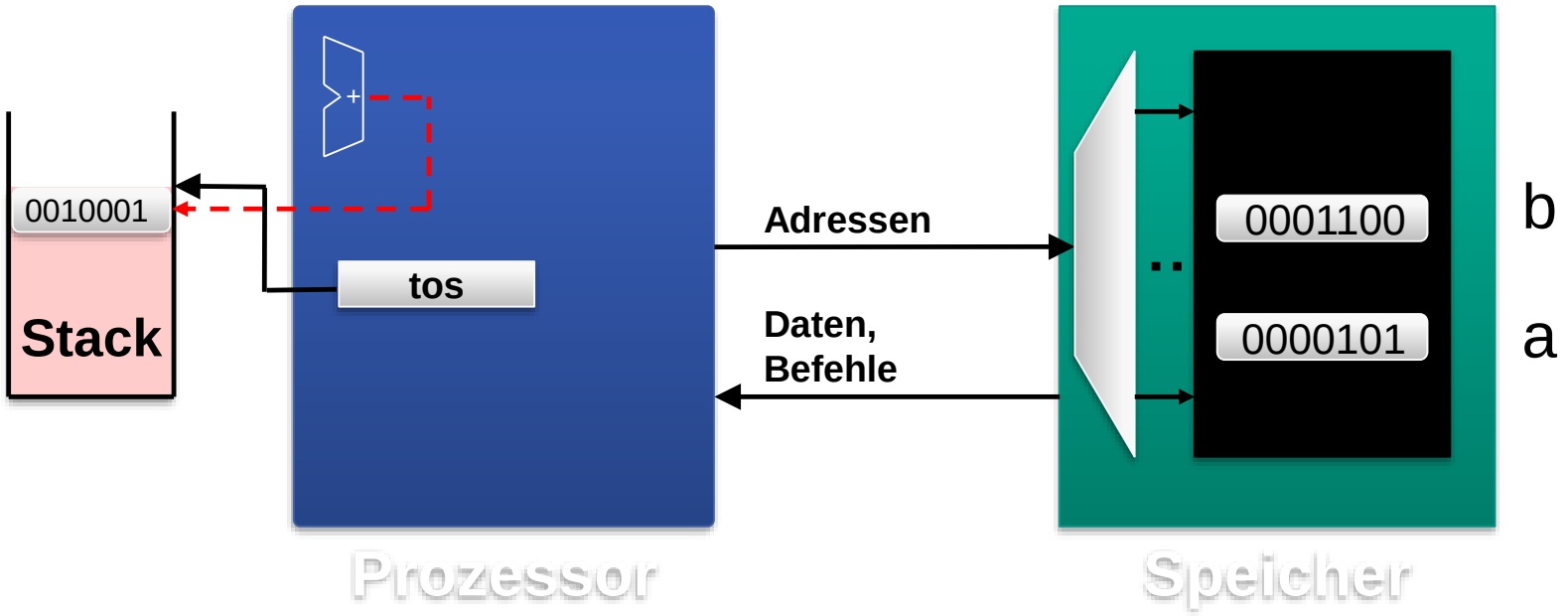


Beispiel: *c := a + b*

PUSH a
PUSH b
ADD
STORE c

- Die ersten beiden Werte auf dem Stack addieren
- Operanden auf dem Stack freigeben (tos verschieben)

Befehlsausführung

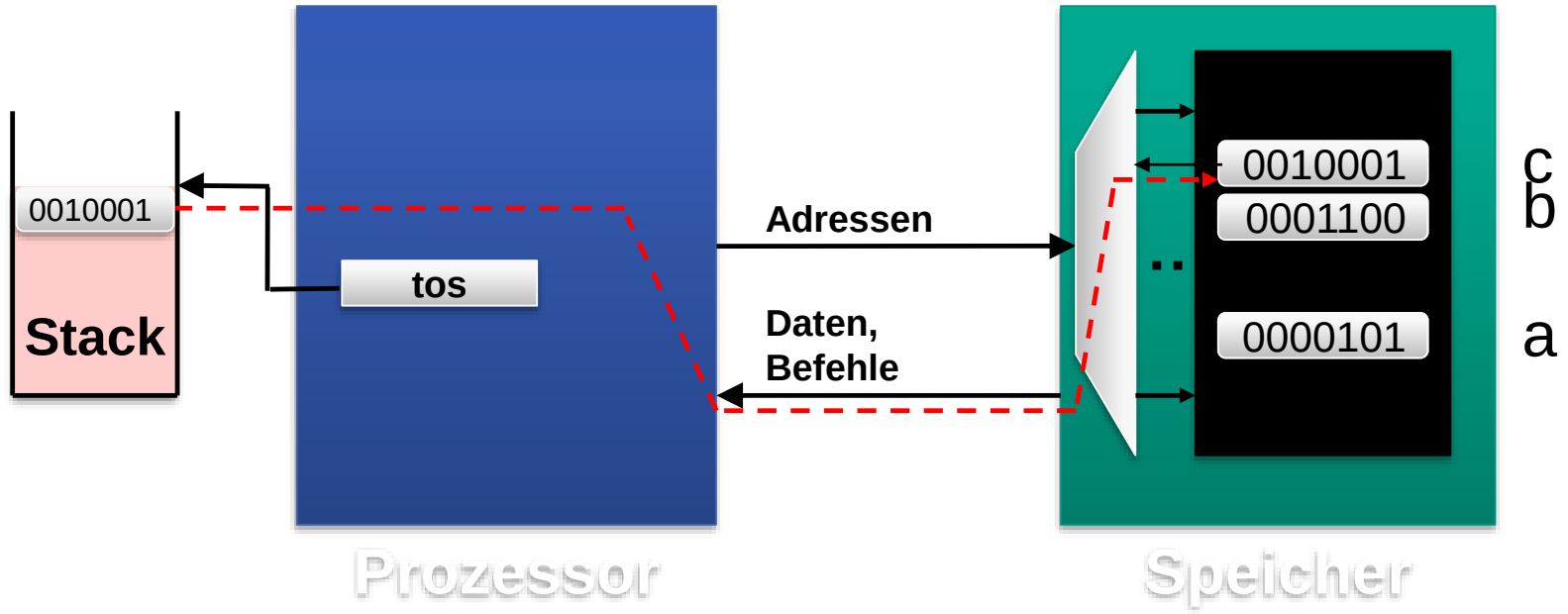


Beispiel: *c := a + b*

PUSH a
PUSH b
ADD
STORE c

- Die Summe wird dann wieder auf dem Stack abgelegt

Befehlsausführung



Beispiel: $c := a + b$

PUSH a
PUSH b
ADD
STORE c

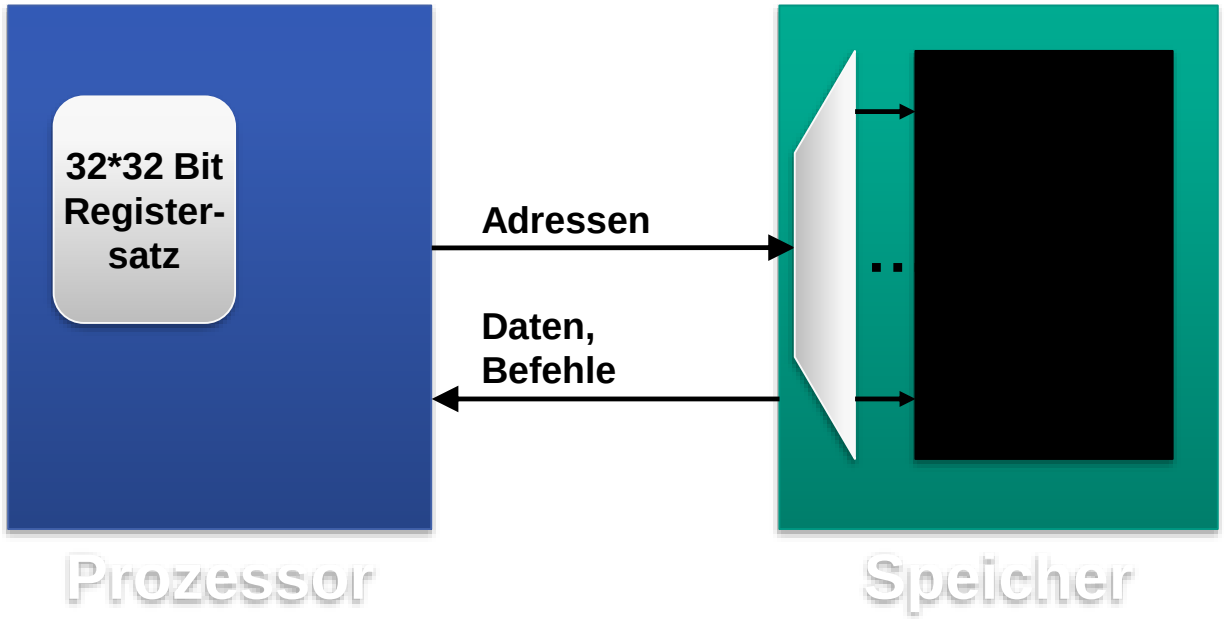
- Das oberste Element des Stacks wird an Adresse c gespeichert

Registersatzmaschine

- Der Prozessor enthält z.B. 16 oder 32 Universalregister
- Üblich sind Dreiadressbefehle: *OP DEST, SRC1, SRC2*
  d.h. *DEST := SRC1 - OP - SRC2*
- Oder Zweiadressbefehle: *OP DEST, SRC*
  d.h. *DEST := DEST - OP - SRC*

z.B.: *ADD R5, R6*
d.h. *R5 := R5 + R6*

Struktur der Registersatzmaschine

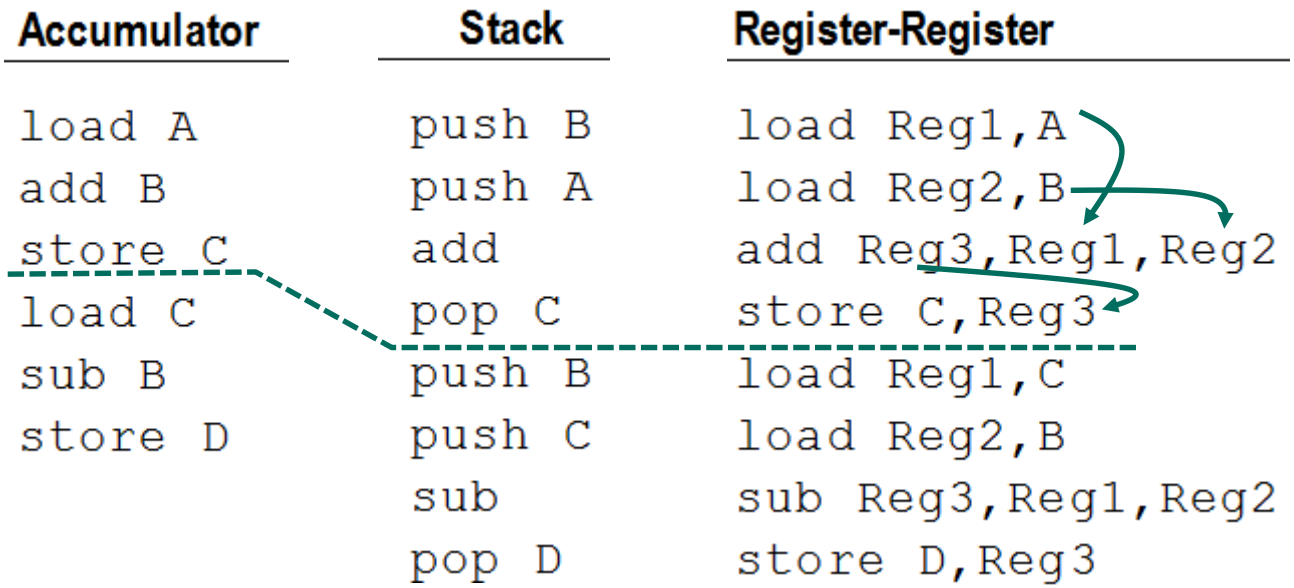


- Im Prozessor befindet sich kein Akkumulator, kein Stack
- Zur Berechnung befüllt der Prozessor seinen internen Registersatz mit Werten aus dem Speicher

Beispiel

```
C = A + B
D = C - B
```

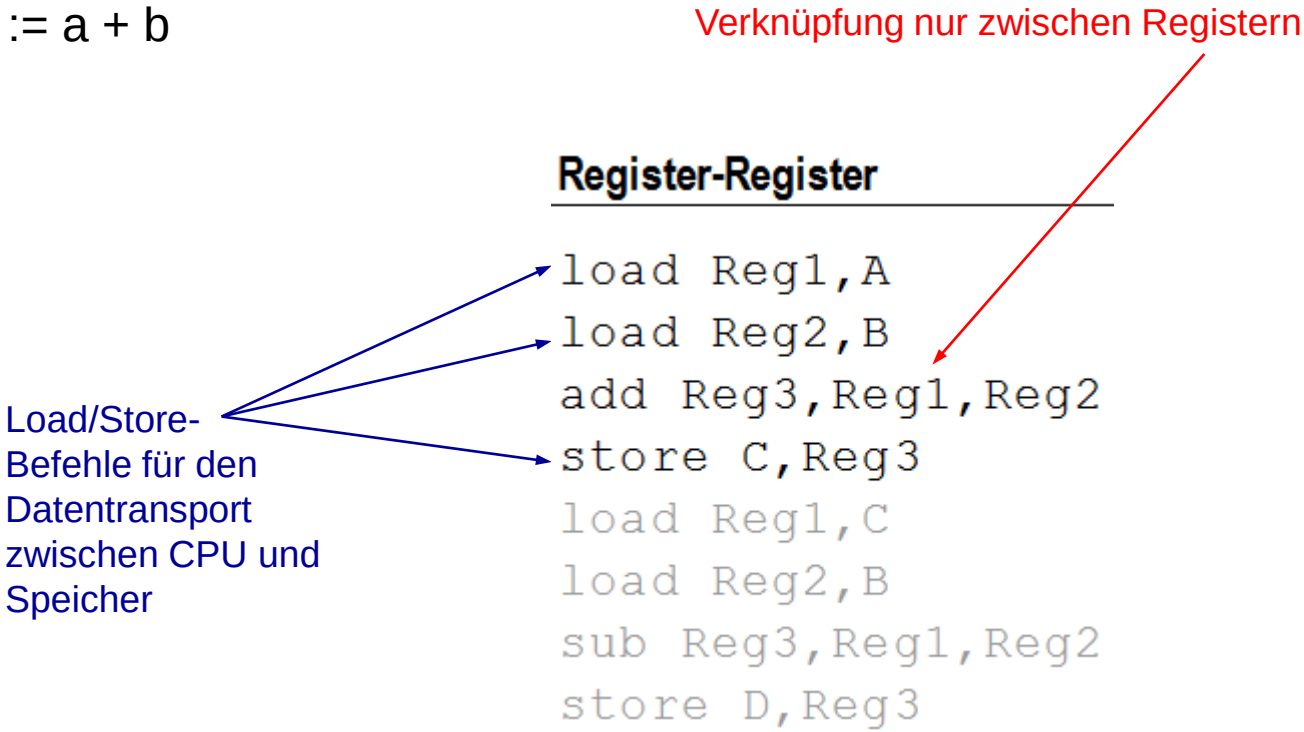
codiert in Klassen von ISA Befehlsformaten / Rechnerarchitekturen:



Accumulator	Stack	Register-Register
load A	push B	load Reg1,A
add B	push A	load Reg2,B
store C	add	add Reg3,Reg1,Reg2
load C	pop C	store C,Reg3
sub B	push B	load Reg1,C
store D	push C	load Reg2,B
	sub	sub Reg3,Reg1,Reg2
	pop D	store D,Reg3

Rechnerarchitekturen

- **RISC (reduced instruction set computer)**, auch Load/Store-Architektur
- Beispiele: SPARC, Alpha, PowerPC
  - Wenige, i.a. gleich-lange, einfache Instruktionen
  - Nur Register als Operanden von Verknüpfungen wie z.B. Addition
  - Beispiel: c := a + b



Typen von Registersatzmaschinen: RISC

RISC - Reduced instruction set computer

- Wenige, i.A. gleich lange, einfache Instruktionen
- Nur Register als Operanden von Verknüpfungen wie z.B: Addition
- i.A. größere (mehr Zeilen) Programme als bei CISC
- 2 bis 5 fache Leistung bei gleicher Technologie (höhere Performance durch weniger Interpretationen
- Beispiele: SPARC, Alpha, PowerPC

Verknüpfung nur zwischen Registern

Register-Register

```
load Reg1,A
load Reg2,B
add Reg3,Reg1,Reg2
store C,Reg3
load Reg1,C
load Reg2,B
sub Reg3,Reg1,Reg2
store D,Reg3
```

Typen von Registersatzmaschinen: RISC und CISC

RISC - Reduced instruction set computer

- Wenige, i.A. gleich lange, einfache Instruktionen
- Nur Register als Operanden von Verknüpfungen wie z.B: Addition
- i.A. größere (mehr Zeilen) Programme als bei CISC
- 2 bis 5 fache Leistung bei gleicher Technologie (höhere Performance durch weniger Interpretationen
- Beispiele: SPARC, Alpha, PowerPC, DEC

CISC - Complex instruction set computer

- Sehr umfangreiche Befehlssätze
- Komplizierte Adressiermodi (z.B: Direktoperand, Register-adressierung…)
- Versuch, die Konstrukte höherer Programmiersprachen durch teilweise sehr komplizierte Befehle zu unterstützen
- Beispiele: VAX 11/780 (1979), INTEL x86 (1981)

Heute nähern sich die Architekturen stark aneinander an

Vergleich: RISC und CISC

	Programmgröße		Transportierte Instruktionen und Daten		CPU Zeit	
	VAX	DEC 3100	VAX	DEC 3100	VAX	DEC 3100
GNU C	410 kB	688 kB	18 MB	21 MB	291 s	90 s
TeX	159 kB	217 kB	67 MB	78 MB	449 s	95 s
Spice	223 kB	372 kB	99 MB	106 MB	352 s	94 s

- CISC Programme (VAX) sind meist kleiner, benötigen jedoch eine längere Ausführungszeit
- RISC Programme (DEC 3100) dagegen sind länger bei gleichzeitig schnellerer Ausführung


Fragen?

IT Inhalt

0. Einleitung und Motivation

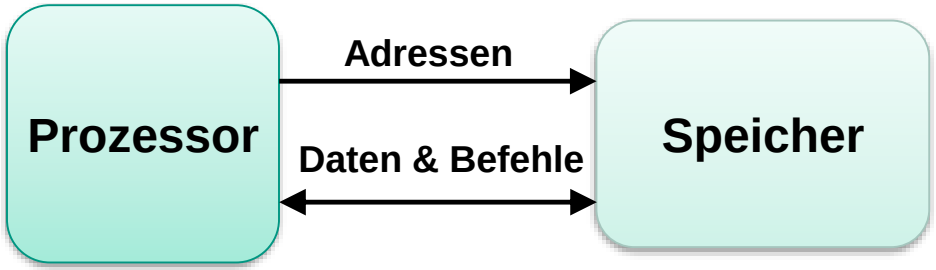
- Organisatorisches
- Begriffe
- Typische Anwendungen

1. Rechnerarchitekturen

- Einführung
- Allgemeiner Aufbau der internen Hardware Architektur
- Instruction Set Architecture (ISA) am Beispiel AVR8
- Klassifikation von Rechnerarchitekturen
- Performanzsteigerung
  - Parallellisierung von Befehls- und Datenzugriffen
    - Harvard Architektur
  - Überlappende Befehlsausführung
    - Pipelining
  - Mehrere Rechenwerke
    - Superskalar
  - Multiprozessor / Multi Core Systeme
  - Beschleunigung Hauptspeicherzugriff
  - Interrupt
- ausgewählte Beispiele

Parallelisierung „Daten- und Befehlszugriffe“

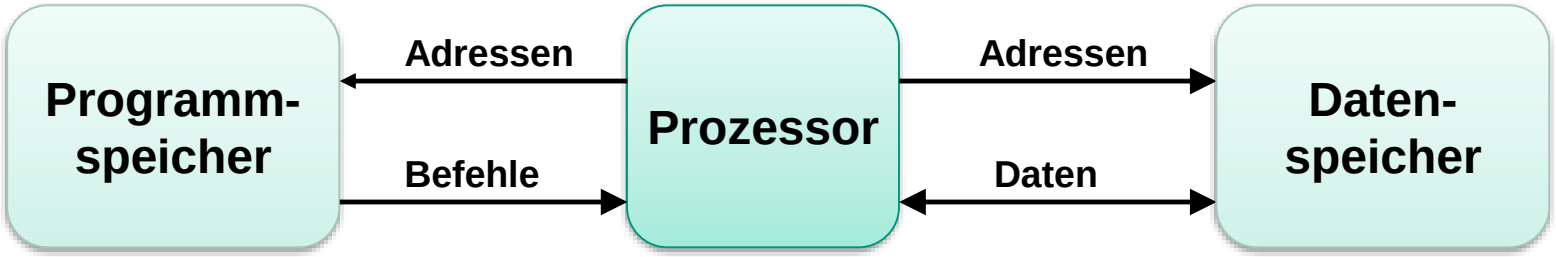
- Von Neumann-Architektur:



Befehlsausführungszeiten im Prozessor heute um Faktor 10-100 mal kleiner als Speicherzugriff: „Von-Neumann-Bottleneck“

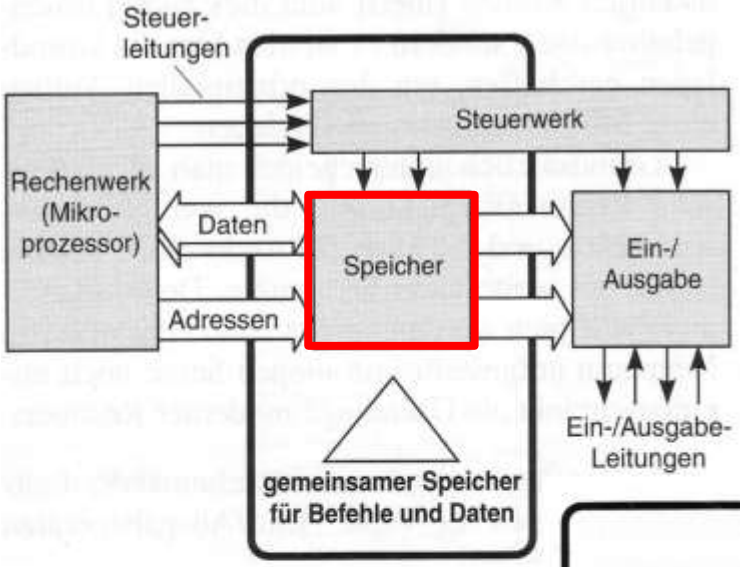
- ➡ mehr Speicher im Prozessor (Register, Cache-Speicher)
- ➡ separate Speicher und Busse für Daten und Befehle (→ Harvard-Architektur)

- Harvard-Architektur:

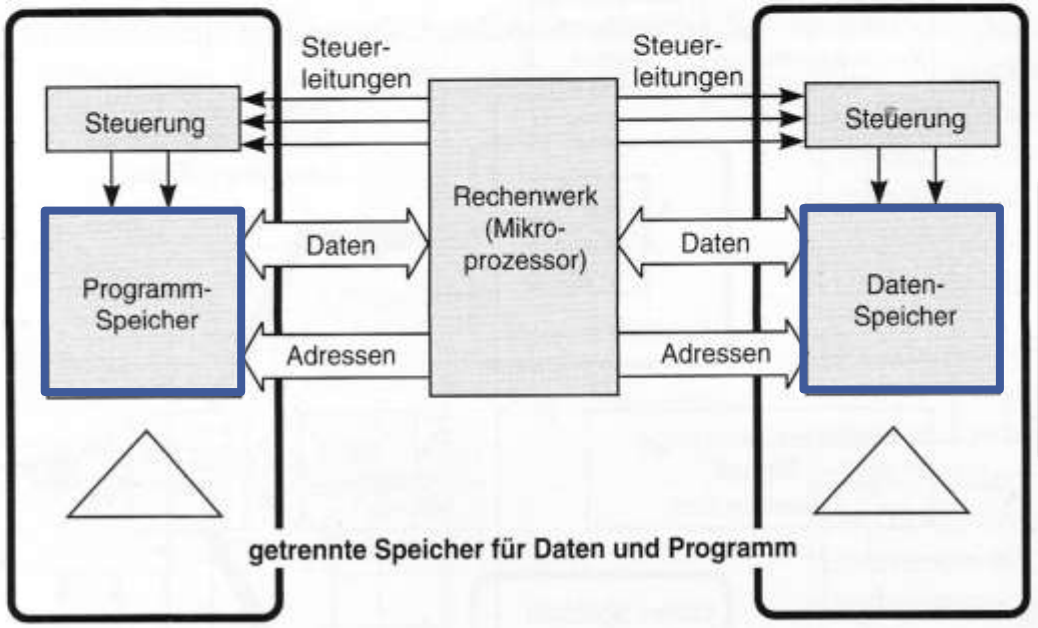


Von Neumann und Harvard Architektur

Von Neumann



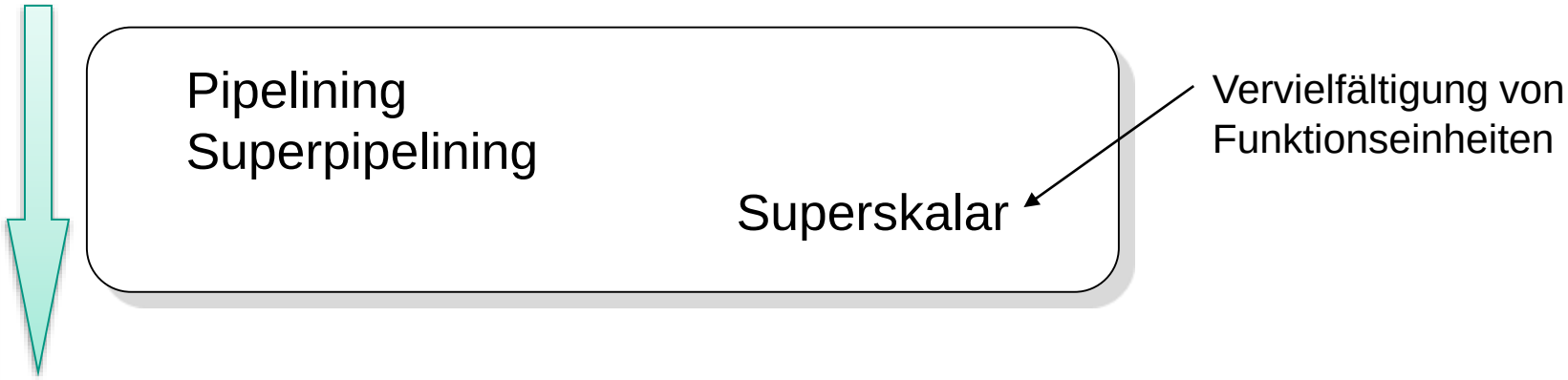
Harvard



Überlappung von Befehlshol- und Ausführungsphase

- Parallel: Befehls- und Datenzugriffe in getrenntem Hauptspeicher
  - Harvard Architektur (s. zuvor)

- Seriell: Überlappung von Befehlshol- und Ausführungsphase
  - Techniken zur Beschleunigung der Instruktionsausführung



Ziel: Überlappung von Befehlshol- und Ausführungsphase

- Beispiel: einfache Akkumulatormaschine

Befehlsausführung seriell

Befehl 1 → **Befehl 2** → **Befehl 3**

Ziel: Überlappung von Befehlshol- und Ausführungsphase

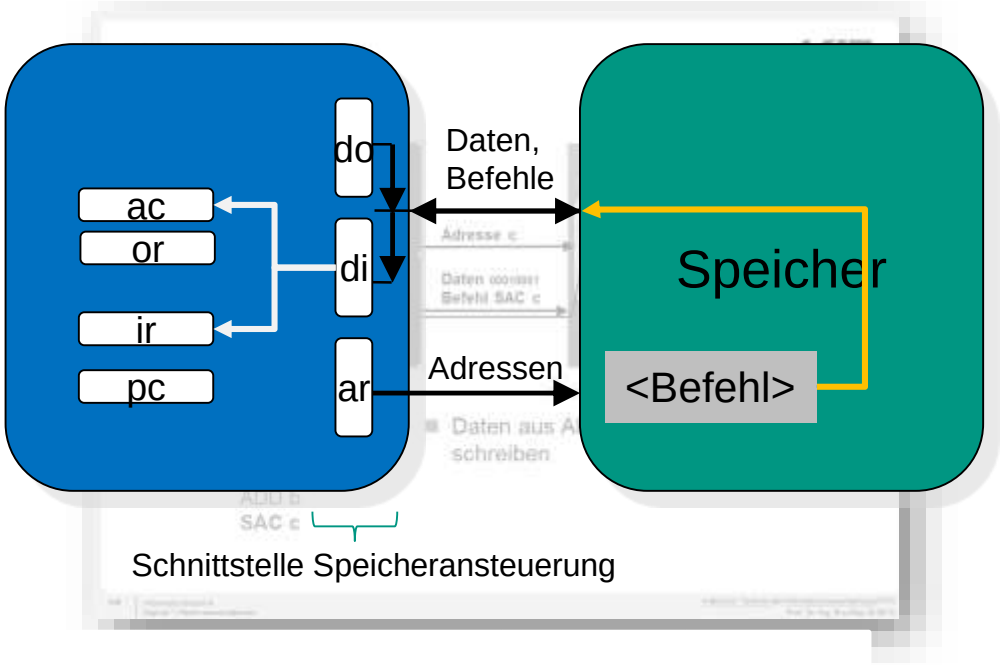
- Beispiel: einfache Akkumulatormaschine

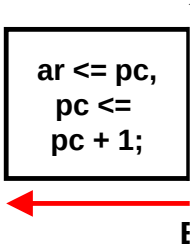
do data out register
Zwischenspeicher für Daten aus Rechenwerk in Speicher schreiben

di data in register
Zwischenspeicher für Daten/Befehle aus Speicher zum Rechenwerk (OR) bzw. zum Steuerwerk (ir)

ar address register
Zwischenspeicher für Speicheradressen

ac accumulator register
or operand register
ir instruction register
pc program counter





Ziel: Überlappung von Befehlshol- und Ausführungsphase

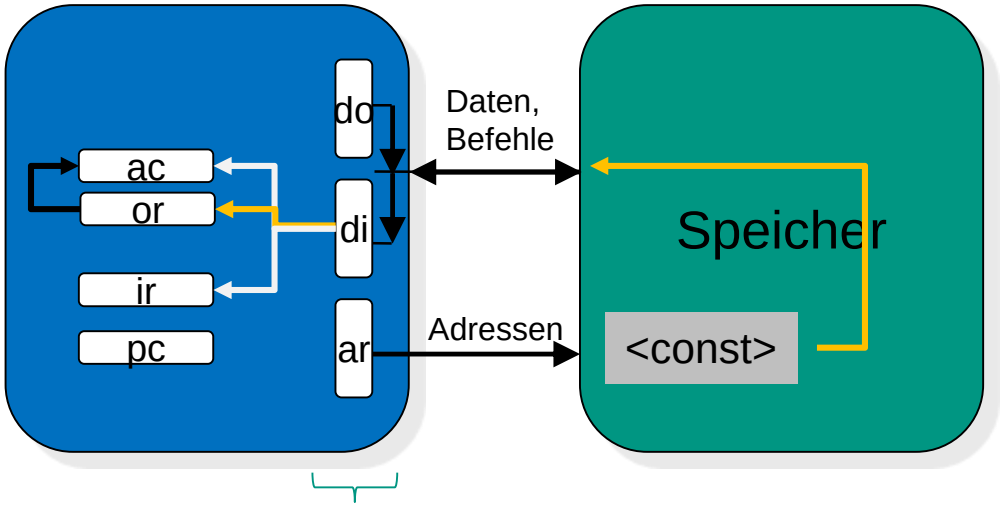
- Beispiel: einfache Akkumulatormaschine

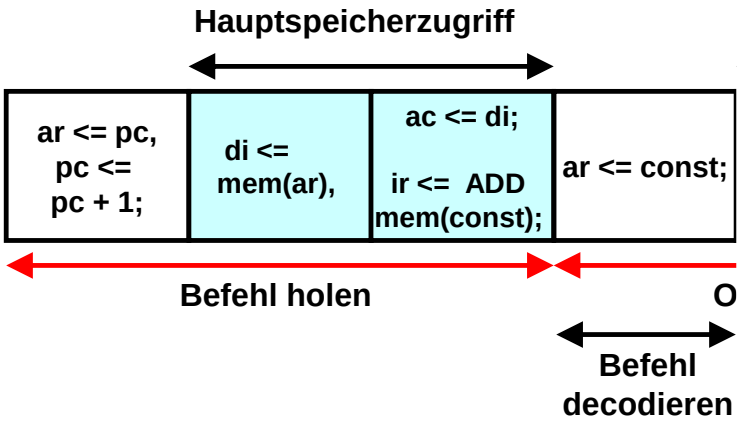
do data out register
Zwischenspeicher für Daten aus Rechenwerk in Speicher schreiben

di data in register
Zwischenspeicher für Daten/Befehle aus Speicher zum Rechenwerk (OR) bzw. zum Steuerwerk (ir)

ar address register
Zwischenspeicher für Speicheradressen

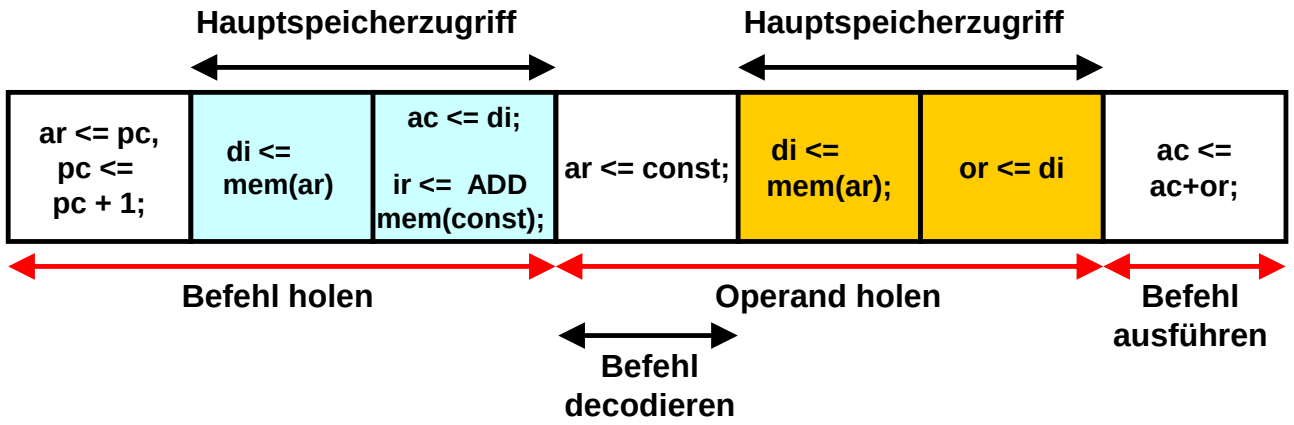
ac accumulator register
or operand register
ir instruction register
pc program counter



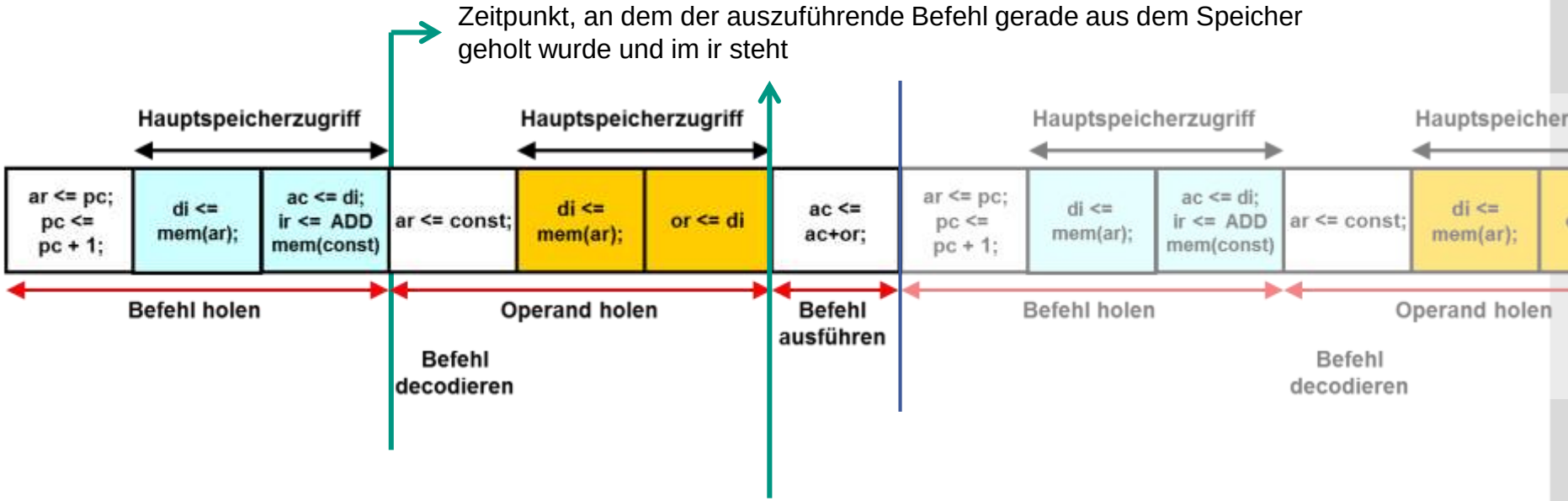


Ziel: Überlappung von Befehlshol- und Ausführungsphase

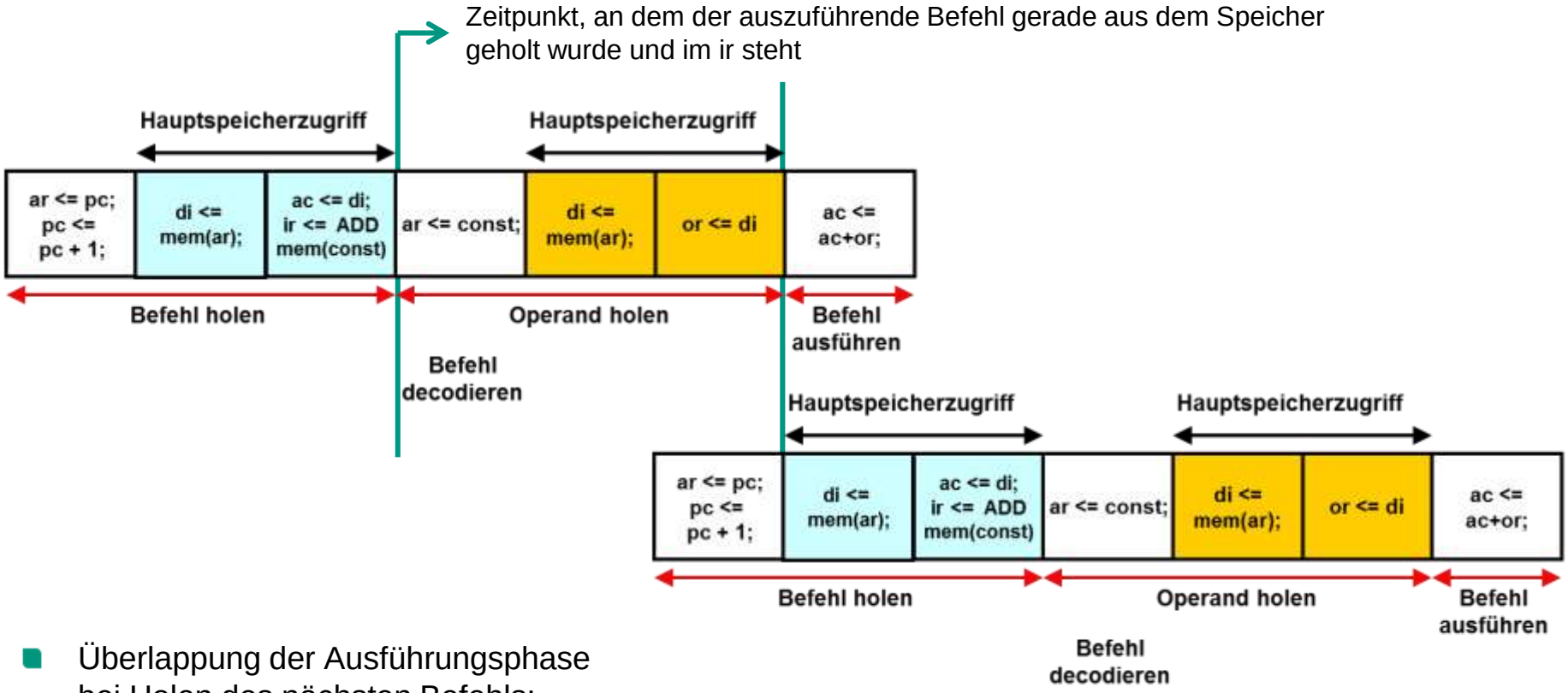
- Hauptspeicherzugriffszeit = 2 * Prozessortaktzeit
- Befehlsauslegung für größtmögliche Geschwindigkeit
- Beispiel: Addiere-zu-Akkumulator-Befehl
- Auslastung CPU: 71%, Speicher 57%



Befehlsausführung überlappend



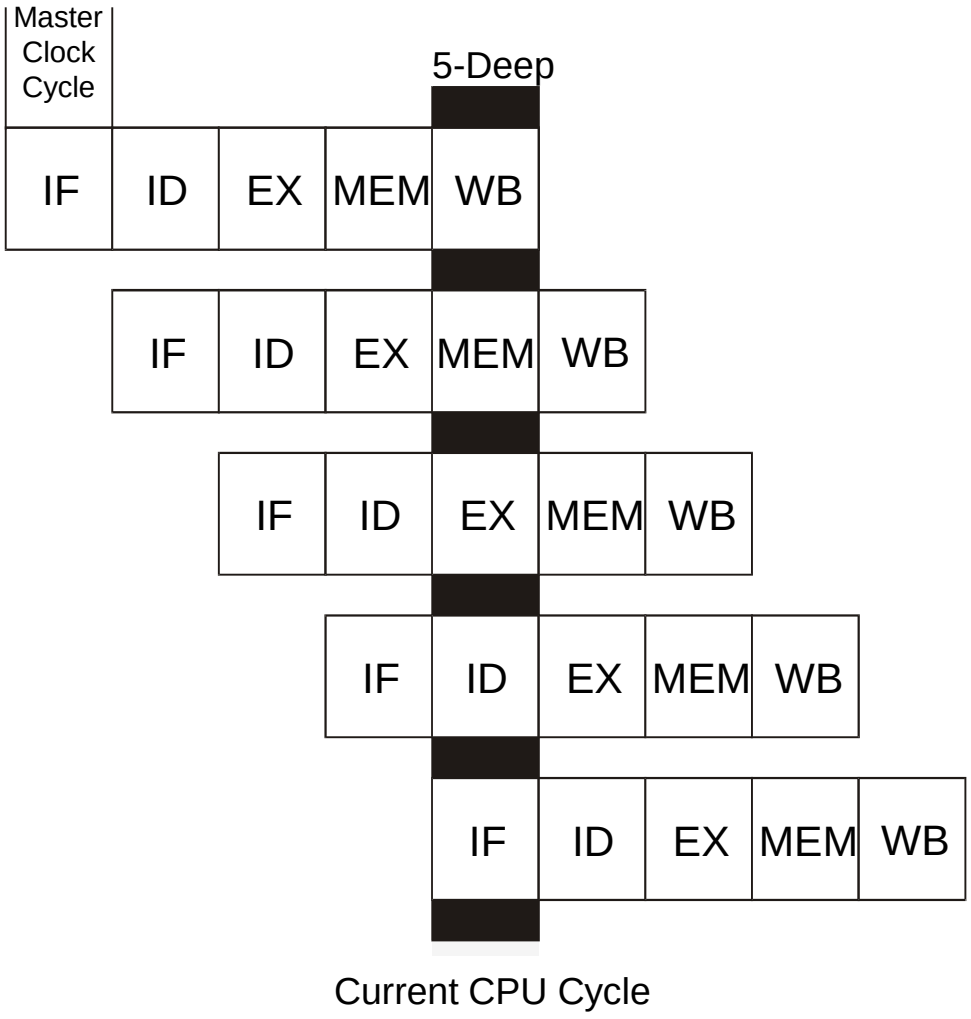
Befehlsausführung überlappend



- Überlappung der Ausführungsphase bei Holen des nächsten Befehls:
  - Während eines Speicherzugriffs wird bereits der nächste Speicherzugriff vorbereitet
- Ebenfalls Parallel zum Speicherzugriff:
  - Kalkulation des Ergebnisses („Befehl ausführen“)
- Keine zusätzlichen Kosten/Ressourcen benötigt
- Zeitliche Einsparung von 2 Takten/Befehl => 28,6 % Leistungssteigerung
- Auslastung: CPU 80%, Speicher 80% (vgl. zuvor: Auslastung CPU: 71%, Speicher 57%)

→ Pipeling 2-stufig

Typische Pipeline (5-stufig)



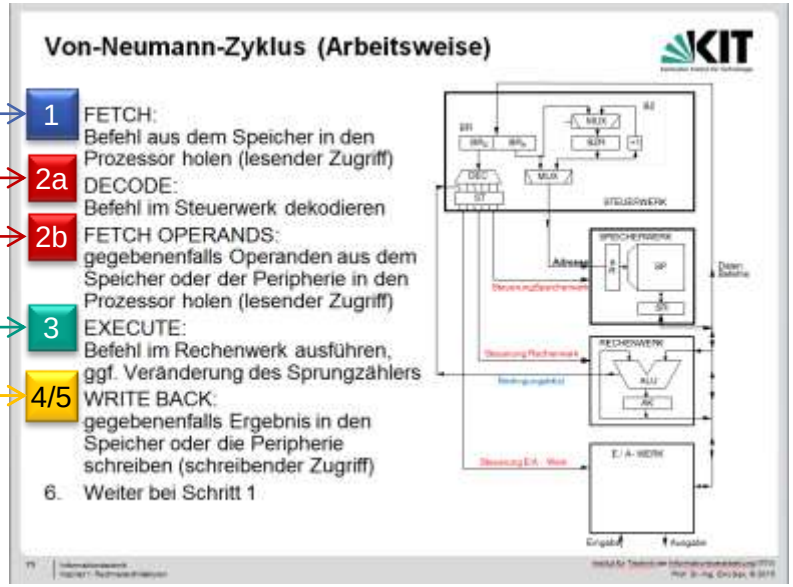
IF -- Instruction Fetch

ID -- Instruction Decode/Register Fetch

EX -- Execute/Address Calculation

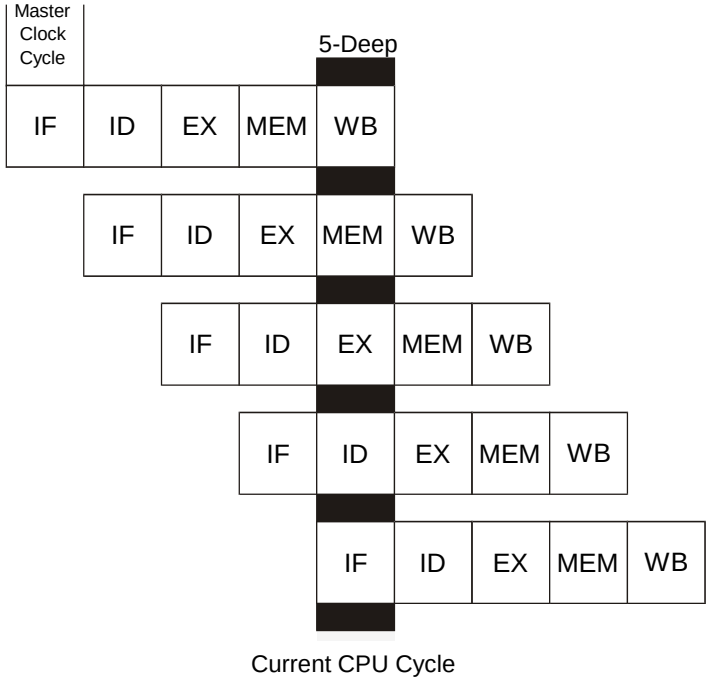
MEM -- Memory Access

WB -- Write Back

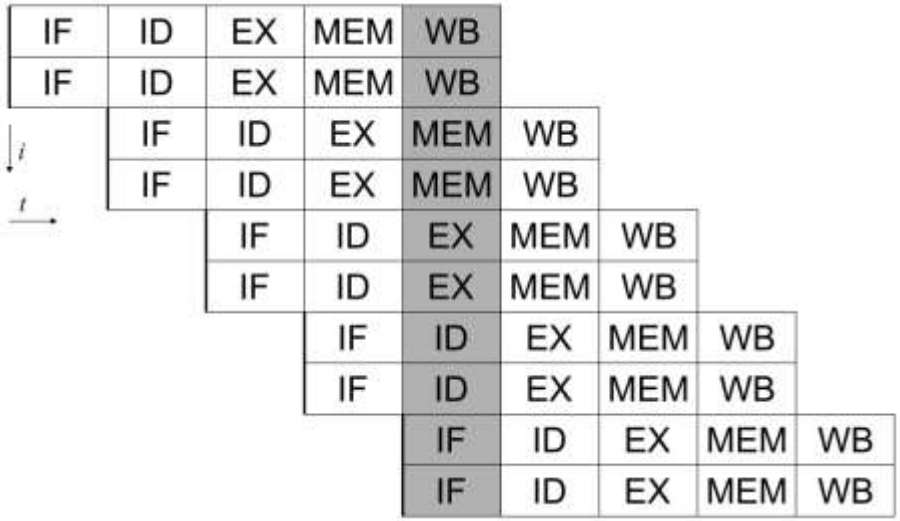


Fortgeschrittene Organisationsprinzipien (1)

- **Pipelining (zeitliche Parallelität)**:
  - Überlappung der Ausführungsphase
  - Erhöhung des Datendurchsatzes
  - Skalar → 1 Ausführung pro Takt



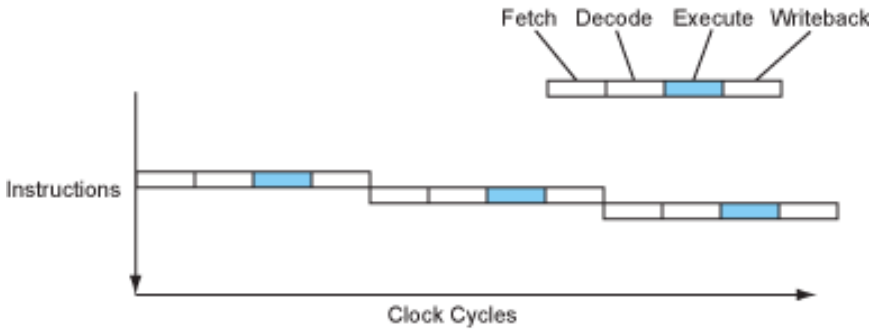
- **Superskalar (räumliche Parallelität):**
  - Paralleles Ausführung von mehreren Befehlen auf einem Rechner
  - Parallel arbeitende Funktionseinheiten (Hardware)
  - Im Idealfall Vervielfachung der Leistung



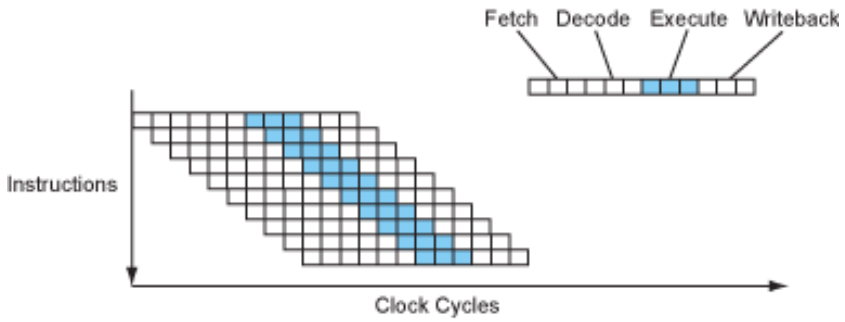
Remark: 5 cycles per instruction (CPI = 5)

Fortgeschrittene Organisationsprinzipien (2)

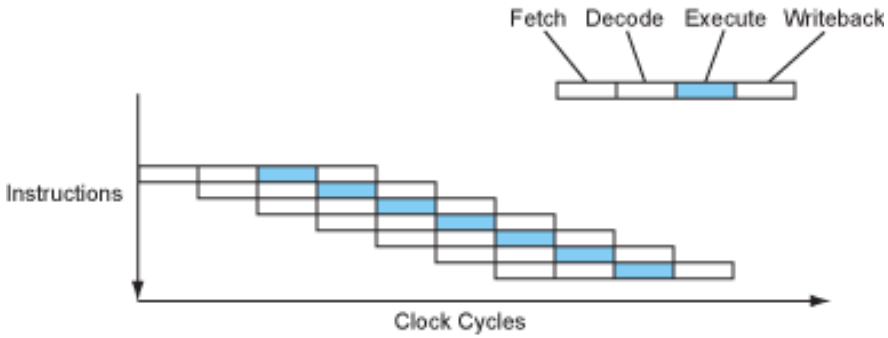
- Sequentielle Abarbeitung



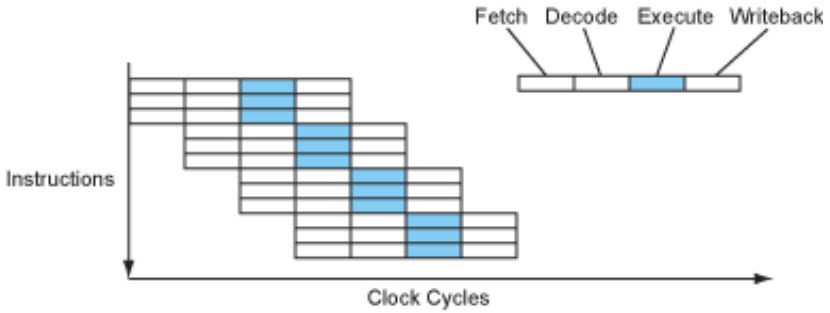
- Superpipelining



- Pipelining



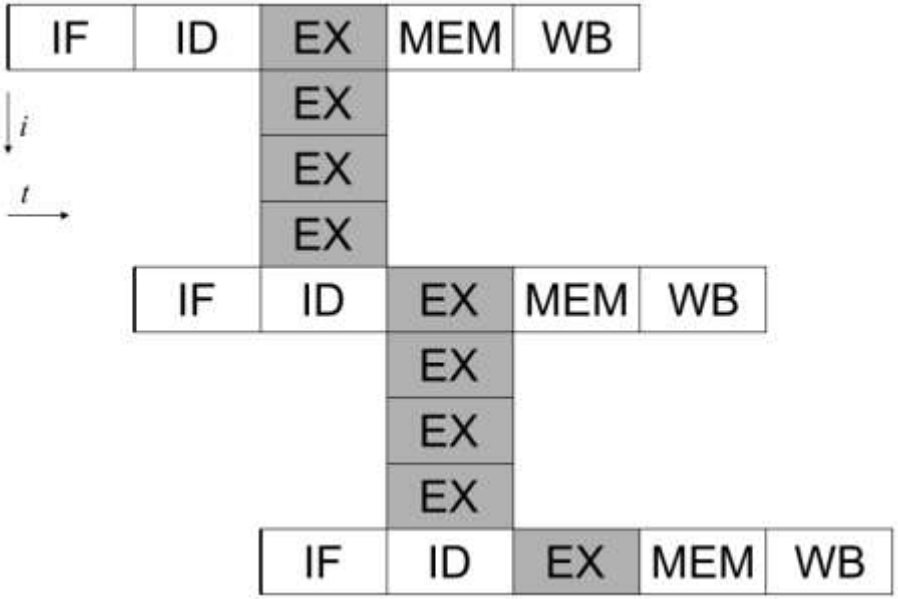
- Superscalar



Remark: 4 cycles per instruction (CPI = 4) Quelle: http://www.lighterra.com

Fortgeschrittene Organisationsprinzipien (3)

- VLIW: Very long instruction word computer
  - Der Compiler gruppiert parallel ausführbare Befehle (vs. dynamisch s. superskalar)
  - Deutlich längere Befehle, in denen die parallel auszuführenden Befehle vorgegeben werden.
  - Teilweise wird dies auch EPIC (explicitly parallel instruction computing) genannt
- Multi-Prozessor/Multi-Core Sys.:
- Pipelining und VLIW

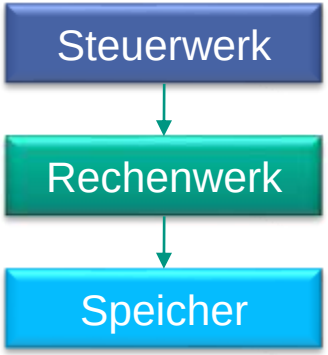


		Data: Single	Data: Multiple
Instruction	Multiple	**MISD**	**MIMD**
Instruction	Single	**SISD**	**SIMD**

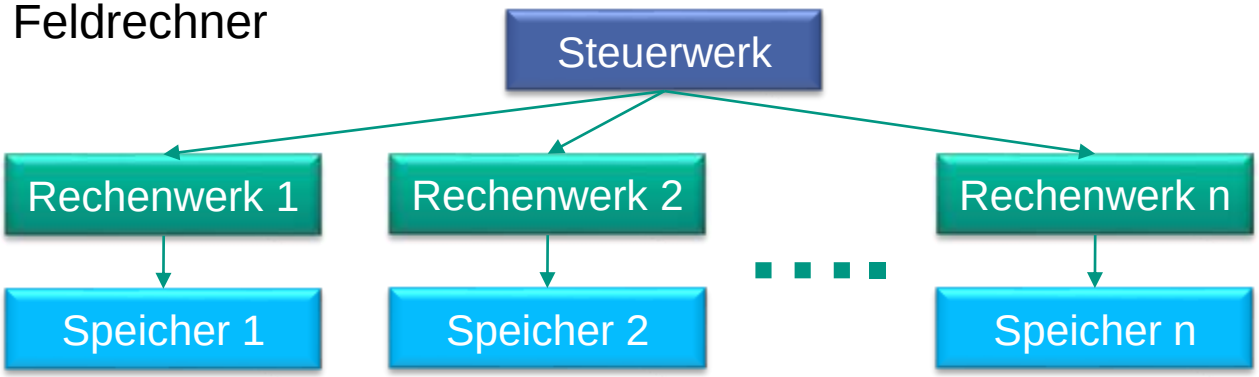
Klassifikation nach Flynn (1966)

Multiprozessor-/Multicore-Systeme

- Single Instruction, Single Data (SISD):
  - Konventionell von Neumann oder Harvard;
  - Pipelining möglich



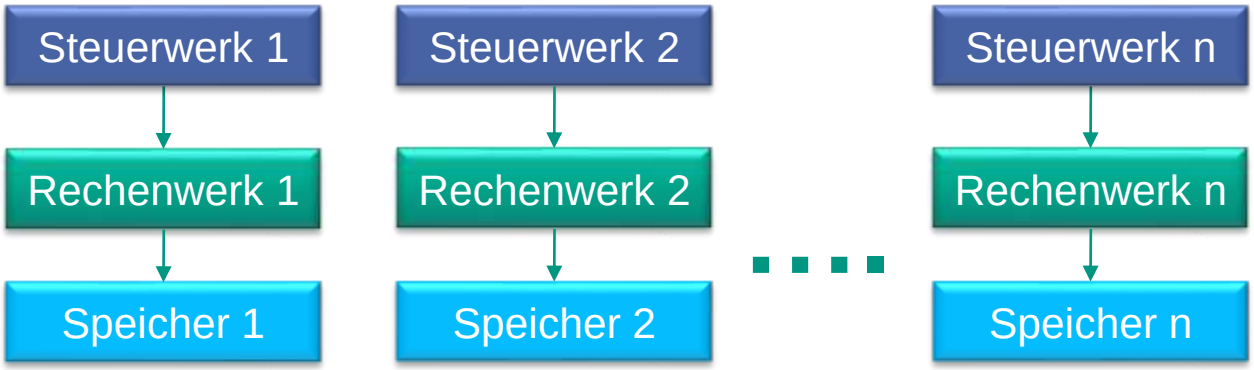
- Single Instruction, Multiple Data (SIMD):
  - Pipelining und Superskalarität möglich
  - Feldrechner



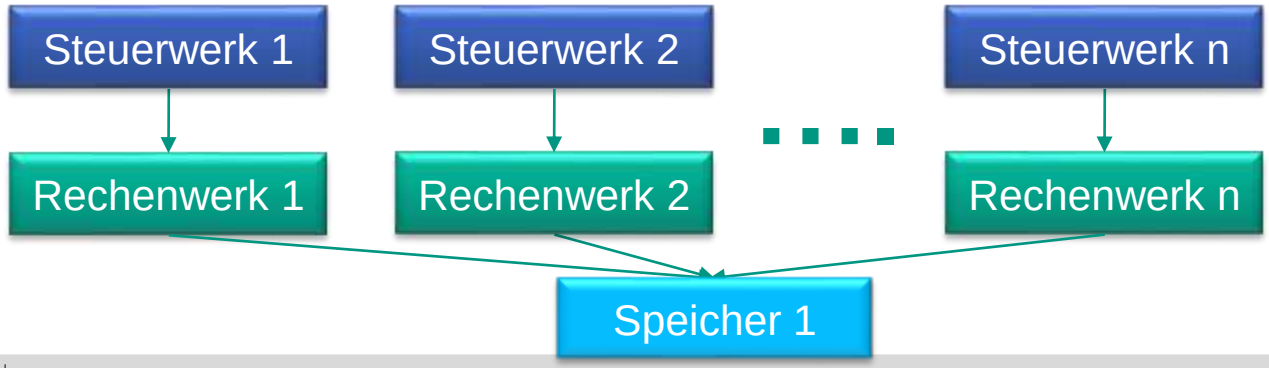
Klassifikation nach Flynn (1966)

Multiprozessor-/Multicore-Systeme

- Multiple Instruction, Multiple Data (MIMD):
  - Komplette Parallelverarbeitung, Multi-Prozessorsysteme



- Multiple Instruction, Single Data (MISD):
  - Keine Bedeutung




Fragen?

Feststellung Status von Ein-/Ausgabe-Geräten oder zeitkritischen Prozessen

- Häufig ist sofortige Reaktion auf Eingabe/Ausgabe-Ereignisse erforderlich
  - z. B. von Tastatur, Maus, Festplatte, Netzwerk, Zeitgeber/Timer Zähler, Division durch 0, Analog-Digital-Wandler, Watchdog, externe Unterbrechungen, usw.
  - auch wenn ein anderer Programmcode (z. B. Anwendungsprogramm) gerade abgearbeitet wird
- Feststellen Status durch „Polling“ oder durch „Interrupts“
  - Polling (programmierte zyklische Abfragen)
    - einfach, benötigt keine zusätzliche Hardware, belegt aber ständig CPU-Leistung. Bei Multitasking-Betriebssystemen ist das Polling als alleinige Methode nicht möglich.
  - Interrupt (engl. to interrupt, unterbrechen)
    - Laufzeiteffizient, aber zusätzliche Hardware notwendig

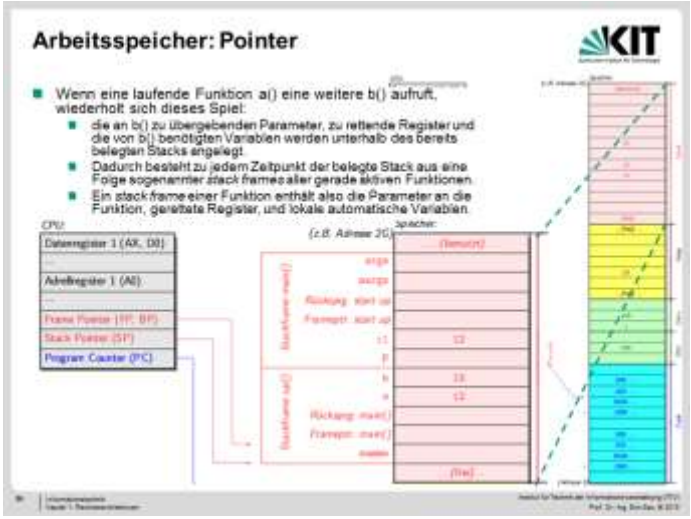
Interrupt vs. Polling - Analogie

- Tür mit Klingel:
  - Während man seine Aufgaben erledigt, kann man jederzeit durch die Klingel unterbrochen werden.
  - Beim Polling – also ohne Klingel – müsste ständig an die Tür gelaufen werden, um nachzuschauen, ob Besuch da ist oder nicht.

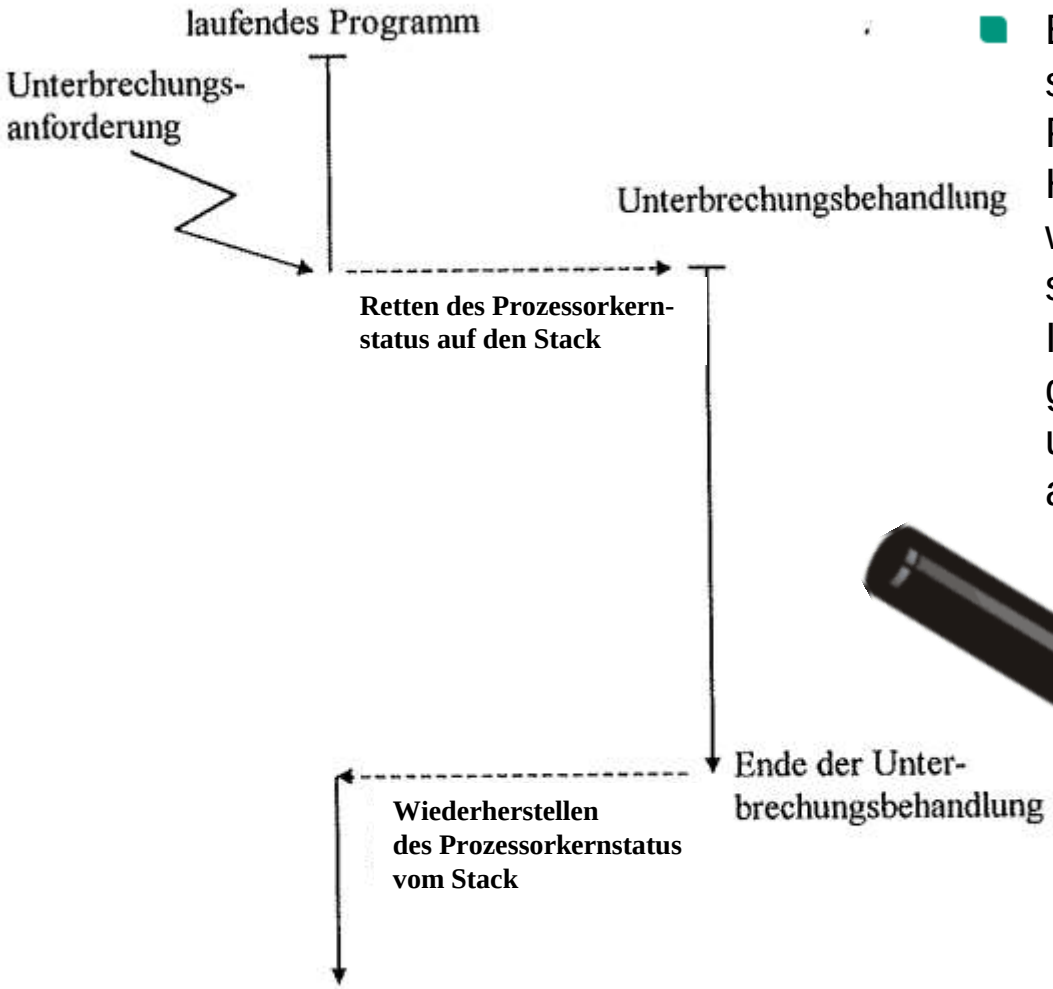
- Beim Erhitzen von Milch hingegen ist es wohl besser, nicht erst auf den „Interrupt“ des Überkochens zu warten, sondern den Prozess vielmehr regelmäßig zu überwachen.

Unterbrechungsverarbeitung - Interrupt

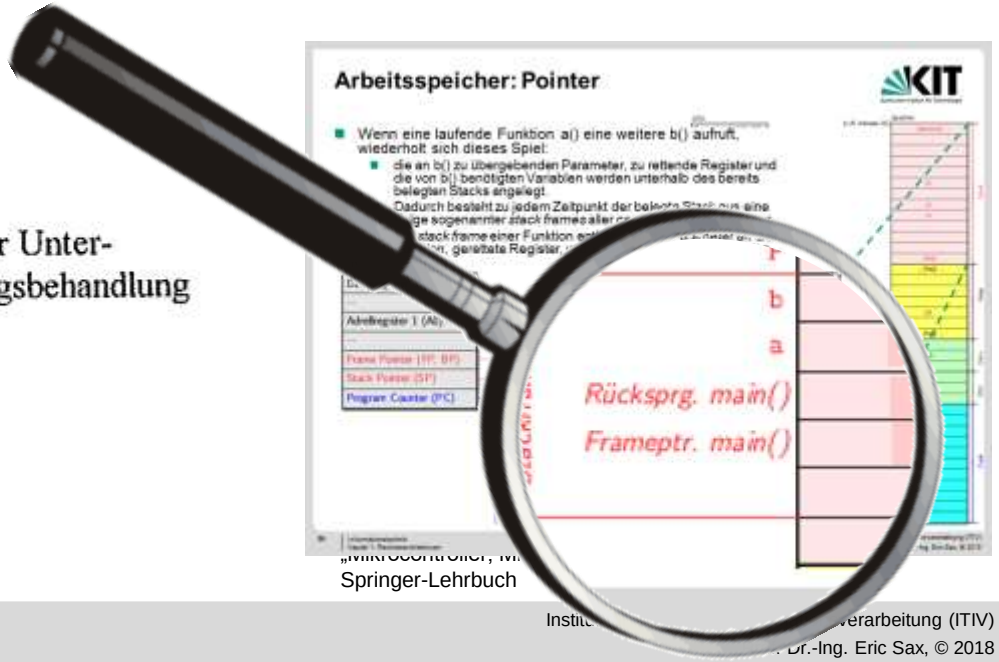
- Interrupt (engl. to interrupt → unterbrechen)
  - die kurzfristige Unterbrechung eines Programms, um eine andere, meist kurze, aber zeitkritische Verarbeitung durchzuführen
- auslösendes Ereignis
  - Unterbrechungsanforderung (Interrupt Request, IRQ oder INT)
- danach Ausführung der Unterbrechungsroutine
  - (Interrupt Service Routine, ISR)
- anschließend wird die Ausführung des Programms an der Unterbrechungsstelle fortgesetzt



Ablauf einer Unterbrechung



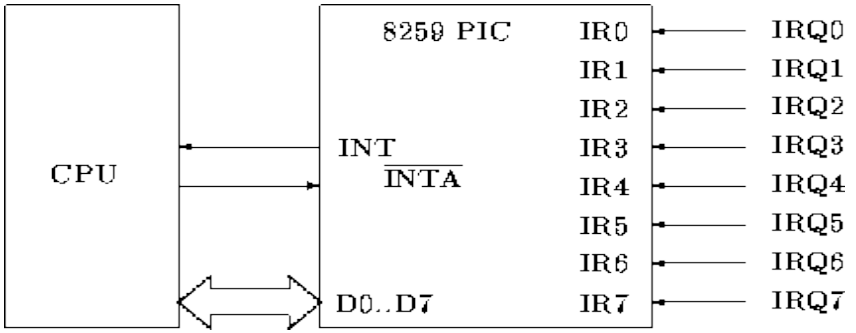
- Eine Unterbrechungs-Routine (interrupt service routine, ISR) ist ein Programmstück, das von einem Hauptprozessor (CPU) ausgeführt wird, wenn er durch eine Unterbrechung-sanforderung (engl. Interrupt request, IRQ) gezwungen wird, den gegenwärtigen Programmablauf zu unterbrechen und einen Interrupt auszuführen.



Springer-Lehrbuch

Ermittlung Startadresse ISR mit Interrupt Controller

- Interrupt (INT) wird durch eine 1-Bit Leitung ausgelöst (IRQn).
  - PIC (programmierbarer Interrupt Controller) hier mit 8 x 1-Bit Eingängen



- Der Prozessor bestätigt mit INTA (*Interrupt Acknowledge*) und ruft die „passende“ (s. nächste Folie) Interrupt-Service-Routine auf.
- Nach Beenden der Service-Routine wird ein EOR (End-of-Interrupt) oder RETI-Befehl an den Controller übergeben.
- Remark:
  - *PIC i.d.R. heute in CPU integriert*

Beispiel für einfache Abarbeitung *(hier: 8 Bit)*

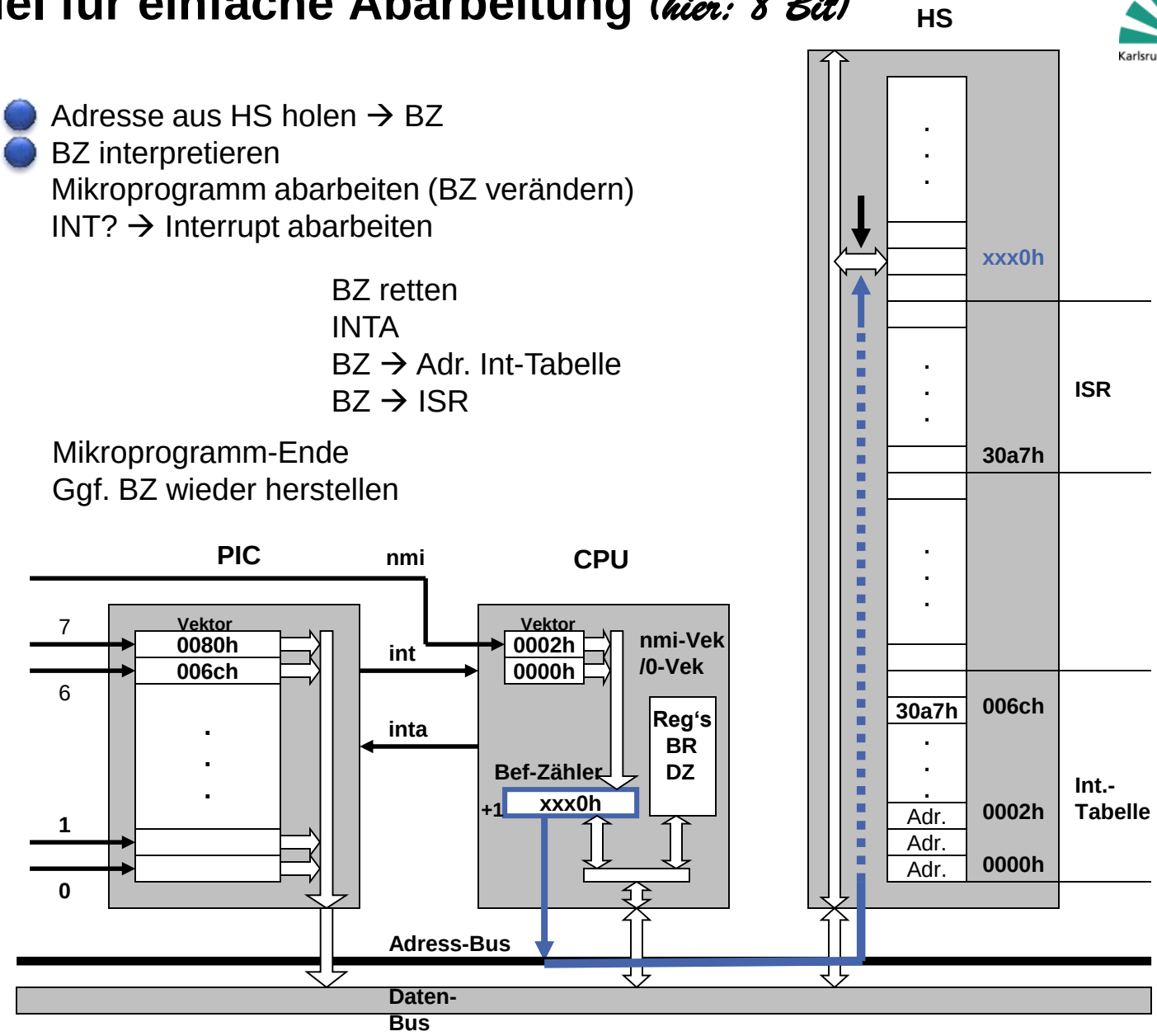
- Adresse aus HS holen → BZ
- BZ interpretieren

Mikroprogramm abarbeiten (BZ verändern)

INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
Ggf. BZ wieder herstellen



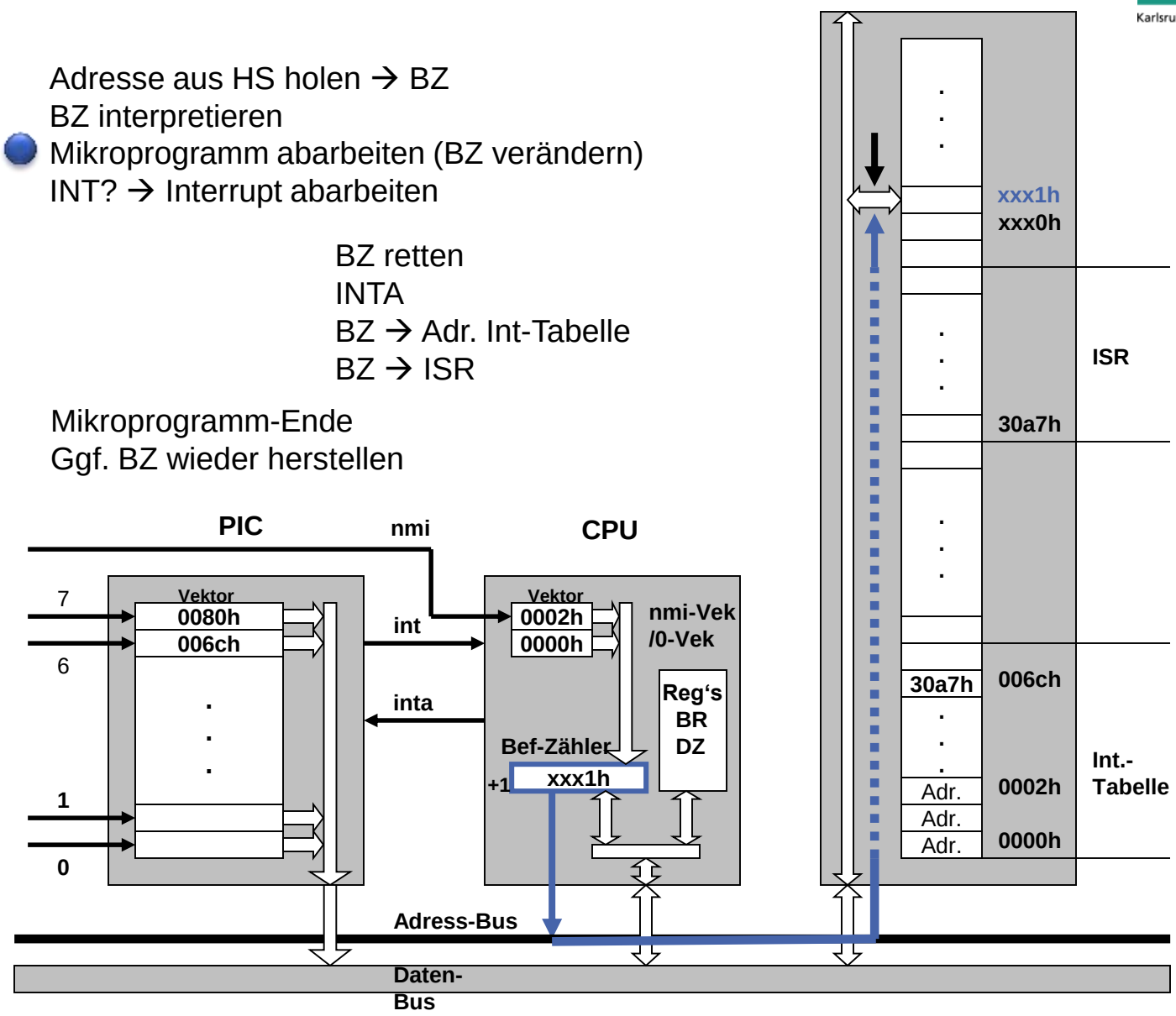
Quelle: http://www.u-r-rennert.de/assembler/rasm/8086_int_prinz.htm

Programm abarbeiten

- Adresse aus HS holen → BZ
- BZ interpretieren
- Mikroprogramm abarbeiten (BZ verändern)
- INT? → Interrupt abarbeiten

  - BZ retten
  - INTA
  - BZ → Adr. Int-Tabelle
  - BZ → ISR

- Mikroprogramm-Ende
- Ggf. BZ wieder herstellen



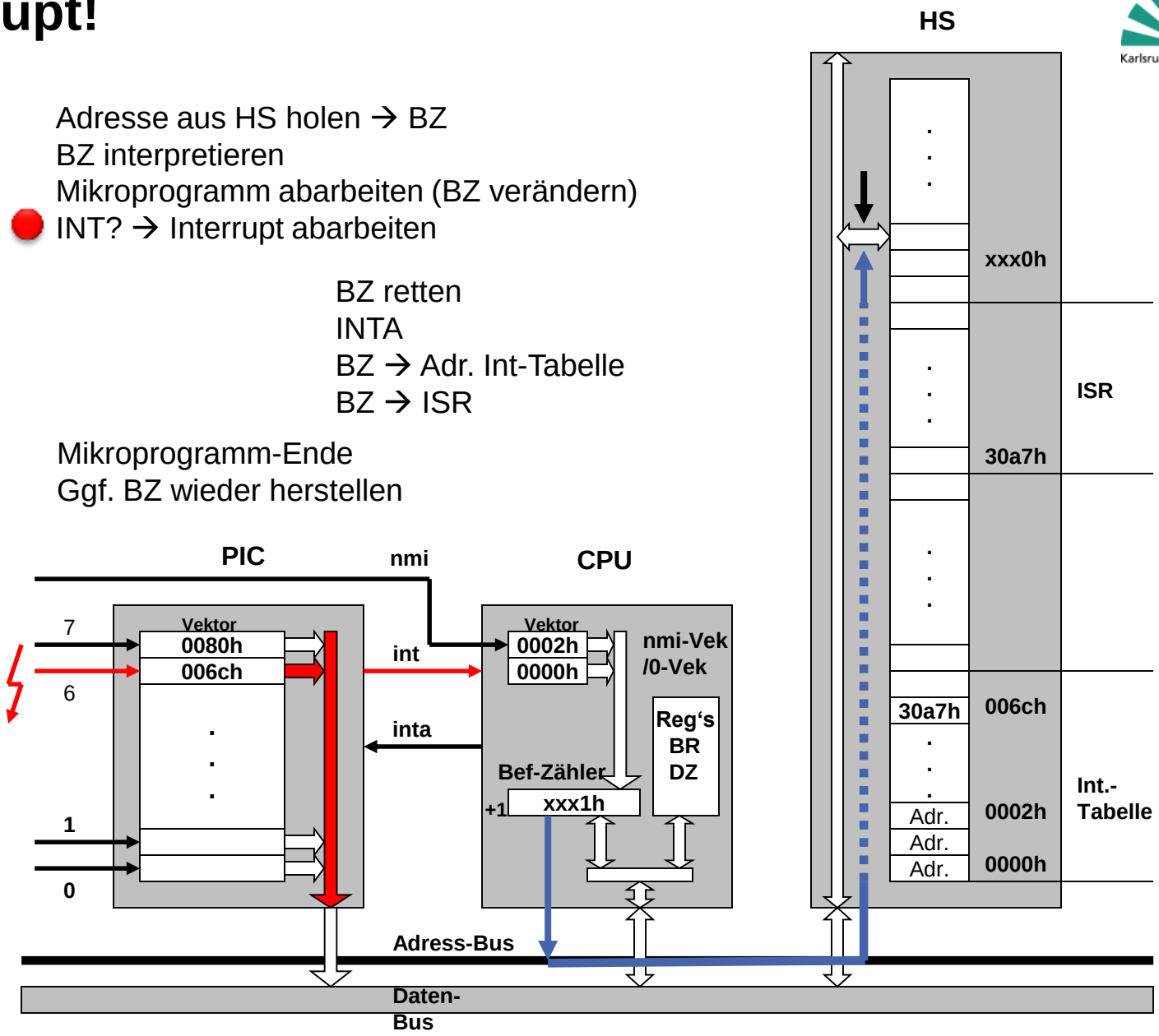
Quelle: http://www.u-r-rennert.de/assembler/rasm/8086_int_prinz.htm

Interrupt!

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
Ggf. BZ wieder herstellen

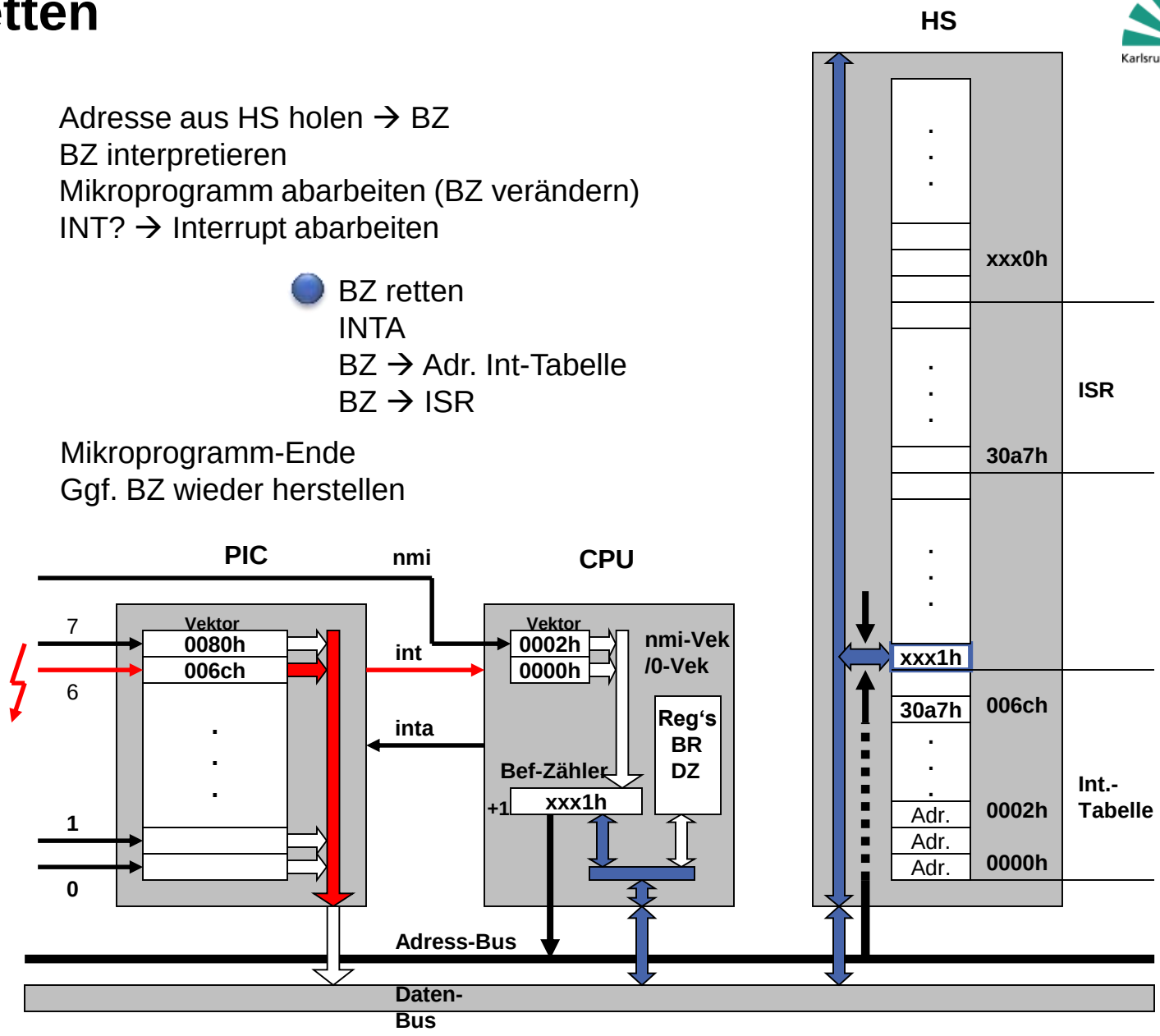


BZ retten

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

- BZ retten
  INTA
  BZ → Adr. Int-Tabelle
  BZ → ISR

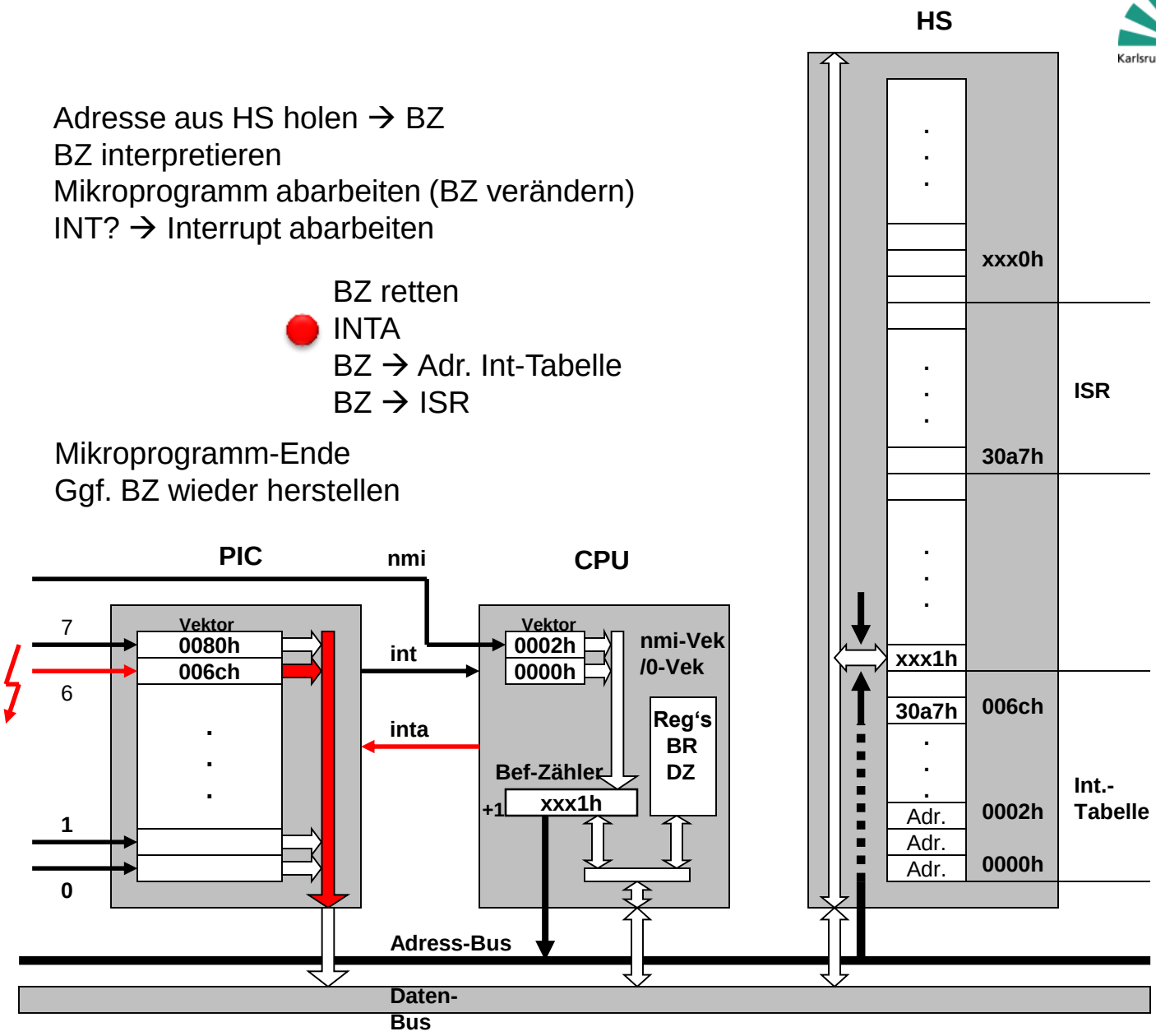
Mikroprogramm-Ende
Ggf. BZ wieder herstellen



Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

- BZ retten
- INTA
- BZ → Adr. Int-Tabelle
- BZ → ISR

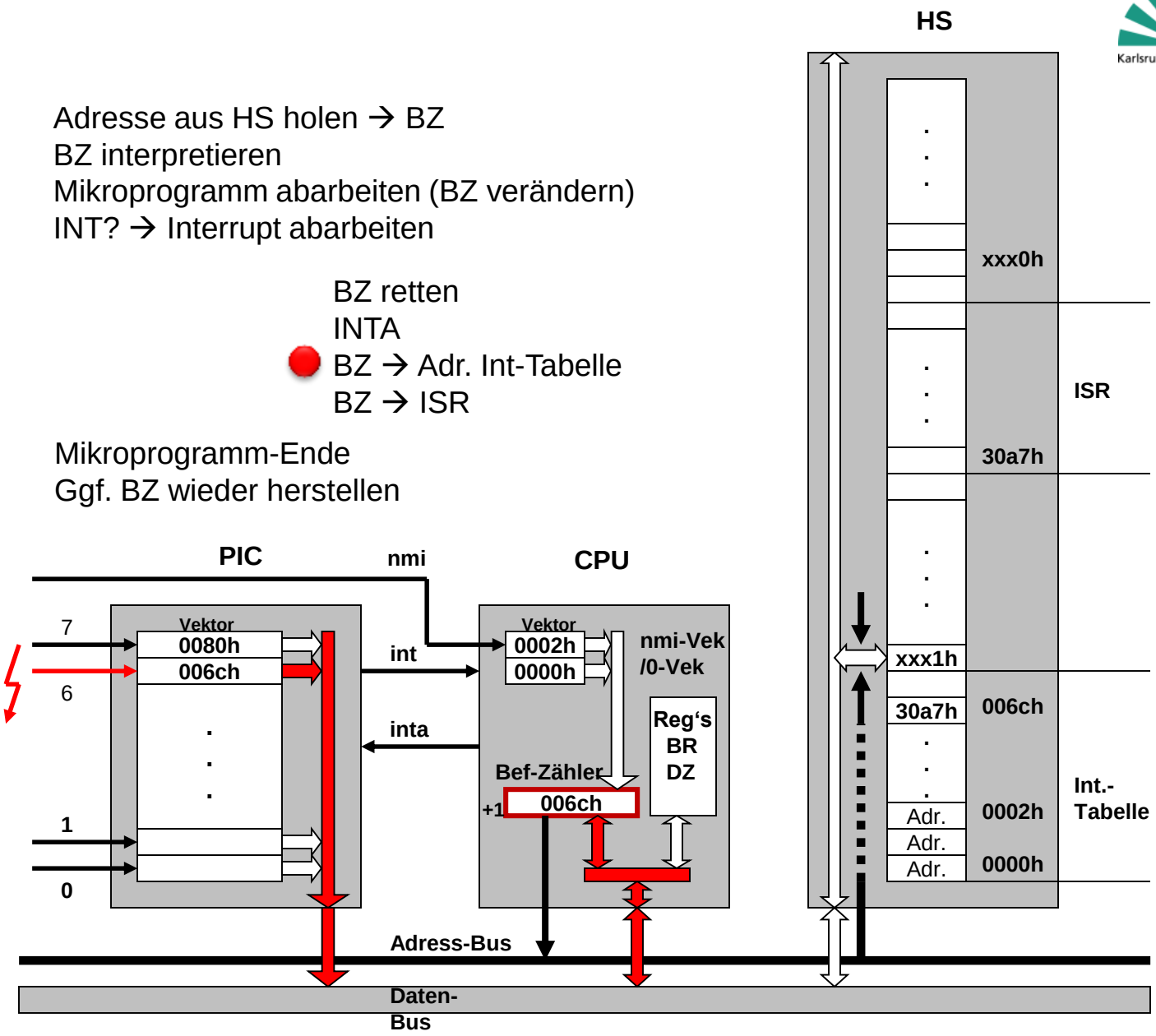
Mikroprogramm-Ende
Ggf. BZ wieder herstellen



Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

- BZ retten
- INTA
- BZ → Adr. Int-Tabelle
- BZ → ISR

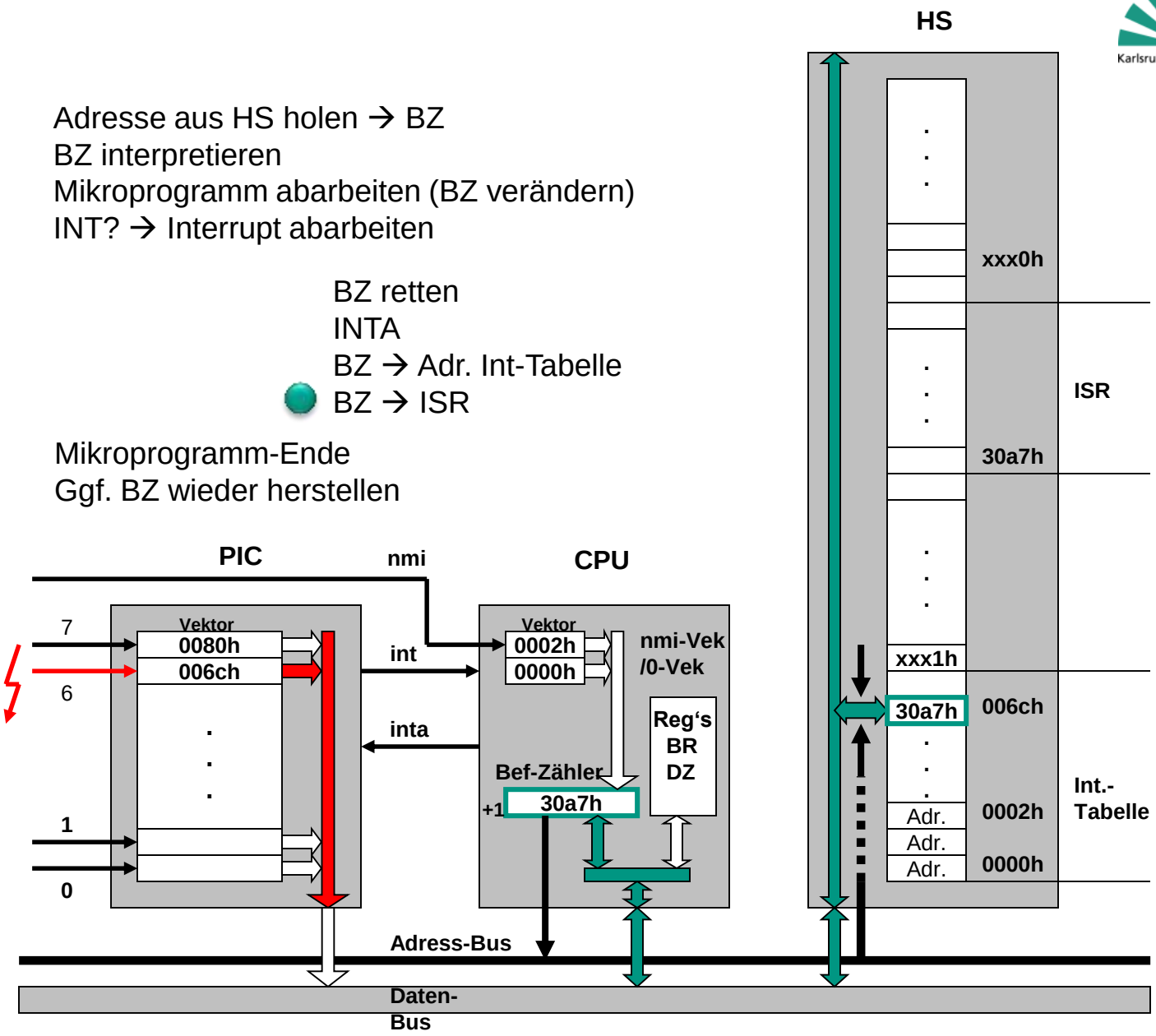
Mikroprogramm-Ende
Ggf. BZ wieder herstellen



Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
Ggf. BZ wieder herstellen

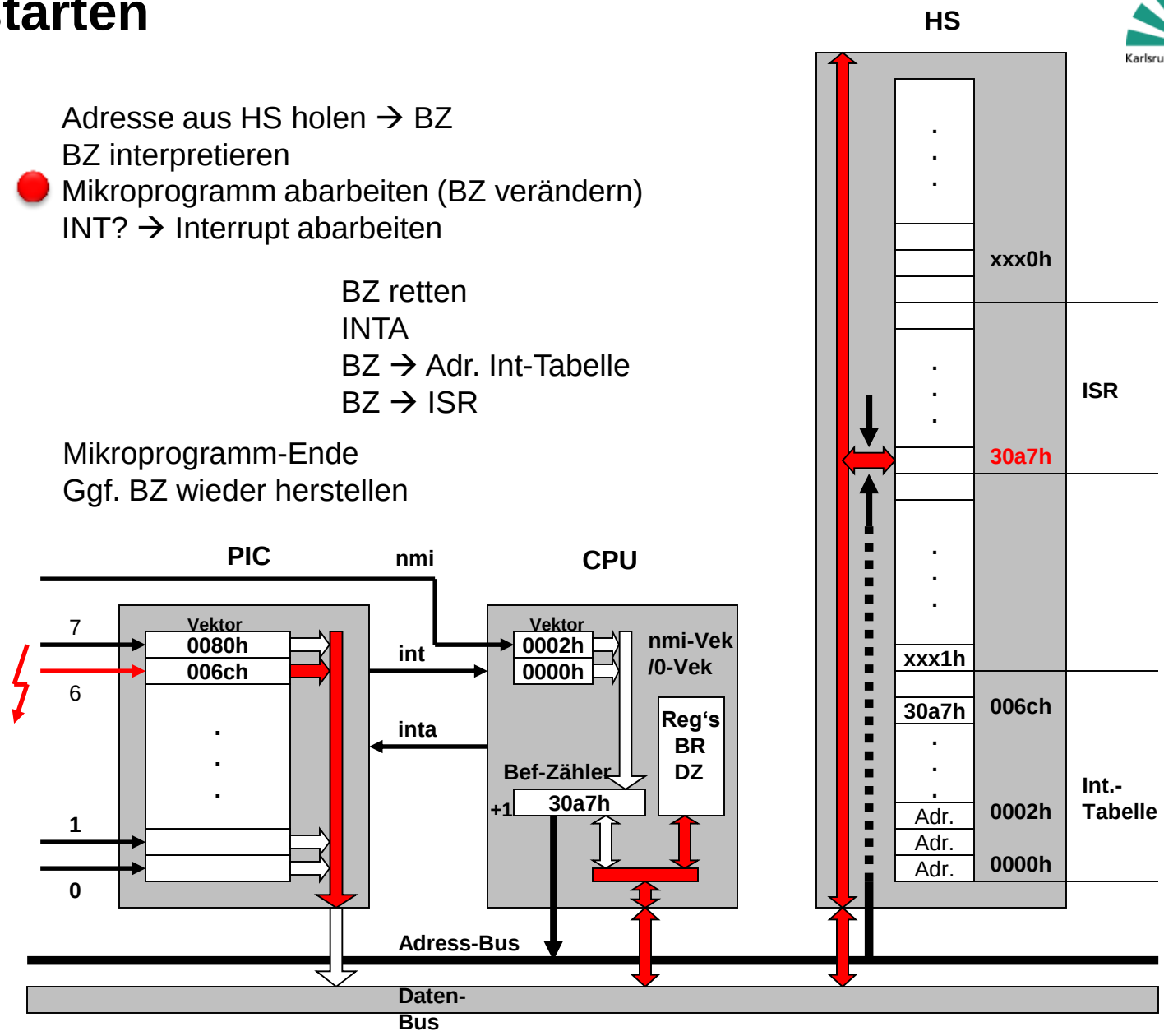


ISR starten

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
Ggf. BZ wieder herstellen

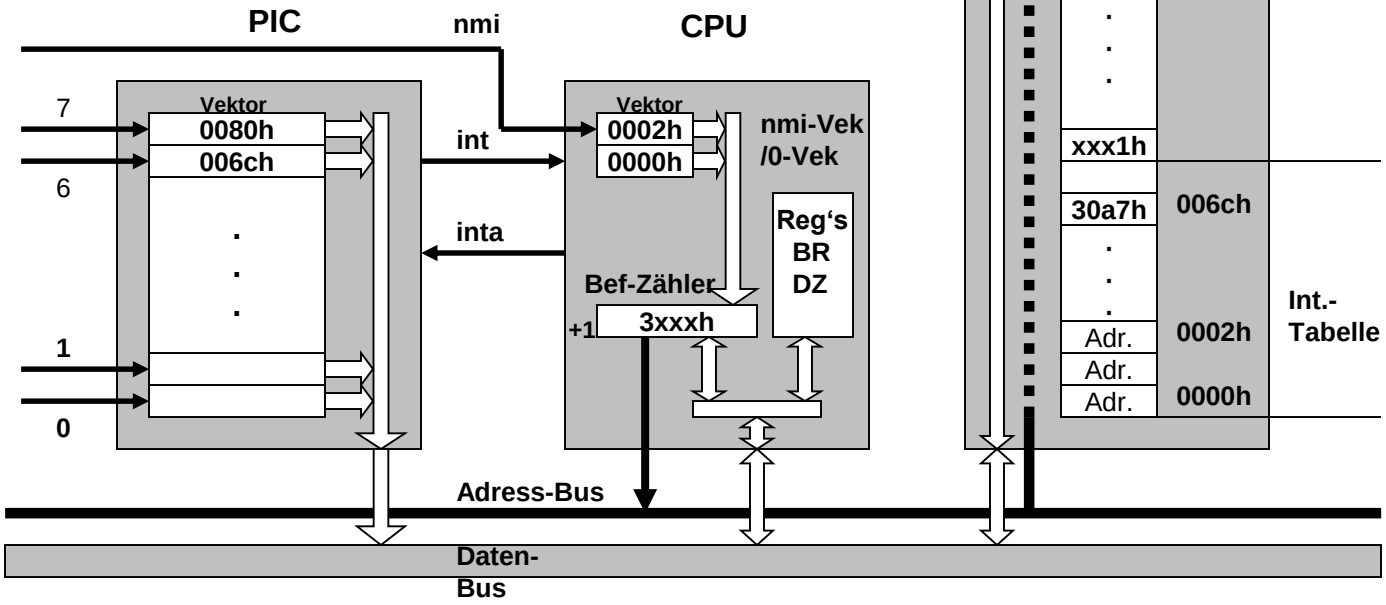


ISR abarbeiten

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
Ggf. BZ wieder herstellen

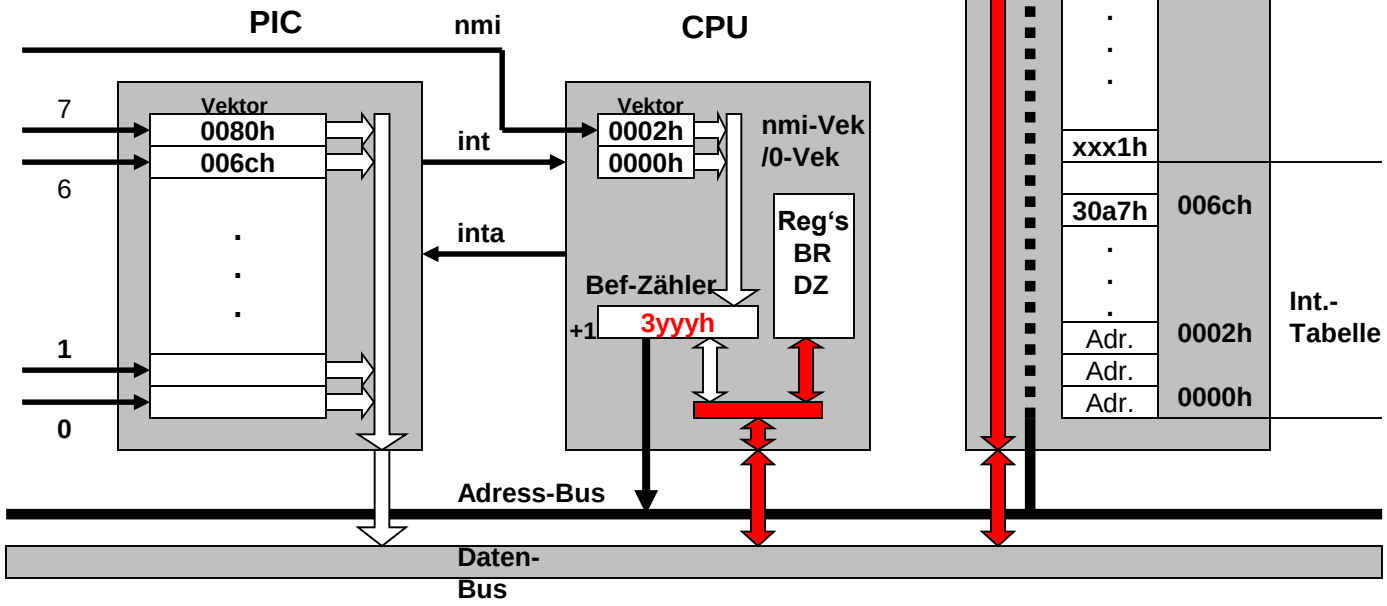


ISR Ende (Return Interrupt / RETI)

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

- Mikroprogramm-Ende
  Ggf. BZ wieder herstellen

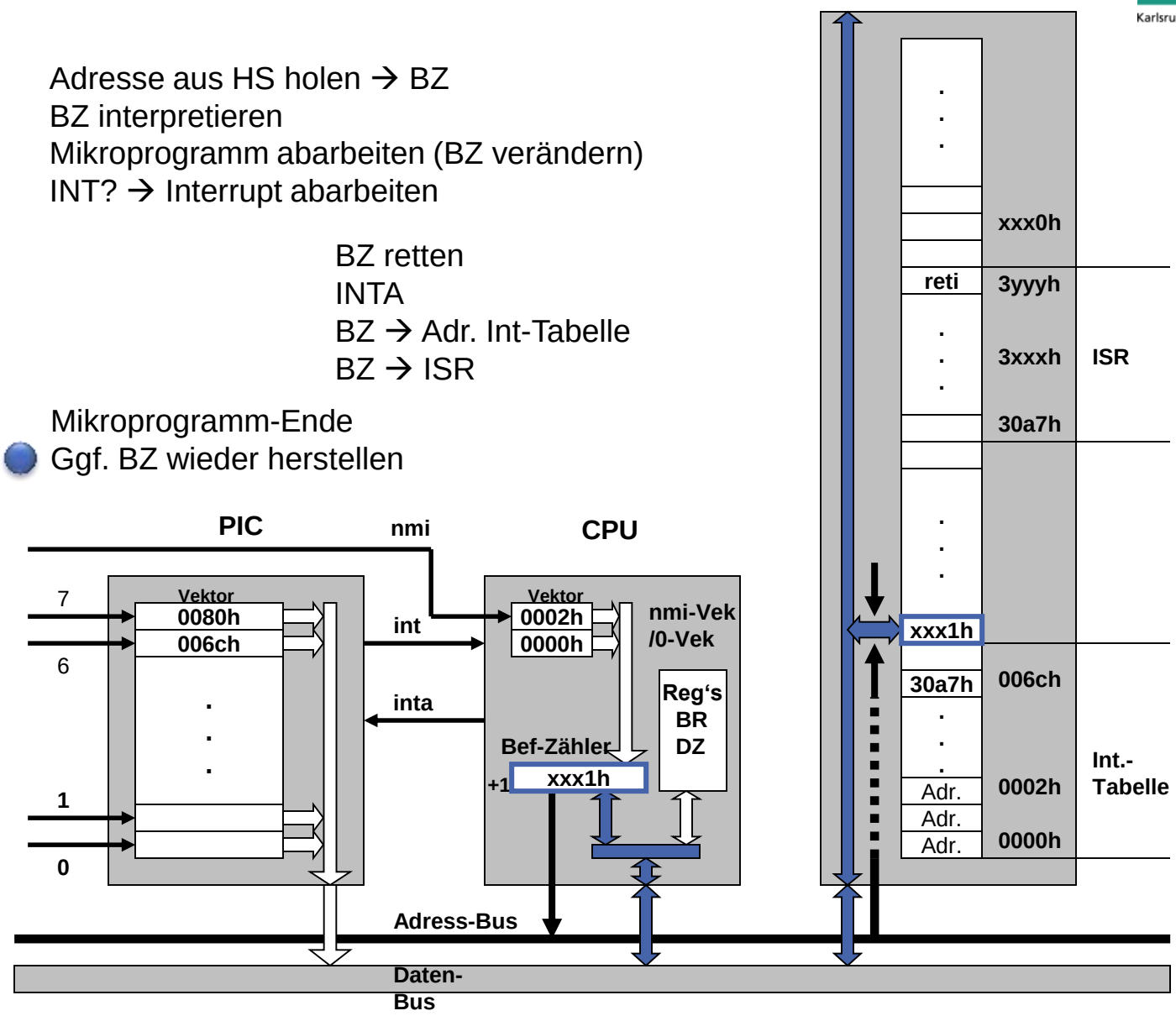


Rücksprung in ursprüngliche Routine

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
- Ggf. BZ wieder herstellen

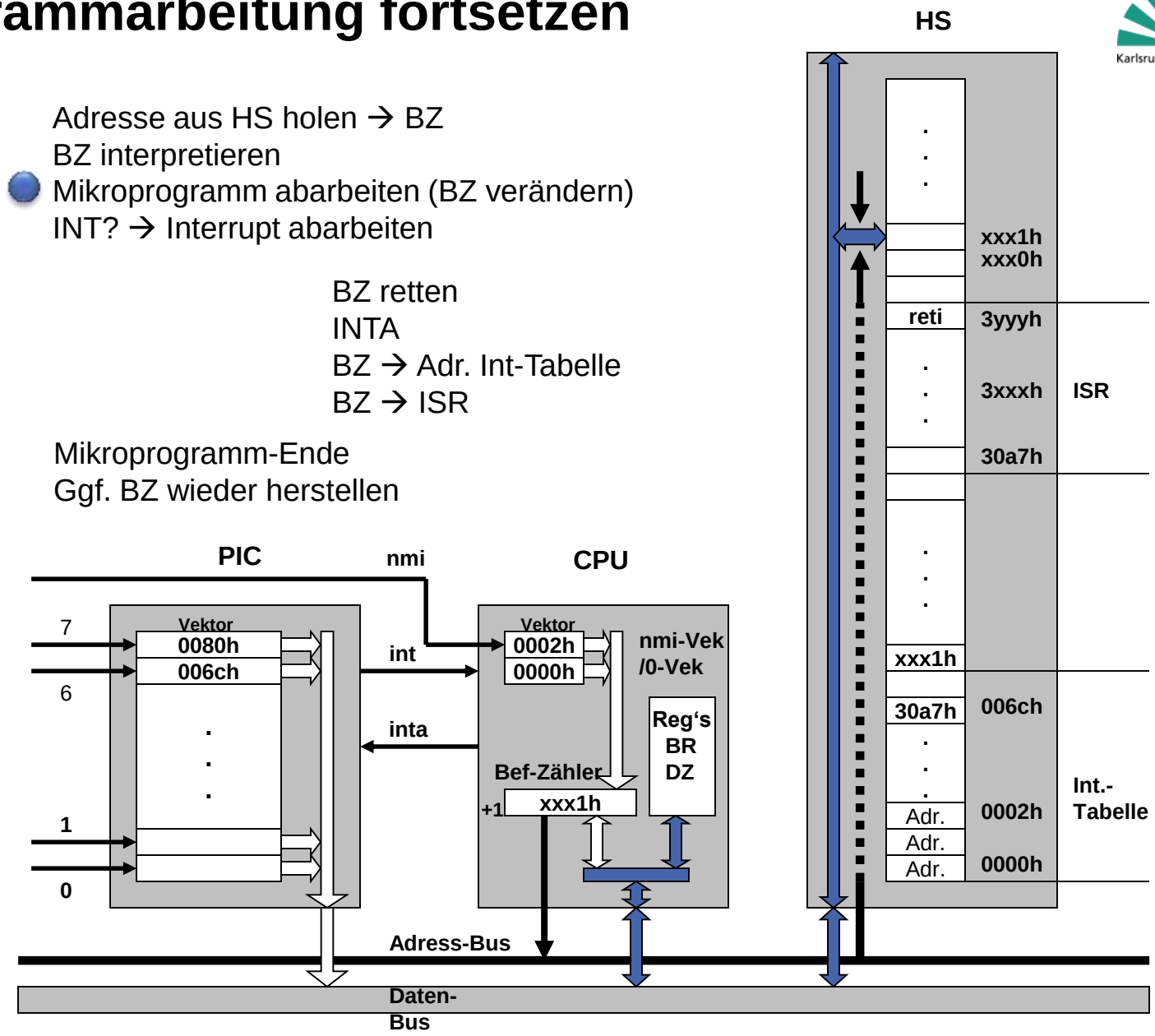


Programmarbeitung fortsetzen

Adresse aus HS holen → BZ
BZ interpretieren
Mikroprogramm abarbeiten (BZ verändern)
INT? → Interrupt abarbeiten

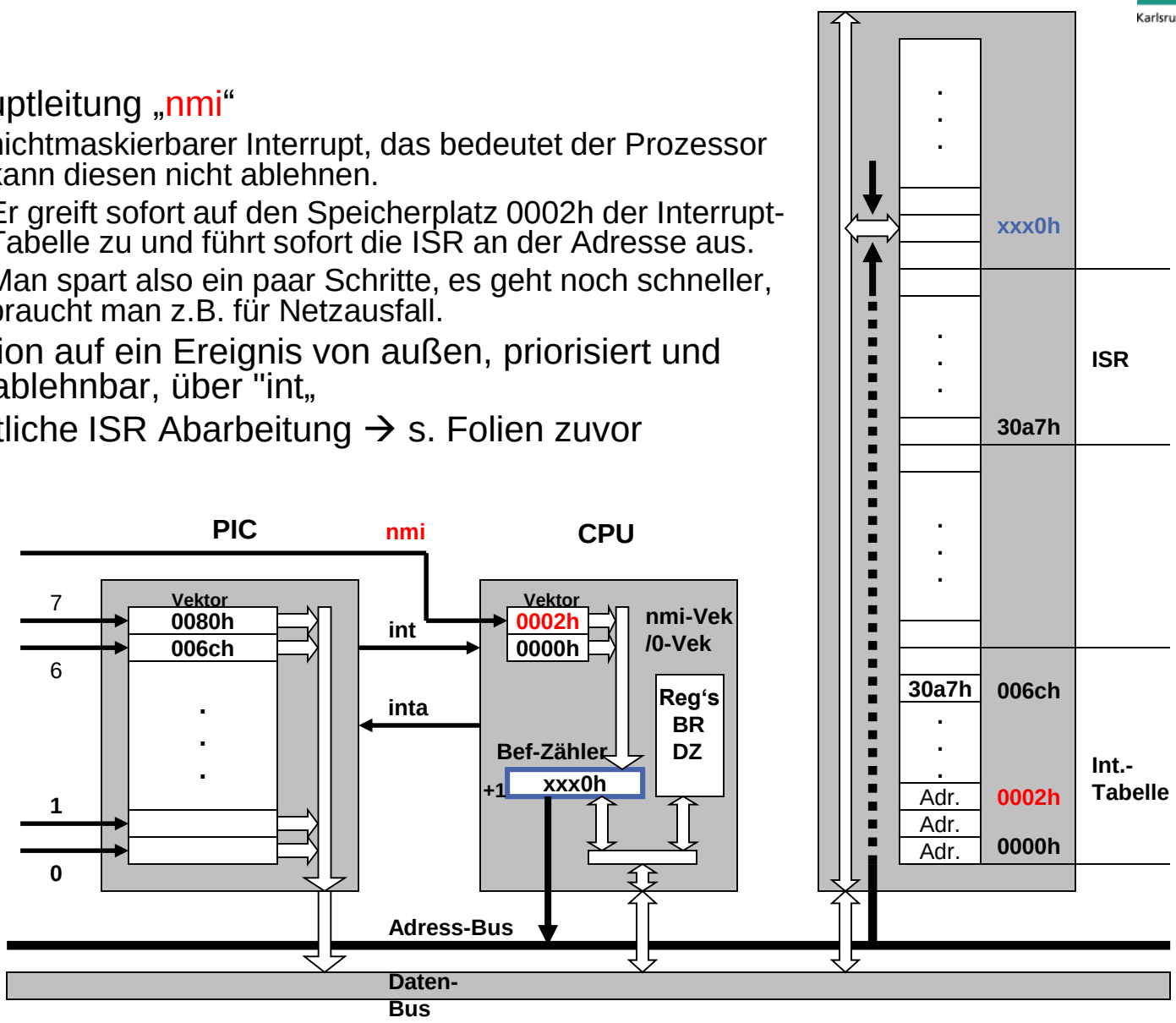
BZ retten
INTA
BZ → Adr. Int-Tabelle
BZ → ISR

Mikroprogramm-Ende
Ggf. BZ wieder herstellen



Beispiel für einfache Abarbeitung *(hier: 8 Bit)*

- Interruptleitung „nmi“
  - nichtmaskierbarer Interrupt, das bedeutet der Prozessor kann diesen nicht ablehnen.
  - Er greift sofort auf den Speicherplatz 0002h der Interrupt-Tabelle zu und führt sofort die ISR an der Adresse aus.
  - Man spart also ein paar Schritte, es geht noch schneller, braucht man z.B. für Netzausfall.
- Reaktion auf ein Ereignis von außen, priorisiert und auch ablehnbar, über "int„
- Eigentliche ISR Abarbeitung → s. Folien zuvor



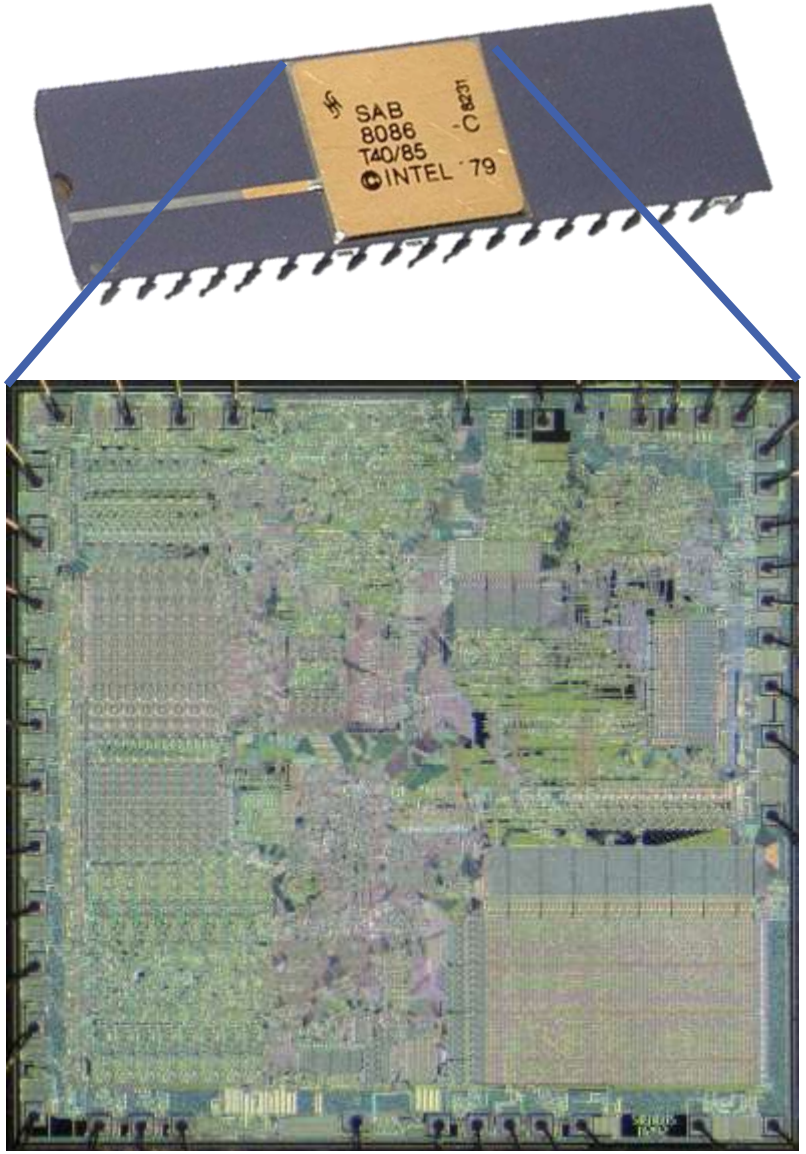
Vektor in PIC identisch Adresse in Interrupt-Tabelle: 256kB Speicheradressierbar

Quelle: http://www.u-r-rennert.de/assembler/rasm/8086_int_prinz.htm

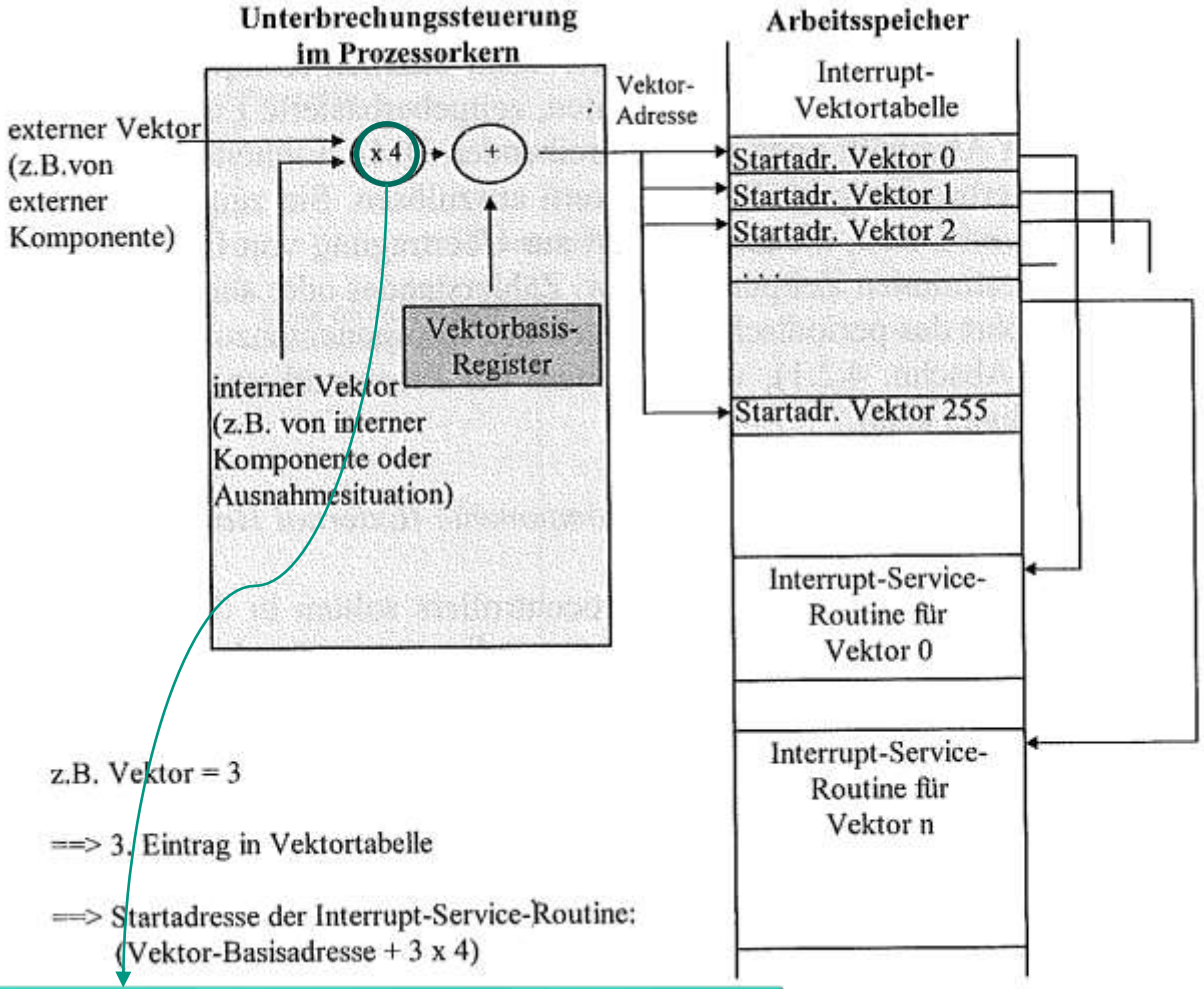
Intel 8086 (16bit)

- Der 8086 ist ein 16-Bit-Mikroprozessor von Intel.
- Er gilt als Urvater der 80x86-Familie.
- Dem 8086 fehlen einige wesentliche Bausteine wie ein Interrupt-Controller, der als externer Chip realisiert ist.
- Wesentliche Kenndaten:
  - Max. adressierbarer Speicher: 1 MB
  - Verarbeitungsbreite: 16 Bit
  - Datenbus: 16 Bit
  - Adressbus: 20 Bit

- Remark:
  - *Der 8086 unterstützt auch keine Gleitkomma-Operationen, kann jedoch von Haus aus mit einem Intel 8087-Koprozessor zusammen-arbeiten, der dann die Gleitkomma-Berechnungen ausführt.*



Beispiel für komplexere Abarbeitung

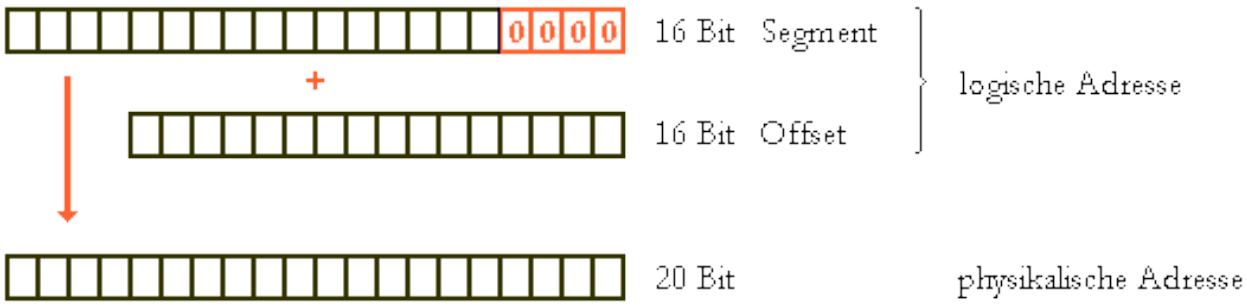


Vektor in PIC wird mit 4 multipliziert und
ergibt die Startadresse der ISR: 1MB Speicher adressierbar

Aus Brinkschulte, Ungerer
„Mikrocontroller, Mikroprozessoren“
Springer-Lehrbuch

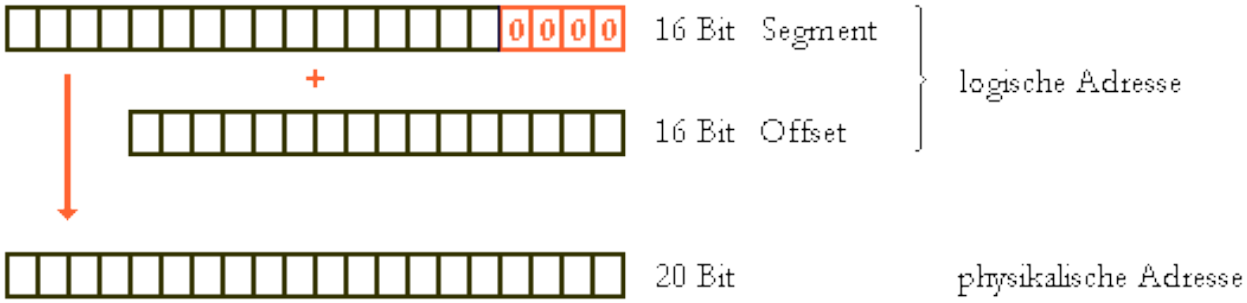
Intel 8086 (16bit → Speichersegmentierung)

- Mit einem 16-Bit-Adressregister ist es nur möglich maximal 64 KB (2^{16}) Speicher zu adressieren.
- Um trotzdem mehr Speicher adressieren zu können, gibt Intel dem Prozessor 20 Adressleitungen für 2^{20} Adressen (1 MB Arbeitsspeicher).
- Um nun eine Adresse zu speichern, werden zwei Register verwendet: ein Segment- und ein Offsetregister.
  - Die Adresse einer physikalischen Speicherstelle berechnet der Prozessor, indem er die Segmentadresse × 16 (4 x links schieben) nimmt und eine Offsetadresse addiert.
  - Allerdings hat der 8086 keine 4-Bit Register, sondern konsequent 16-Bit Register, auch für den Offset.



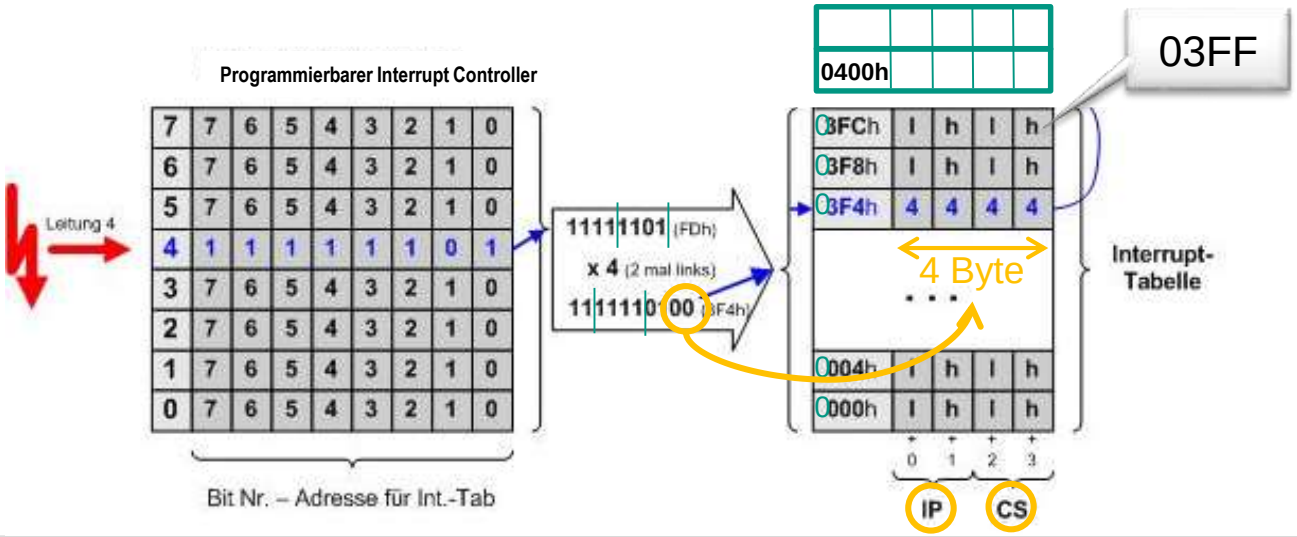
Intel 8086 (16bit → Speichersegmentierung)

- So entstehen im Speicher 65.536 um 16 Byte versetzte, sich überlappende Adressbereiche (Segmente) zu je 64 KB Größe.
- Remark:
  - *Die Überlappung erlaubt unterschiedliche und speicheroptimierte ISR-Größen*
- Wenn beispielsweise das Segmentregister die Adresse 1234h und das Offset-Register die Adresse 5678 enthält, was meist als 1234:5678 geschrieben wird, dann wird im Speicher auf die Adresse 1234h × 10h + 5678 = 12340h + 5678h = 179B8h zugegriffen.
- Auf diese Weise kann der Chip 1 MB adressieren, wobei innerhalb jedes Segments mit 16-Bit-Adressen gearbeitet werden kann.



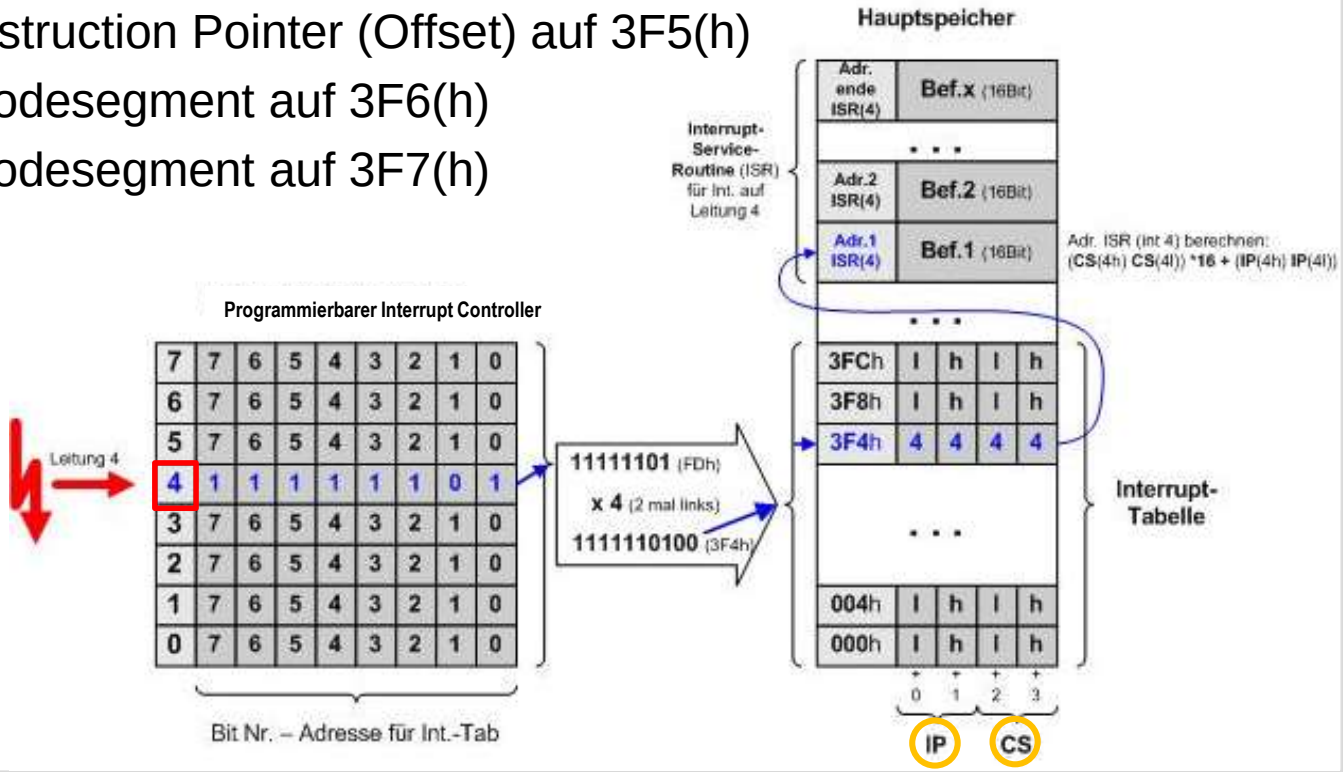
Ermittlung Startadresse einer ISR (8086, 16bit)

- Die Interrupt-Tabelle befindet sich im ersten Kilobyte des Hauptspeichers (0000h-03FFh)
- Mit 8Bit des PIC kann man einen maximalen Wert von 255 erreichen.
- Um die Speicherplätze des Hauptspeichers (1MB) adressieren zu können, wird die Segmentadresse und der Zeiger in dem Segment (Offset) gebraucht.
  - Zum Erreichen einer beliebigen Adresse also 32Bit also 4 Byte (Segment plus Offset).
- Das bedeutet, dass die Adressen immer im Abstand von 4 Byte stehen müssen.
- Dafür wird das Byte aus dem PIC mit 4 multipliziert bzw. 2 mal nach links geschoben.
  - Aus 255 bzw. 11111111 wird 1111111100 bzw. 3FC(h).
    - Von da an 4 Byte weiter erreicht man die Adresse 400(h), die erste Adresse, die nicht durchs Interrupt-System genutzt wird (nicht werden kann!), das sind (dez) genau 1KByte!
  - Im Beispiel wird die Leitung 4 mit 11111101 (FD(h)) belegt, daraus wird als Adresse für die Interrupt-Tabelle die Adresse 1111110100 (3F4(h)) berechnet.



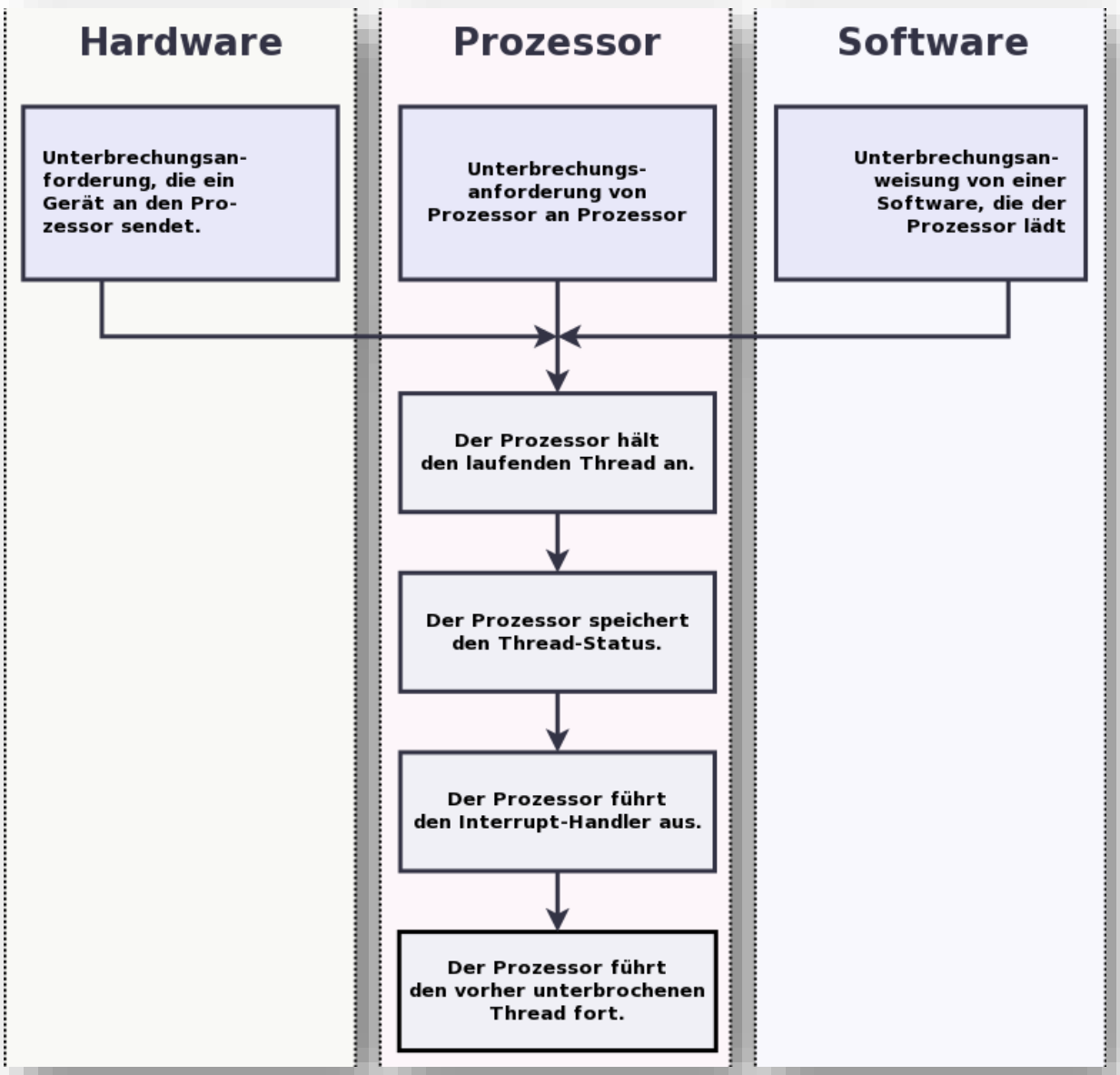
Ermittlung Startadresse ISR durch Interrupt Controller

- In der Interrupt-Tabelle steht nun die Segmentadresse (CS) und ein Zeiger in dem Segment (Instruction Pointer / IP).
- Die ISR-Adresse berechnet sich bei 16bit-Segmentsektor im HS wie folgt: `Code-Segment * 16 + Offset`
  - lo-Byte vom Instruction Pointer (Offset) auf 3F4(h)
  - hi-Byte vom Instruction Pointer (Offset) auf 3F5(h)
  - lo-Byte vom Codesegment auf 3F6(h)
  - hi-Byte vom Codesegment auf 3F7(h)



Unterbrechungsvorgang

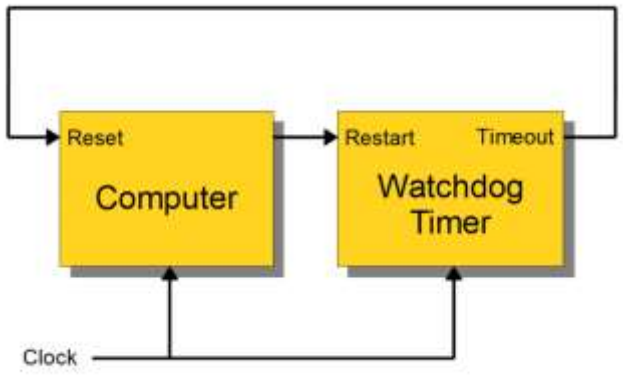
- Ausgehend von 3 möglichen Quellen



Mikrocontroller: Watchdog

- „Wachhund“ zur Überwachung der Programmaktivitäten eines Mikrocontrollers
- Programm muss in regelmäßigen Abständen Lebenszeichen liefern
  - Bleiben diese aus, so nimmt der Wachhund einen Fehler im Programmablauf an → Reset
- Hardware-Watchdog
  - Der Hardware-Watchdog ist in den Mikrocontroller integriert oder ein auf der Platine verbauter Mikroelektronik-Baustein oder eine externe Mikroelektronikkomponente mit Kommunikation zum Mikrocontroller
- Software-Watchdog
  - Der Software-Watchdog ist eine prüfende Software im Mikrocontroller.
  - Das Watchdog-Software-Modul kontrolliert, ob alle wichtigen Programm-Module in einem vorgegebenen Zeitrahmen korrekt ausgeführt werden, oder ob ein Modul unzulässig lange für die Bearbeitung benötigt.

Single Stage Watchdog




Fragen?

IT Inhalt

0. Einleitung und Motivation

- Organisatorisches
- Begriffe
- Typische Anwendungen

1. Rechnerarchitekturen

- Einführung
- Allgemeiner Aufbau der internen Hardware Architektur
- Instruction Set Architecture (ISA) am Beispiel AVR8
- Klassifikation von Rechnerarchitekturen
- Performanzsteigerung
- ausgewählte Beispiele

Prozessoren für eingebettete Systeme

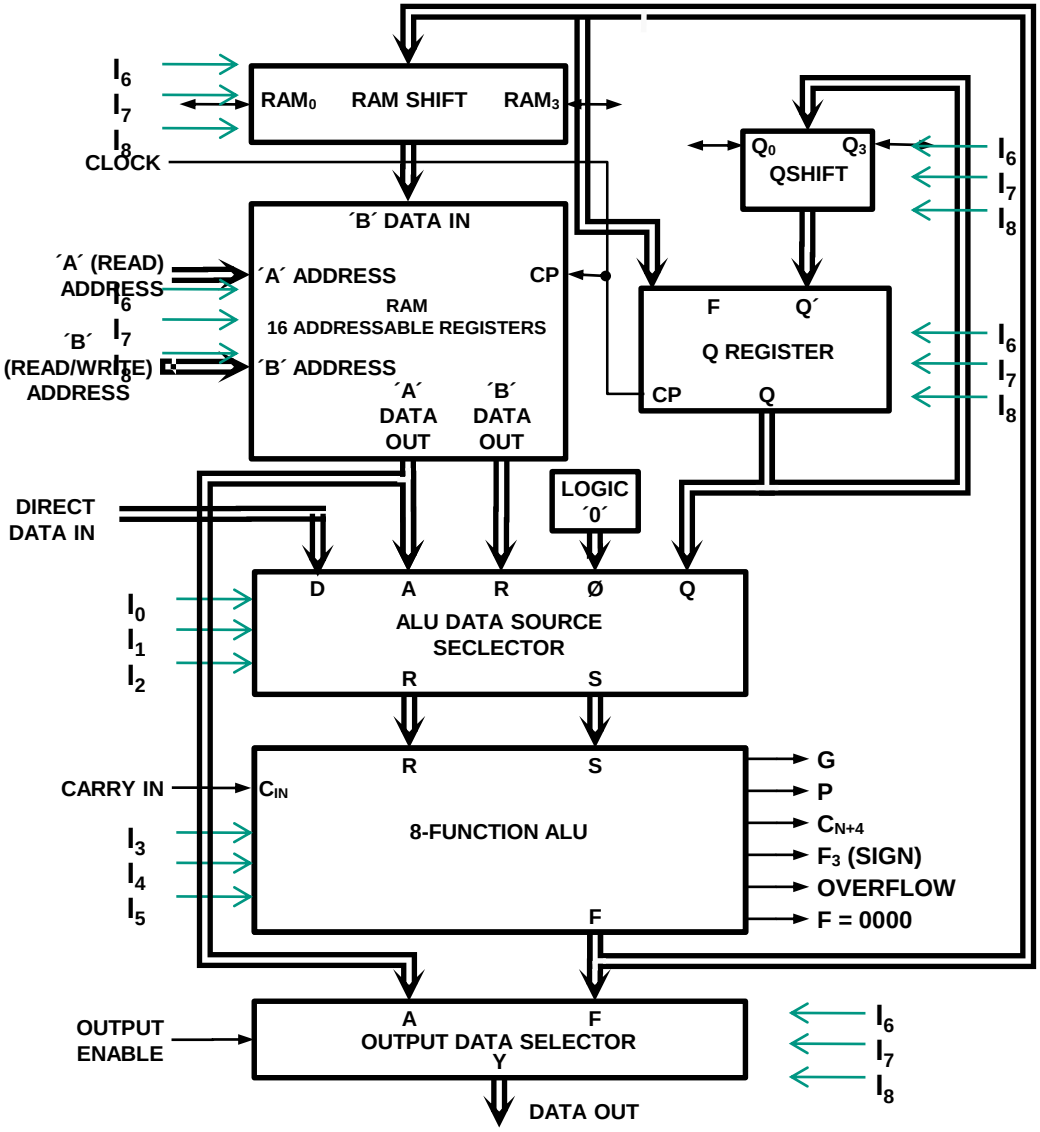
- Anforderungen an Prozessoren sind anders als bei Desktop-/Server-Systemen:
  - Niedriger Stromverbrauch wichtiger als Performance
  - Geringer Energiebedarf, geringe Abwärme
  - Skalierbarkeit (gleicher Kern mit verschiedener Ausstattung)
  - Integrierte I/O-Komponenten ("System on a Chip")
  - Geringere Anforderungen an Kompatibilität
- Gewaltiger Markt: Weit über 90% der Jahresproduktion an Prozessoren
- Breites Spektrum an Technologie und Leistungsfähigkeit
  - Von 4 Bit bis zu mehr als 64 Bit
  - Von weniger als einem KB adressierbaren Speicher bis hin zu vielen TB

4-Bit Arithmetik-Logik Einheit
(AMD 2901 Slice)

GENERAL DESCRIPTION

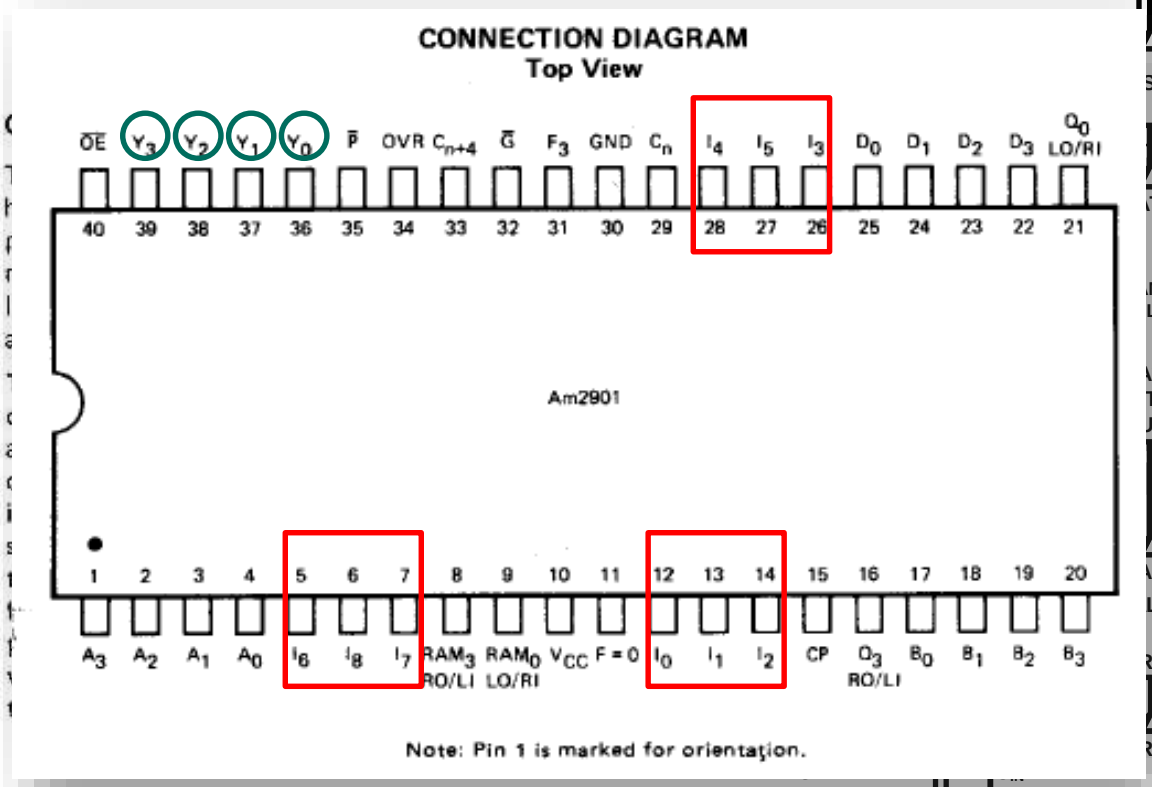
The four-bit bipolar microprocessor slice is designed as a high-speed cascadable element intended for use in CPU's, peripheral controllers, programmable microprocessors and numerous other applications. The microinstruction flexibility of the Am2901 will allow efficient emulation of almost any digital computing machine.

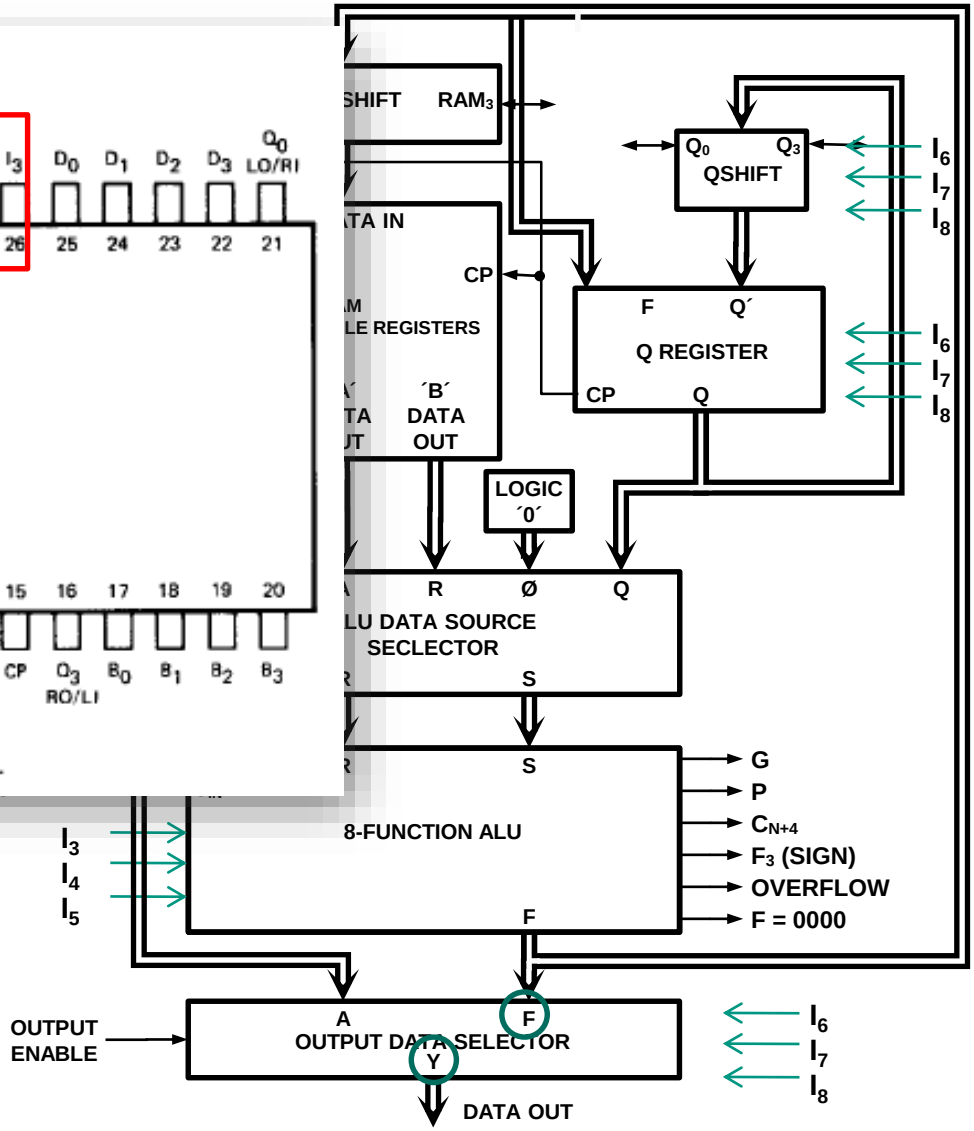
The device, as shown in the block diagram below, consists of a 16-word by 4-bit two-port RAM, a high-speed ALU, and the associated shifting, decoding and multiplexing circuitry. The nine-bit microinstruction word is organized into three groups of three bits each and selects the ALU source operands, the ALU function, and the ALU destination register. The microprocessor is cascadable with full look-ahead or with ripple carry, has three-state outputs, and provides various status flag outputs from the ALU. Advanced low-power Schottky processing is used to fabricate this 40-lead LSI chip.



I = Instruction Control Line

4-Bit Arithmetik-Logik Einheit
(AMD 2901 Slice)





Data Sheet: Pin Description

PIN DEFINITIONS

A_{0-3} The four address inputs to the register stack used to select one register whose contents are displayed through the A-port.

B_{0-3} The four address inputs to the register stack used to select one register whose contents are displayed through the B-port and into which new data can be written when the clock goes LOW.

I_{0-8} The nine instruction control lines to the Am2901, used to determine what data sources will be applied to the ALU (I_{012}), what function the ALU will perform (I_{345}), and what data is to be deposited in the Q-register or the register stack (I_{678}).

RO/LI A shift line at the MSB of the Q register (Q_3RO/LI) and the register stack (RAM_3RO/LI). Electrically these lines are three-state outputs connected to TTL inputs internal to the Am2901. When the destination code on I_{678} indicates a right shift (octal 6 or 7) the three-state outputs are enabled and the MSB of the Q register is available on the Q_3RO/LI pin and the MSB of the ALU output is available on the RAM_3RO/LI pin. Otherwise, the three-state outputs are OFF (high-impedance) and the pins are electrically LS-TTL inputs. When the destination code calls for a down (left) shift, the pins are used as the data inputs to the MSB of the Q register (octal 4) and RAM (octal 4 or 5).

LO/RI Shift lines like RO/LI, but at the LSB of the Q-register and RAM. These pins are tied to the RO/LI pin of the adjacent device to transfer data between devices for left and right shifts of the Q register and ALU data.

D_{0-3} Direct data inputs. A four-bit data field which may be selected as one of the ALU data sources for entering data into the Am2901. D_0 is the LSB.

Y_{0-3} The four data outputs of the Am2901. These are three-state output lines. When enabled, they display either the four outputs of the ALU or the data on the A-port of the register stack, as determined by the destination code I_{678}.

$\overline{OE}$ Output Enable. When $\overline{OE}$ is HIGH, the Y outputs are OFF; when $\overline{OE}$ is LOW, the Y outputs are active (HIGH or LOW).

$\overline{P}$, $\overline{G}$ The carry generate and propagate outputs of the Am2901's ALU. These signals are used with the Am2902 for carry-lookahead. See Figure 8 for the logic equations.

OVR Overflow. This pin is logically the Exclusive-OR of the carry-in and carry-out of the MSB of the ALU. At the most significant end of the word, this pin indicates that the result of an arithmetic two's complement operation has overflowed into the sign-bit. See Figure 8 for logic equation.

F = 0 This is an open collector output which goes HIGH (OFF) if the data on the four ALU outputs F_{0-3} are all LOW. In positive logic, it indicates the result of an ALU operation is zero.

C_n The carry-in to the Am2901's ALU.

C_{n+4} The carry-out of the Am2901's ALU. See Figure 8 for equations.

CP The clock to the Am2901. The Q register and register stack outputs change on the clock LOW-to-HIGH transition. The clock LOW time is internally the write enable to the 16 x 4 RAM which comprises the "master" latches of the register stack. While the clock is LOW, the "slave" latches on the RAM outputs are closed, storing the data previously on the RAM outputs. This allows synchronous master-slave operation of the register stack.

Data Sheet: Functional Tables

Am2901B/Am

Mnemonic	MICRO CODE				ALU SOURCE OPERANDS	
	I_2	I_1	I_0	Octal Code	R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	0	Q
ZB	L	H	H	3	0	B
ZA	H	L	L	4	0	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	0

Figure 3. ALU Source Operand Control.

Mnemonic	MICRO CODE				ALU Function	SYMBOL
	I_5	I_4	I_3	Octal Code		
ADD	L	L	L	0	R Plus S	R + S
SUBR	L	L	H	1	S Minus R	S − R
SUBS	L	H	L	2	R Minus S	R − S
OR	L	H	H	3	R OR S	R ∨ S
AND	H	L	L	4	R AND S	R ∧ S
NOTRS	H	L	H	5	$\overline{R}$ AND S	$\overline{R} \wedge S$
EXOR	H	H	L	6	R EX-OR S	$R \overline{\vee} S$
EXNOR	H	H	H	7	R EX-NOR S	$\overline{R \overline{\vee} S}$

Figure 4. ALU Function Control.

Mnemonic	MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
	I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load		RAM_0	RAM_3	Q_0	Q_3
QREG	L	L	L	0	X	NONE	NONE	F→Q	F	X	X	X	X
NOP	L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
RAMA	L	H	L	2	NONE	F→B	X	NONE	A	X	X	X	X
RAMF	L	H	H	3	NONE	F→B	X	NONE	F	X	X	X	X
RAMQD	H	L	L	4	DOWN	F/2→B	DOWN	Q/2→Q	F	F_0	IN_3	Q_0	IN_3
RAMD	H	L	H	5	DOWN	F/2→B	X	NONE	F	F_0	IN_3	Q_0	X
RAMQU	H	H	L	6	UP	2F→B	UP	2Q→Q	F	IN_0	F_3	IN_0	Q_3
RAMU	H	H	H	7	UP	2F→B	X	NONE	F	IN_0	F_3	X	Q_3

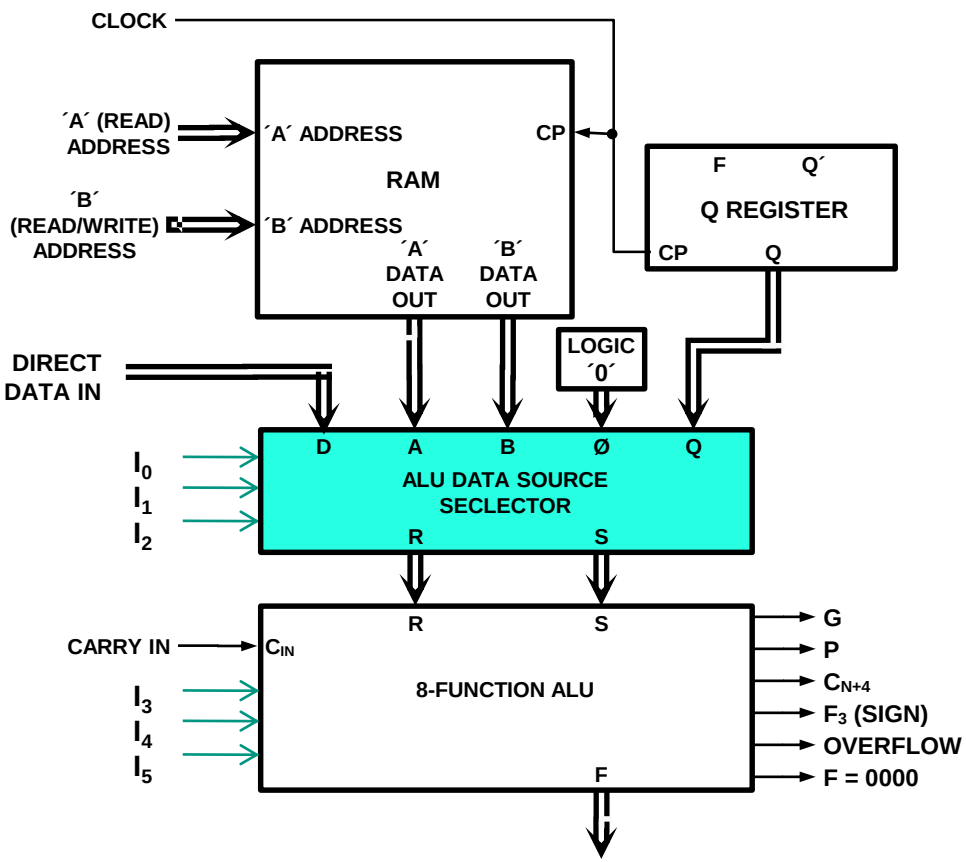
X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.
B = Register Addressed by B inputs.
UP is toward MSB, DOWN is toward LSB.

Figure 5. ALU Destination Control.

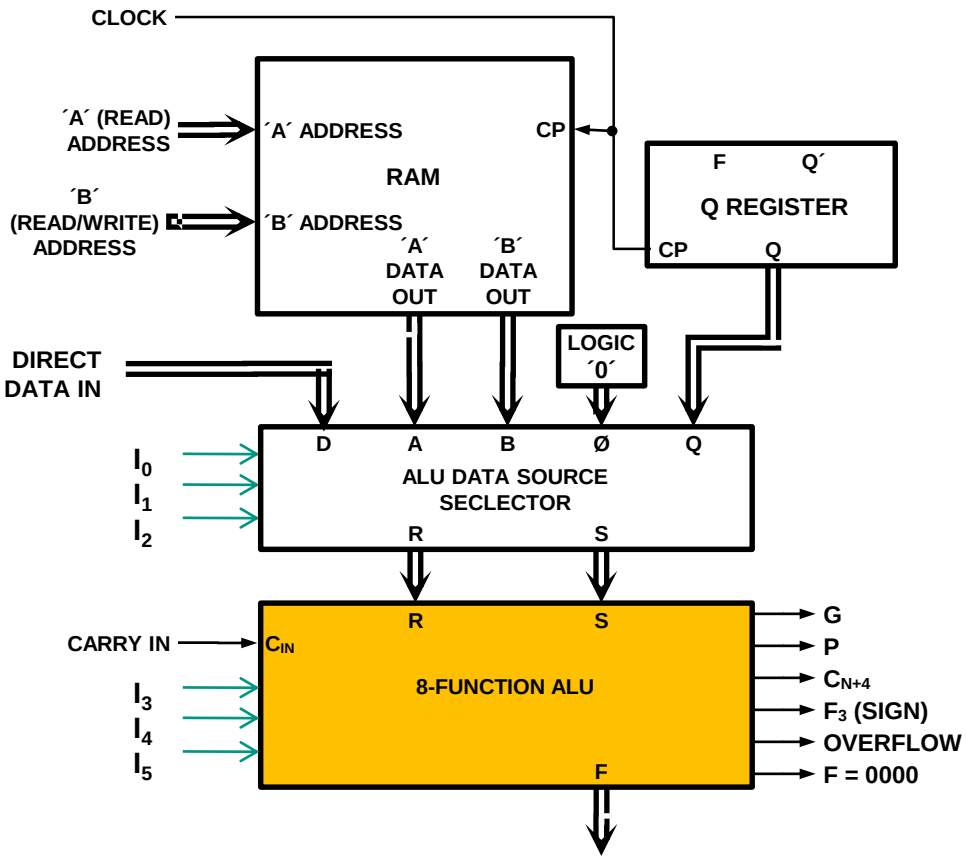
4-Bit Arithmetik-Logik-Einheit
(AMD 2901 ALU Slice)

	Micro Code				ALU Source Operands	
Mnemonic	I_2	I_1	I_0	Octal Code	R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	0	Q
ZB	L	H	H	3	0	B
ZA	H	L	L	4	0	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	0

ALU Source Control



4-Bit Arithmetik-Logik-Einheit
(AMD 2901 ALU Slice)

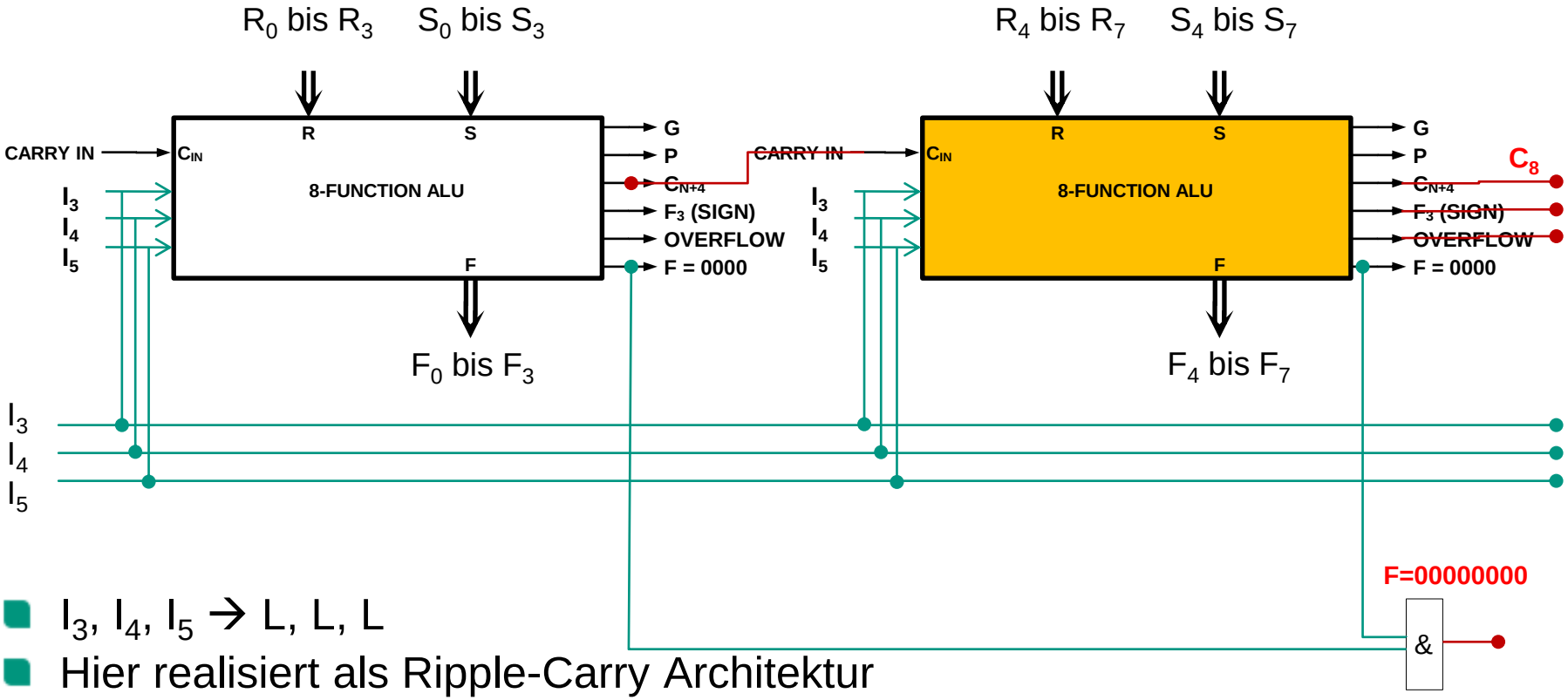


ALU Function Control

	Micro Code				ALU Function	Symbol
Mnemonic	I_5	I_4	I_3	Octal Code		
ADD	L	L	L	0	R Plus S	R + S
SUBR	L	L	H	1	S Minus R	S - R
SUBS	L	H	L	2	R Minus S	R - S
OR	L	H	H	3	R OR S	$R \vee S$
AND	H	L	L	4	R AND S	$R \wedge S$
NOTRS	H	L	H	5	R AND S	$\overline{R} \wedge S$
EXOR	H	H	L	6	R EXOR S	$R \veebar S$
EXNOR	H	H	H	7	R E NOR S	$\overline{R \veebar S}$

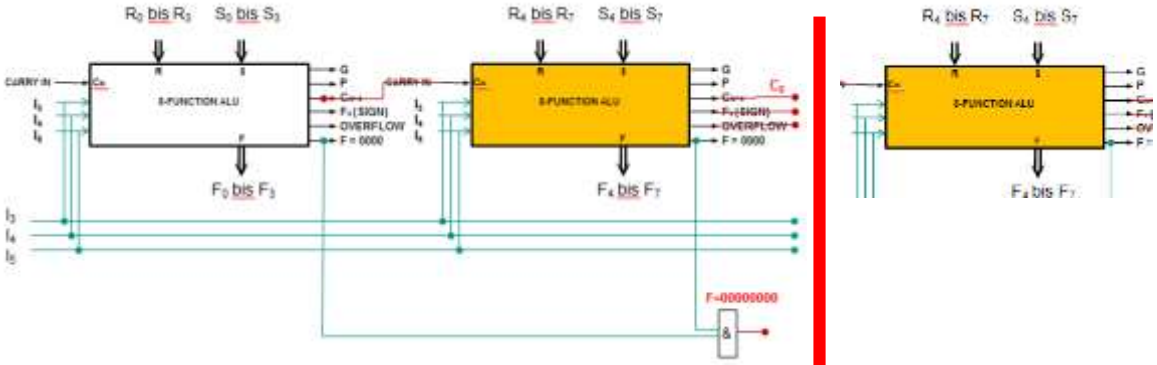
8-Bit Arithmetik-Logik-Einheit
(2 mal AMD 2901 ALU Slice)

- *Bit-Slicing* bezeichnete eine Methode aus der Rechnerarchitektur, bei der man aus mehreren Einzelbausteinen, die oft alle für relativ kleine Wörter – den Bit-Slices – (meist 4 bit lang) ausgelegt waren, größere Rechenwerke zusammenbaut.



- I_3, I_4, I_5 → L, L, L
- Hier realisiert als Ripple-Carry Architektur
  - zusammengesetzt aus 2 Volladierern

8-Bit Arithmetik-Logik-Einheit
(2 mal AMD 2901 ALU Slice)



Lo bit ALU	Hi bit ALU		1 ALU
1110	1010		1110
0111	1110		0111

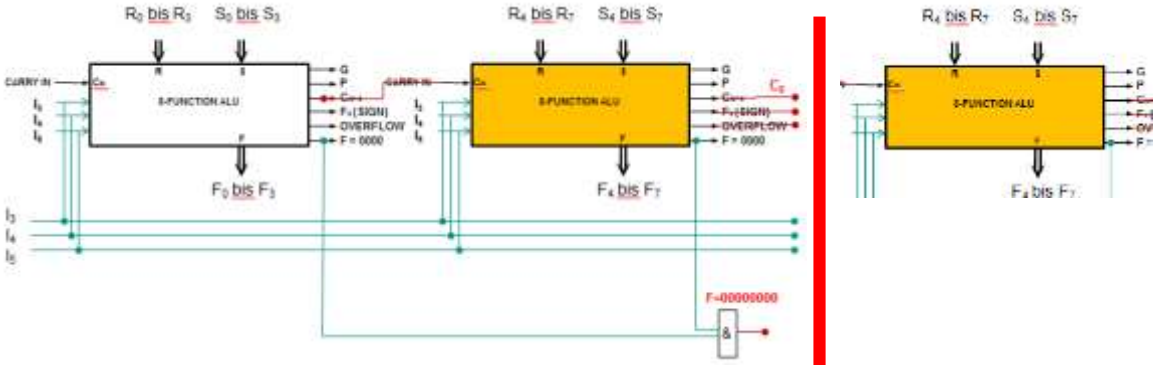
Beispiel:

1010 1110
\+ *1110 0111*
1 1
= *1001 0101*

Performanzsteigerung durch Parallelisierung (Superskalar):

2 Schritte vs. 3 Schritte

8-Bit Arithmetik-Logik-Einheit
(2 mal AMD 2901 ALU Slice)



Beispiel:

1010 1110
\+ *1110 0111*
1 1
= *1001 0101*

	Lo bit ALU	Hi bit ALU		1 ALU	
	1110	1010		1110	
	0111	1110		0111	
Akku 1 ←, Carry	1+0101	1+1000		1+0101	→ Register (lo bit)
		+1		1010	
Akku 2 ←, Carry		1+1001		1110	
				1+1000	
				+1	← Carry
				1+1001	→ Carry, Register (hi bit)

Performanzsteigerung durch Parallelisierung (Superskalar):

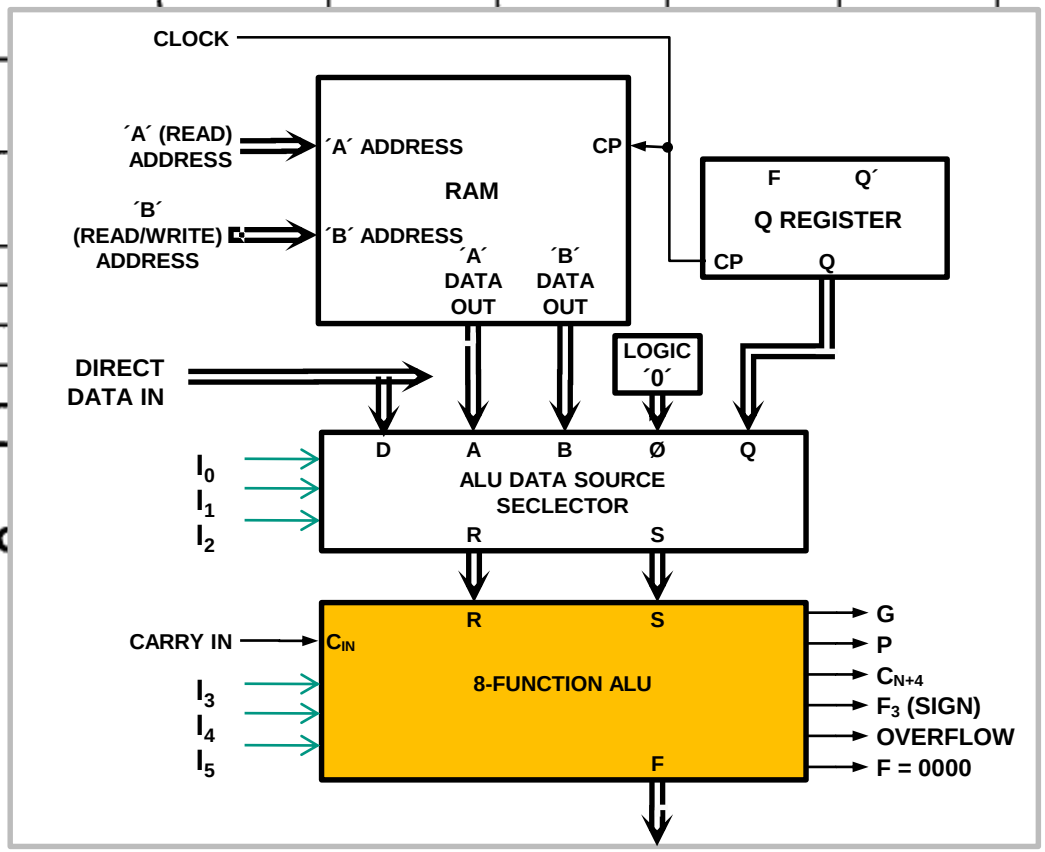
2 Schritte vs. 3 Schritte

ALU Function Matrix

		I_{210} OCTAL							
		0	1	2	3	4	5	6	7
OCTAL I_{543}	ALU Function	ALU Source							
		A,Q	A,B	0,Q	0,B	O,A	D,A	D,Q	D,0
0	C_n = L R Plus S C_n = H	A + Q A + Q + 1	A + B A + B + 1	Q	B	A	D + A	D + Q	D
1	C_n = L S Minus R C_n = H	Q − A − 1 Q − A	B − A − 1 B − A						
2	C_n = L R Minus S C_n = H	A − Q − 1 A − Q	A − B − 1 A − B						
3	R OR S	A∨Q	A∨B						
4	R AND S	A∧Q	A∧B						
5	$\overline{R}$ AND S	$\overline{A}$∧Q	$\overline{A}$∧B						
6	R EX-OR S	A∇Q	A∇B						
7	R EX-NOR S	$\overline{A \nabla Q}$	$\overline{A \nabla B}$						

+ = Plus; − = Minus; ∨ = OR; ∧ = AND; ∇ = EX-OR

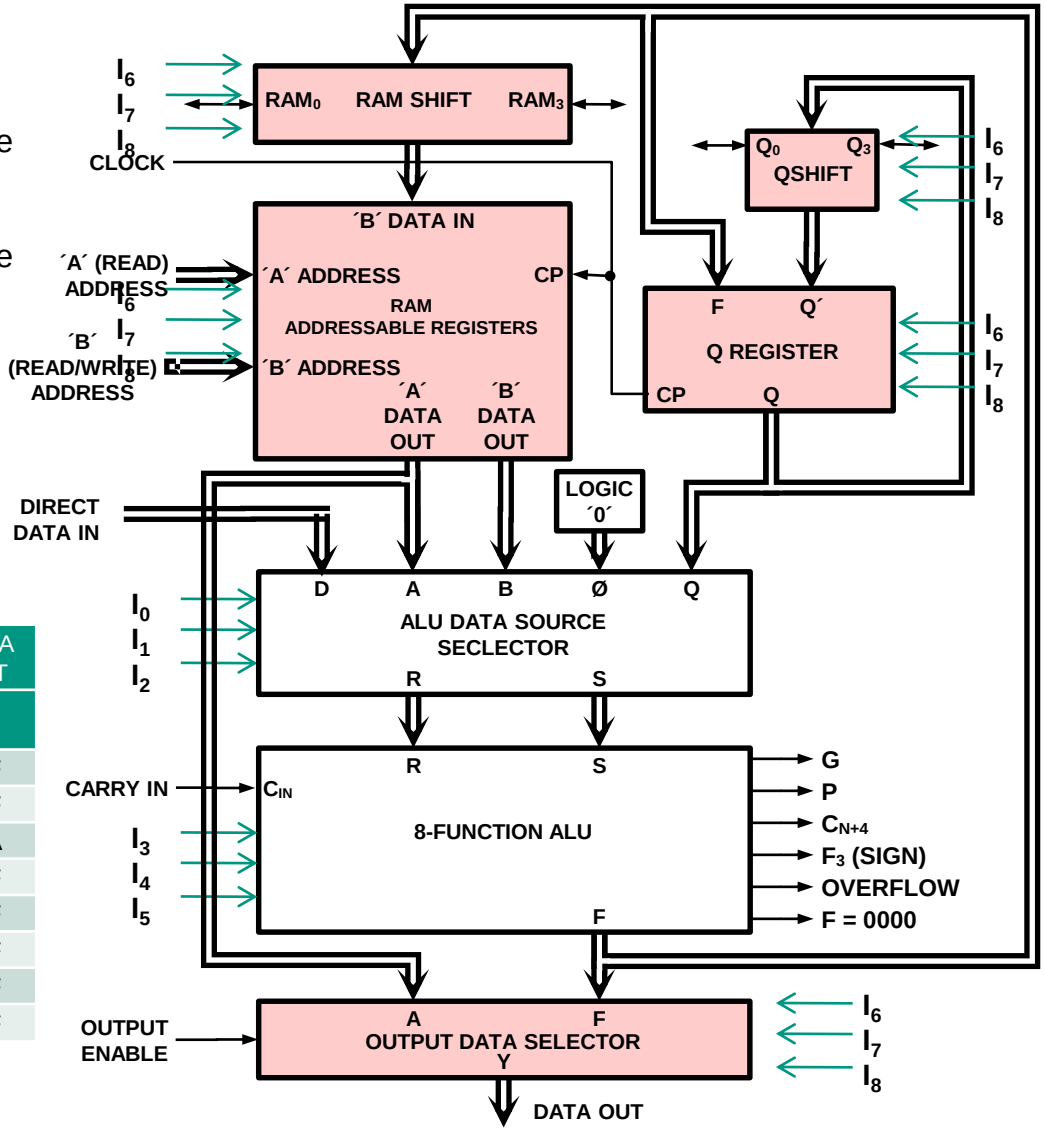
Figure 6. Source (



4-Bit Arithmetik-Logik-Einheit
(AMD 2901 ALU Slice)

Beispiel:

- RAMQD: RAM- und Q-Register werden um 1 Stelle in Richtung LSB (least significant Bit) verschoben, D für downshift
- RAMQU: RAM- und Q-Register werden um 1 Stelle in Richtung MSB (most significant Bit) verschoben, U für upshift



ALU Destination Control

	Micro Code				RAM Function	Q-Reg Function	DATA OUT
Mnemonic	I_8	I_7	I_6	Octal Code	Load	Load	
QREG	L	L	L	0	NONE	F into Q	F
NOP	L	L	H	1	NONE	NONE	F
RAMA	L	H	L	2	F into B	NONE	A
RAMF	L	H	H	3	F into B	NONE	F
RAMQD	H	L	L	4	F/2 into B	Q/2 into Q	F
RAMD	H	L	H	5	F/2 into B	NONE	F
RAMQU	H	H	L	6	2F into B	2Q into Q	F
RAMU	H	H	H	7	2F into B	NONE	F

Zsf.: 4-Bit Arithmetik-Logik-Einheit

ALU Steuersignale vom Steuerwerk:

8	7	6	5	4	3	2	1	0
DESTINATION CONTROL			ALU FUNCTION			ALU SOURCE		

I_6 I_3 I_0
I_7 I_4 I_1
I_8 I_5 I_2

CARRY IN

DIRECT DATA IN

´A´ (READ) ADDRESS

´B´ (READ/WRITE) ADDRESS

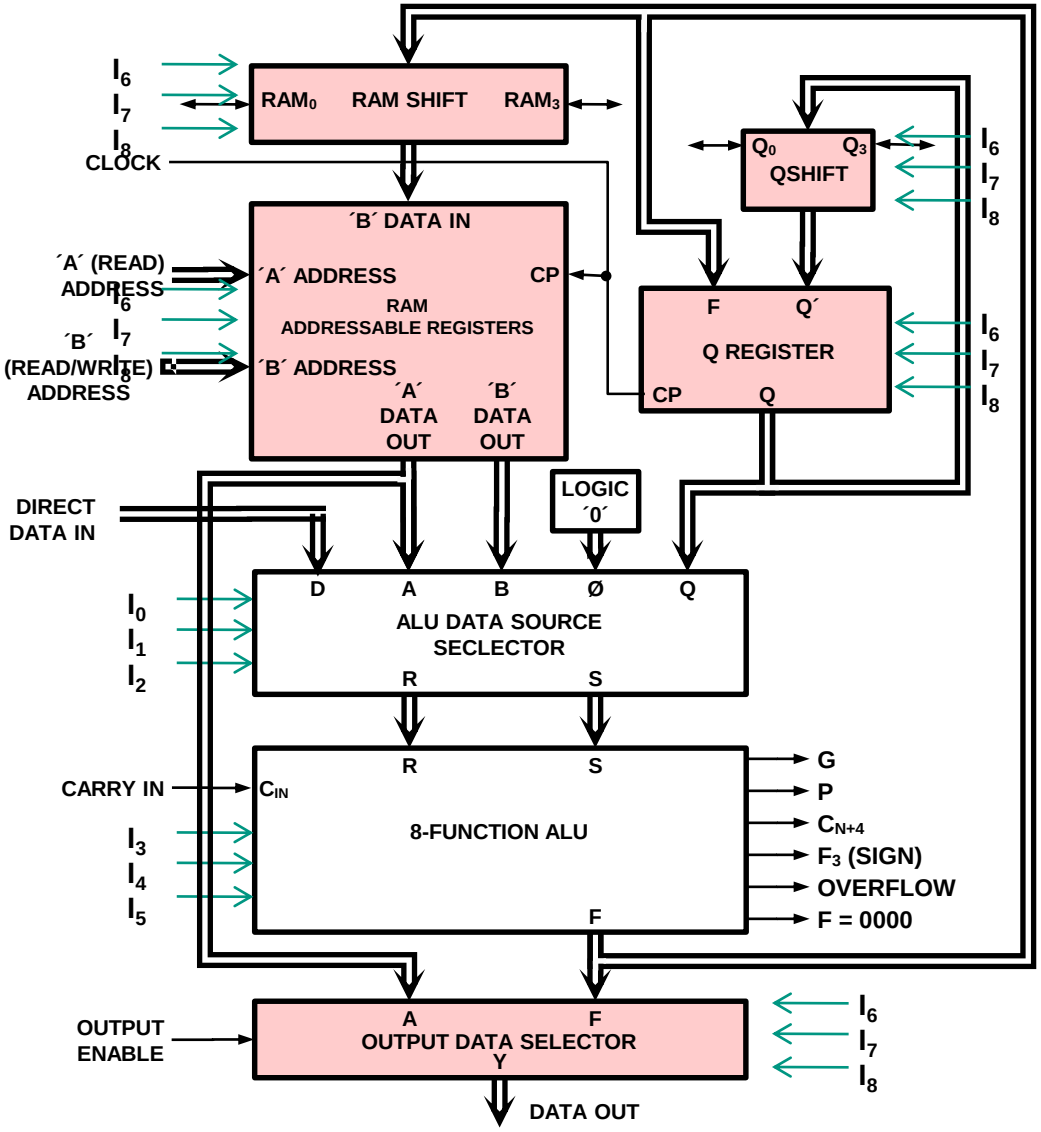
OUTPUT ENABLE

SHIFT: RAM0, RAM3, Q0, Q3

ALU Statussignale zum Steuerwerk:

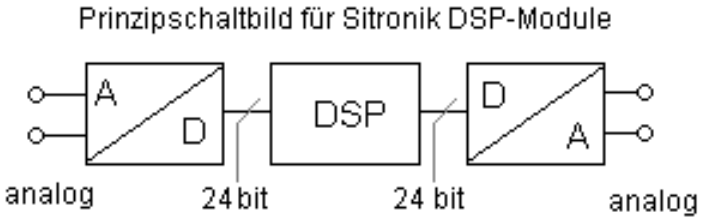
C_{N+4} F_3 (SIGN) OVERFLOW

F = 0000

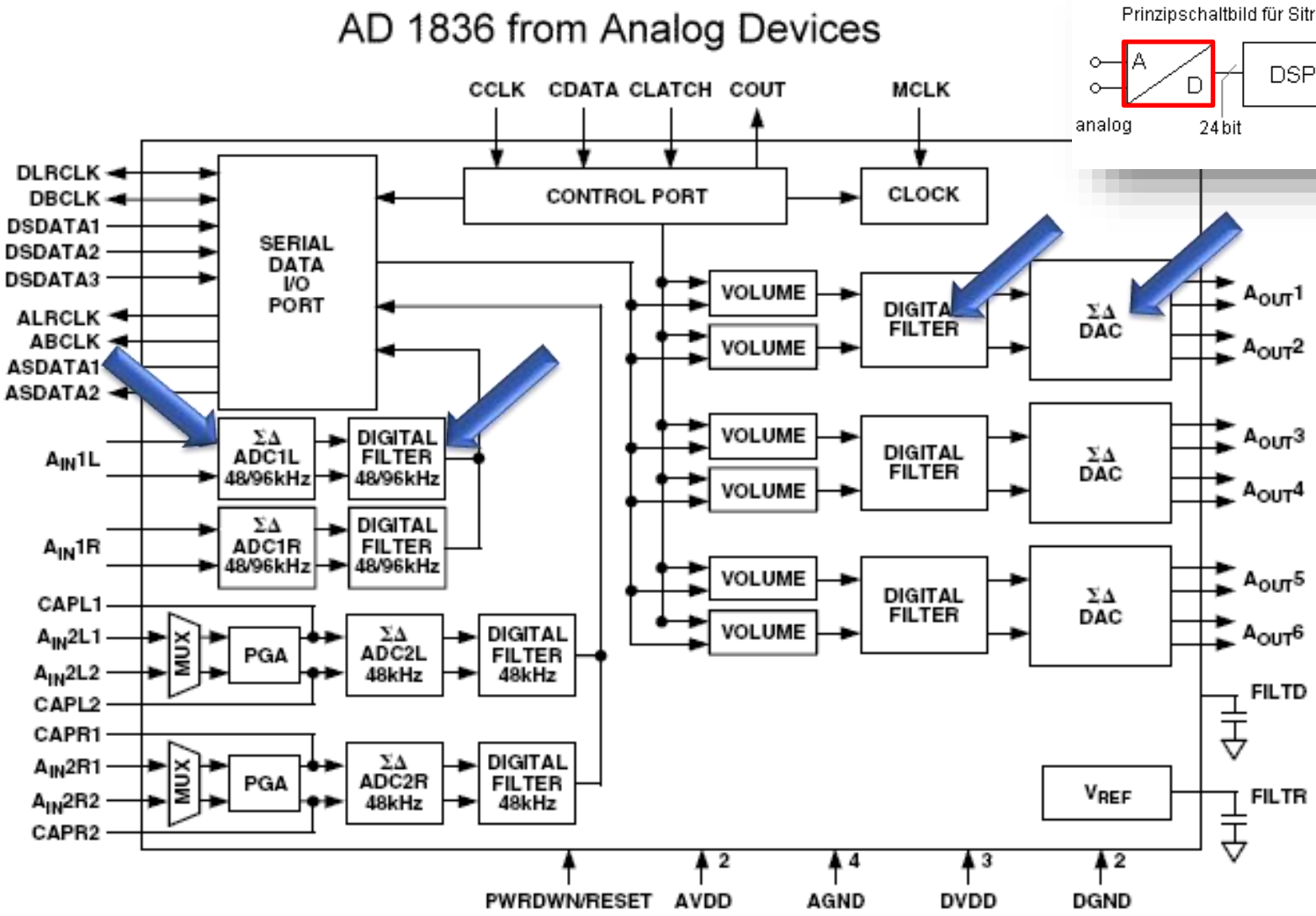


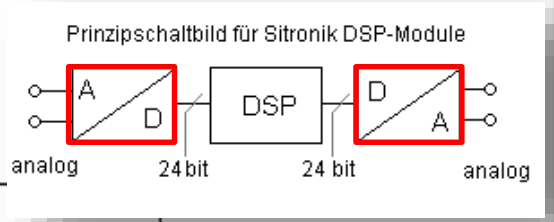
Digitale Signal Prozessoren (DSP)

- Typischerweise viele mathematische Operationen, die schnell ausgeführt werden sollen
  - Z.B. Signale für Audio/Video Anwendungen
- DSPs für Aufgaben, die analog nur schwer oder überhaupt nicht lösbar wären:
  - Frequenzfilter hoher Ordnung (z. B. Klangbeeinflussung beim Abmischen und in Mischpulten)
  - Dynamikkompression und Rauschunterdrückung mit dynamischen (adaptiven) Parametern
  - Störaustastung unter Berücksichtigung des Charakters des Signales
  - Implementierung von Effekten wie Echo, Hall oder Verfremdung von Stimmen
  - Datenkomprimierung zur digitalen Weiterverarbeitung
  - Spracherkennung und Sprachsynthese
  - Messungen in Oszilloskopen
  - Schnelle digitale Bildbearbeitung
- Entsprechende Anwendungsgebiete
  - MP3-Player
  - Amateurfunkgeräte
  - Spektrumanalysatoren
  - Mischpulte zur Klangbeeinflussung (Frequenzspektrum,Tonhöhenanpassung, Hall, etc.)
  - Moderne Empfangsgeräte des Kurzwellenrundfunks
  - Audiobeschleuniger und Soundkarten in Personalcomputern
  - Bordelektronik in Kraftfahrzeugen (speziell Motorsteuerung)
  - Frequenzweiche für Lautsprecher
  - Radar
  - Leistungselektronik (z. B. Erzeugung spezieller Signalformen zur Ansteuerung von Elektromotoren)

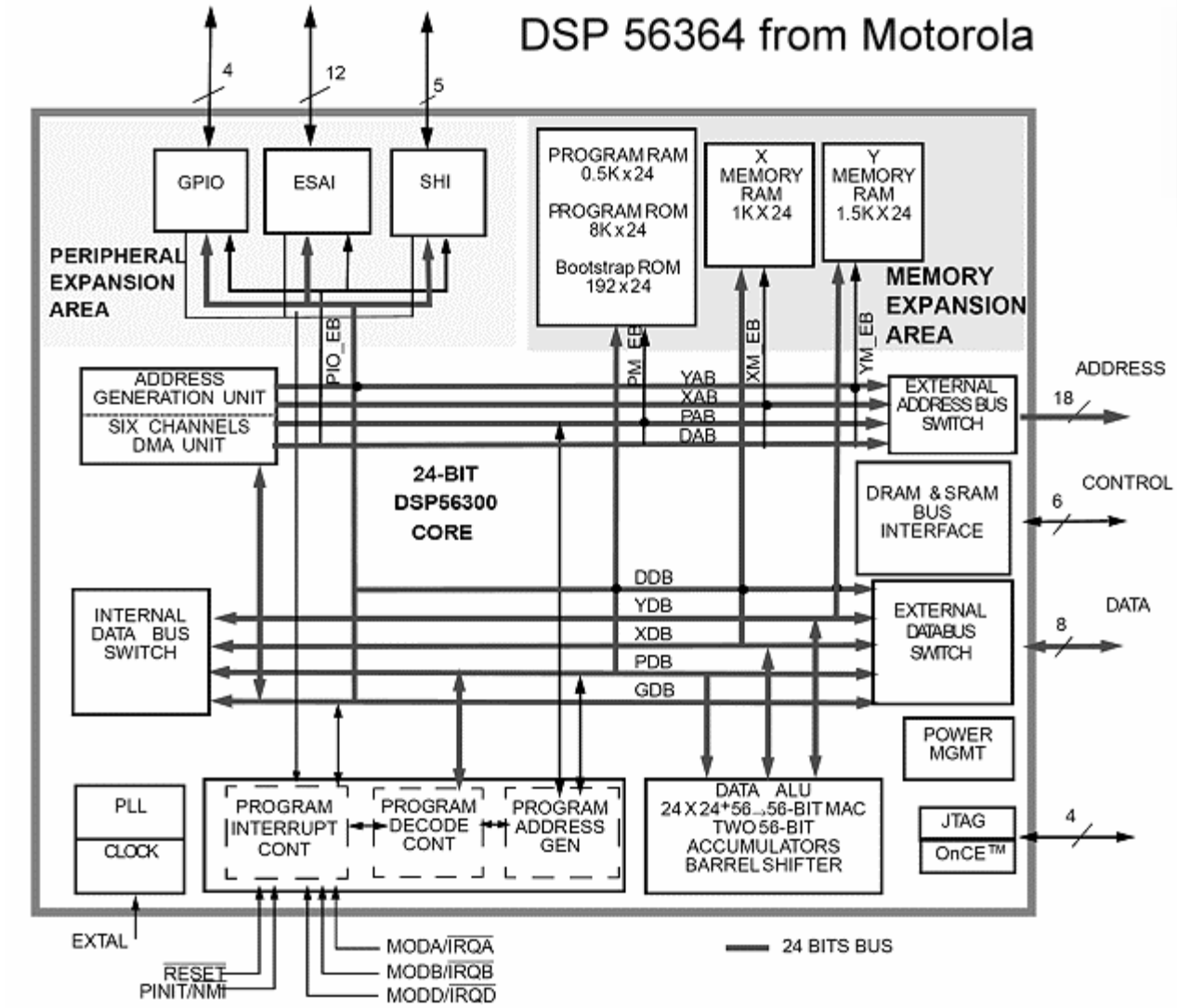


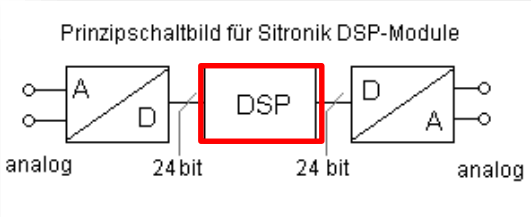
ADC und DAC im AD 1836



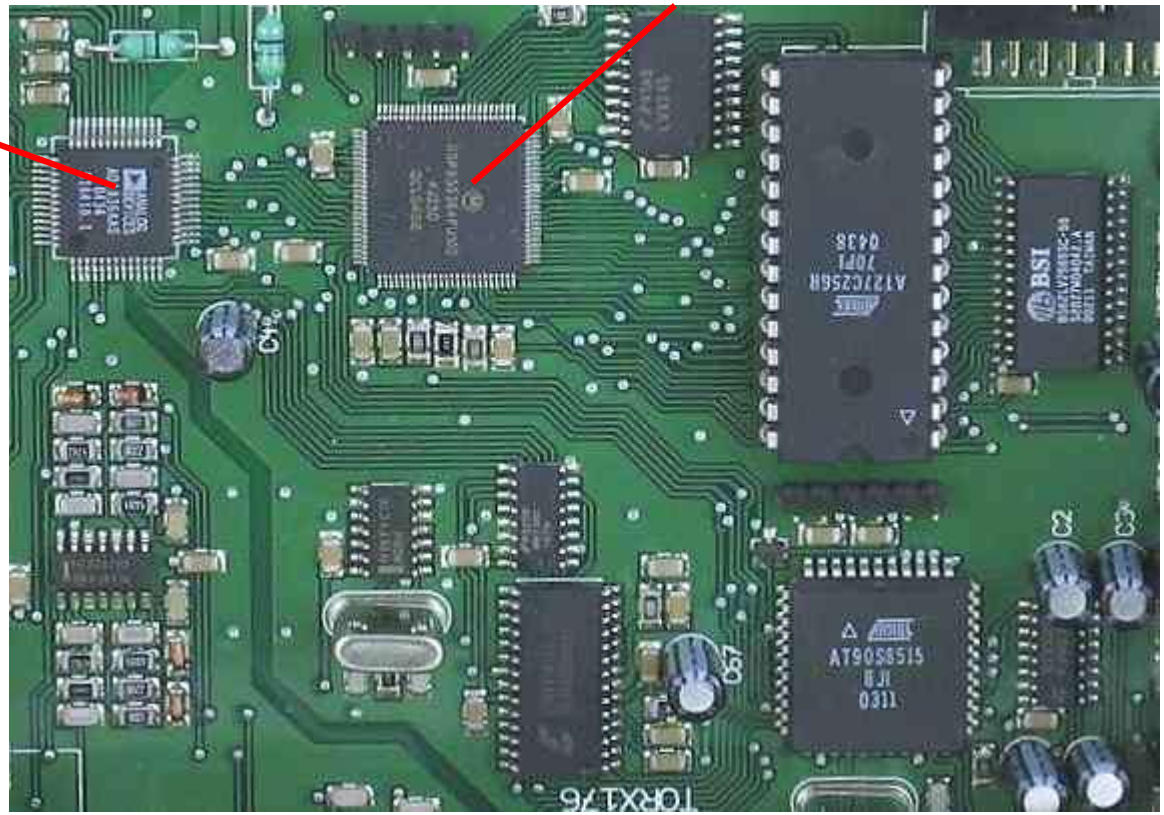


DSP Beispiel





Platine



- Die beiden Bausteine mit den Wandlern und dem DSP werden von weiteren Bausteinen umgeben, die Hilfsfunktionen erbringen:
  - Speicher, Logik, 8bit-Microcontroller mit programmierbarem Flash, Treiber, zwei Quarze und passive Bauteile.

Besonderheiten DSP

- Echtzeitfähigkeit
  - Ein DSP muss eine bestimmte Datenmenge pro Zeiteinheit sicher verarbeiten können.
  - Dies ergibt sich aus der Forderung einer meist fixen und von außen vorgegebenen Datenrate, mit der die Eingangsdaten in den DSP gelangen bzw. die verarbeiteten Daten wieder geschrieben werden müssen.
  - Eine Art Handshake oder zeitliches Anhalten bei der Datenverarbeitung ist bei dieser echtzeitfähigen Verarbeitung meistens nicht möglich.
  - Folgende Maßnahmen dienen der Erhöhung der Verarbeitungsgeschwindigkeit:
    - Spezielle synchrone, serielle Schnittstellen für die Ein- und Ausgabe der digitalen Signale.
    - Sogenannte MAC-Befehle für die gleichzeitige Multiplikation und Addition in einem Maschinenzyklus.
    - Adressgeneratoren für die Implementierung von Schleifen und Ringpufferstrukturen ohne software seitigen Overhead.
    - Implementierung des Prozessors in Harvard-Architektur.
    - Existenz eines dedizierten Hardware-Stacks.
    - Mehrmaliger Zugriff auf den Speicher in einem Zyklus.
    - Very-Long-Instruction-Word-Anweisungen.
- Konsequenz:
  - DSPs enthalten im Vergleich zu Standard-Prozessoren einen auf häufig benötigte mathematische Operationen hin geschwindigkeitsoptimierten Prozessor.
- Verarbeitung von analogen Signalen in Verbindung mit Analog-Digital-Umsetzern und Digital-Analog-Umsetzern


Fragen?

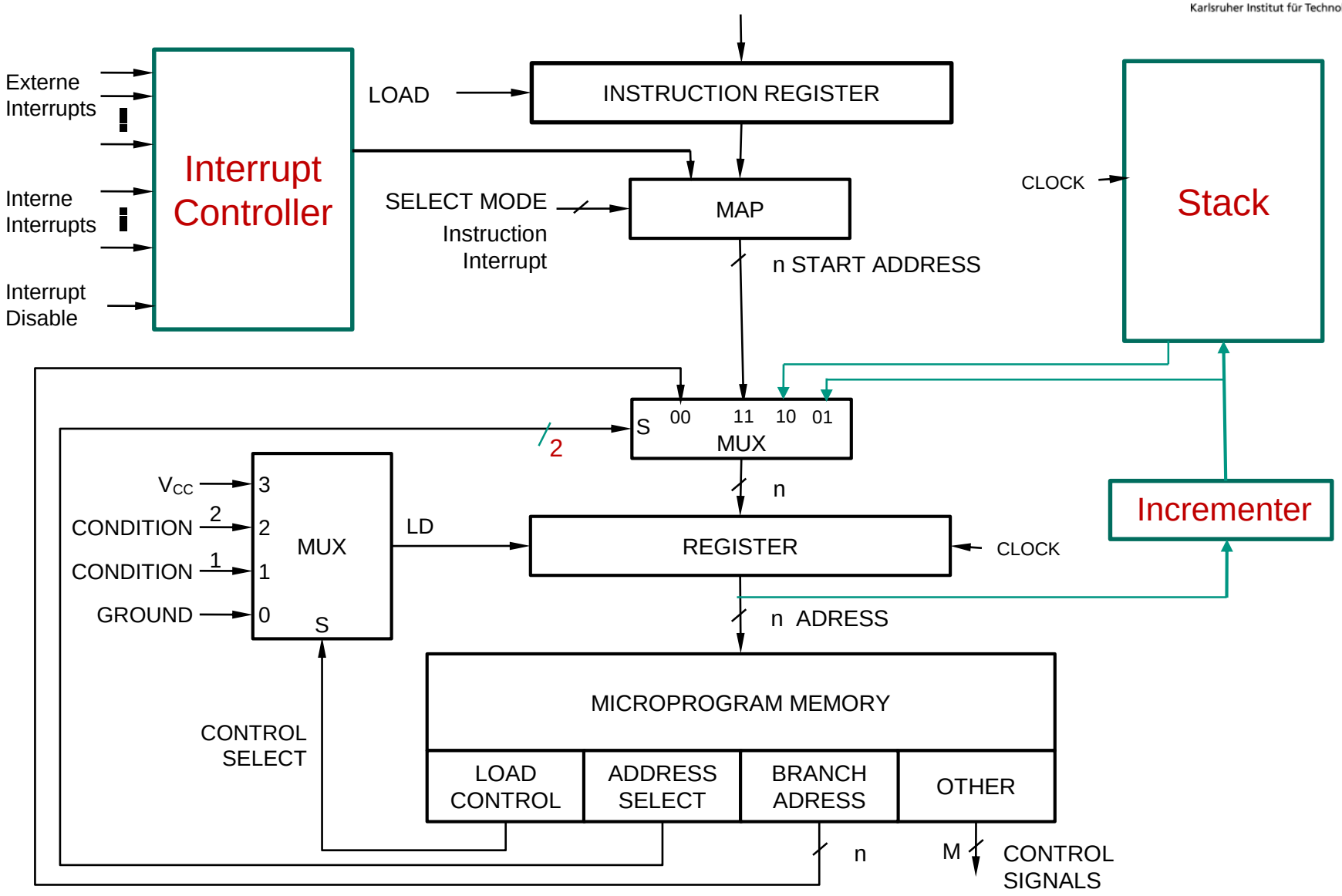

Backup
Shift

Mikrocontroller: Übersicht

- **Microchip – PIC**
  - 8-Bit, relativ alt aber stark verbreitet
- **Intel – 8051**
  - 8-Bit, relativ alt aber stark verbreitet
  - An viele Halbleiterhersteller lizenziert (z.B. Atmel)
- **Atmel – AVR8**
  - 8-Bit, neuere RISC Architektur
- **Texas Instruments – MSP430**
  - 16-Bit, ultra low power
- **ARM Limited – ARM**
  - 32-Bit, verschiedene Untertypen verfügbar: z.B. ARM7TDMI
  - An viele Halbleiterhersteller lizenziert (z.B. ST Microelectronics, NXP)
- Und viele mehr…

- Generell existieren von jeder Familie eine Vielzahl an Varianten mit unterschiedlicher Speicher- und Peripherieausstattung

Interrupt Control



1-Adress Befehlsformat (AVR8)

- opcode | Src

AVR Instruction Set

NEG – Two's Complement

Description:

Replaces the contents of register Rd with its two's complement; the value $80 is left unchanged.

Operation:

(i) Rd ← $00 - Rd

	Syntax:	Operands:	Program Counter:
(i)	NEG Rd	$0 \le d \le 31$	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0001

Remark: Negative Ganzzahlen werden im sogenannten 2er-Komplement dargestellt.
Die Regel zur Bildung des 2er-Komplements lautet:

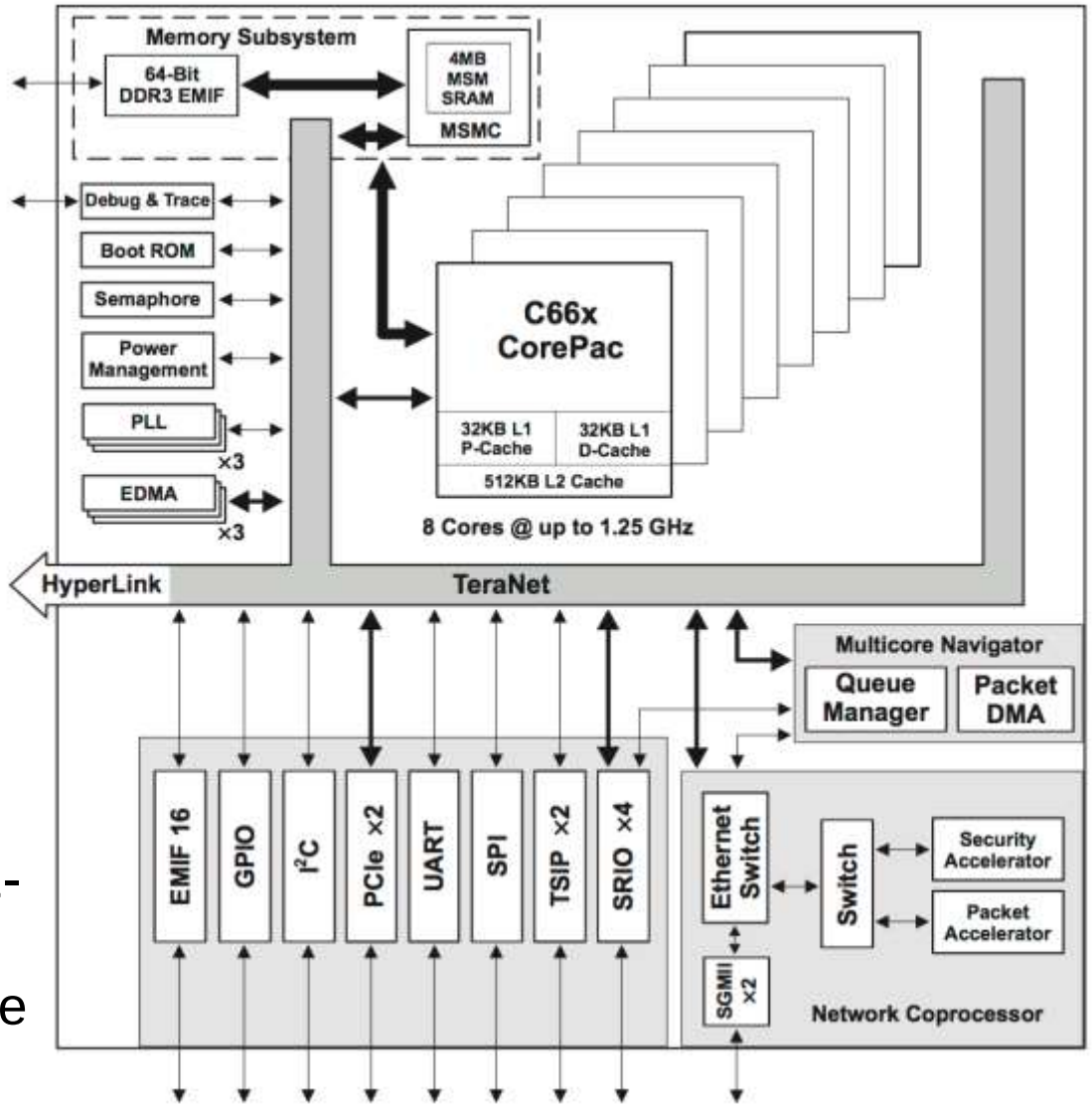
- *Alle Bits der entsprechenden positiven Zahl werden zuerst invertiert, d.h. aus einem 1 Bit wird ein 0 Bit und umgekehrt.*
- *Zu der so erhaltenen 'Zahl' wird der Wert 1 hinzuaddiert.*
- *Ein eventuell auftretender Überlauf wird verworfen*

Atmel AVR instruction set overview

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NOP
0	0	0	0	0	0	0	1	D	D	D	D	R	R	R	R	MOVW Rd,Rr Move register pair
0	0	0	0	0	0	1	0	d	d	d	d	r	r	r	r	MULS Rd,Rr
0	0	0	0	0	0	1	1	0	d	d	d	0	r	r	r	MULSU Rd,Rr
0	0	0	0	0	0	1	1	0	d	d	d	1	r	r	r	FMUL Rd,Rr
0	0	0	0	0	0	1	1	1	d	d	d	u	r	r	r	FMULS(U) Rd,Rr
0	0	0	cy	0	1	r		d	d	d	d	d	r	r	r	CPC/CP Rd,Rr
0	0	0	cy	1	0	r		d	d	d	d	d	r	r	r	SBC/SUB Rd,Rr
0	0	0	cy	1	1	r		d	d	d	d	d	r	r	r	ADC/ADD Rd,Rr ROL/LSL Rd (ADC/ADD with Rd=Rr)
0	0	0	1	0	0	r		d	d	d	d	d	r	r	r	CPSE Rd,Rr
0	0	1	0	0	0	r		d	d	d	d	d	r	r	r	TST Rd,Rr
0	0	1	0	0	1	r		d	d	d	d	d	r	r	r	EOR Rd,Rr
0	0	1	0	1	0	r		d	d	d	d	d	r	r	r	OR Rd,Rr
0	0	1	0	1	1	r		d	d	d	d	d	r	r	r	MOV Rd,Rr
0	0	1	1	K	K	K	K	d	d	d	d	K	K	K	K	CPI Rd,K
0	1	0	cy	K	K	K	K	d	d	d	d	K	K	K	K	SBCI/SUBI Rd,K
0	1	1	0	K	K	K	K	d	d	d	d	K	K	K	K	ORI Rd,K SBR Rd,K
0	1	1	1	K	K	K	K	d	d	d	d	K	K	K	K	ANDI Rd,K CBR Rd,K
1	0	k	0	k	k	s		d	d	d	d	d	y	k	k	k	LDD/STD to Z+k or Y+k
1	0	0	1	0	0	s		d	d	d	d	d	0	0	0	0	LDS rd,i/STS i,rd
16-Bit immediate SRAM-Adress i																
1	0	0	1	0	0	1		d	d	d	d	d	0	1	0	0	XCH Z,Rd
1	0	0	1	0	0	1		d	d	d	d	d	0	1	0	1	LAS Z,Rd
1	0	0	1	0	0	0		d	d	d	d	d	0	1	1	0	ELPM Rd,Z
1	0	0	1	0	0	1		d	d	d	d	d	0	1	1	0	LAC Z,Rd
1	0	0	1	0	0	0		d	d	d	d	d	0	1	1	1	ELPM Rd,Z+
1	0	0	1	0	0	1		d	d	d	d	d	0	1	1	1	LAT Z,Rd
1	0	0	1	0	0	s		d	d	d	d	d	1	1	1	1	POP/PUSH Rd
																1-operand instructions:
1	0	0	1	0	1	0		d	d	d	d	d	0	0	0	COM
														0	0	1	NEG
														0	1	0	SWAP
														0	1	1	COM
														1	0	1	ASR
														1	1	0	LSR
														1	1	1	ROR
1	0	0	1	0	1	0	0	B̄	b	b	b	1	0	0	0	SEx/CLx Status register clear/set bit
1	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0	Misc instructions:
													0	0	0	RET
													0	0	1	RETI
													1	0	0	SLEEP
													1	0	1	BREAK
													1	1	0	WDR
													1	1	0	LPM
													1	1	0	ELPM
													1	1	1	SPM
													1	1	1	SPM Z+
1	0	0	1	0	1	0	c	0	0	0		1	0	0	1	Indirect jump/call to Z or EIND:Z
1	0	0	1	0	1	0		d	d	d	d	d	1	0	1	0	DEC Rd
1	0	0	1	0	1	0	0	k	k	k	k	1	0	1	1	DES round k
1	0	0	1	0	1	0		k	k	k	k	k	1	1	c	k	JMP/CALL abs22
1	0	0	1	0	1	1	0	k	k	p	p	k	k	k	k	ADIW Rp,uimm6
1	0	0	1	0	1	1	1	k	k	p	p	k	k	k	k	SBIW Rp,uimm6
1	0	0	1	1	0	B	0	a	a	a	a	a	b	b	b	CBI/SBI a,b (IO-Operation)
1	0	0	1	1	0	B	1	a	a	a	a	a	b	b	b	SBIC/SBIS a,b (IO-Skip-Nextstep)
1	0	0	1	1	1	r		d	d	d	d	d	r	r	r	r	MUL, unsigned: R1:R0 = Rr×Rd
1	0	k	0	k	k	s		d	d	d	d	d	y	k	k	k	See `10k0` above
1	0	1	1	s	a	a		d	d	d	d	d	a	a	a	a	OUT/IN to I/O space
1	1	0	c	12 bit signed offset													Relative jump/call to PC ± 2×simm12
1	1	1	0	K	K	K	K	d	d	d	d	K	K	K	K	LDI Rd,K
1	1	1	1	0	B̄	7-bit signed offset								b	b	b	Conditional branch on status register bit
1	1	1	1	1	0	s		d	d	d	d	d	0	b	b	b	BLD/BST register bit to STATUS.T
1	1	1	1	1	1	B		d	d	d	d	d	0	b	b	b	SBRC/SBRS skip if register bit equals B

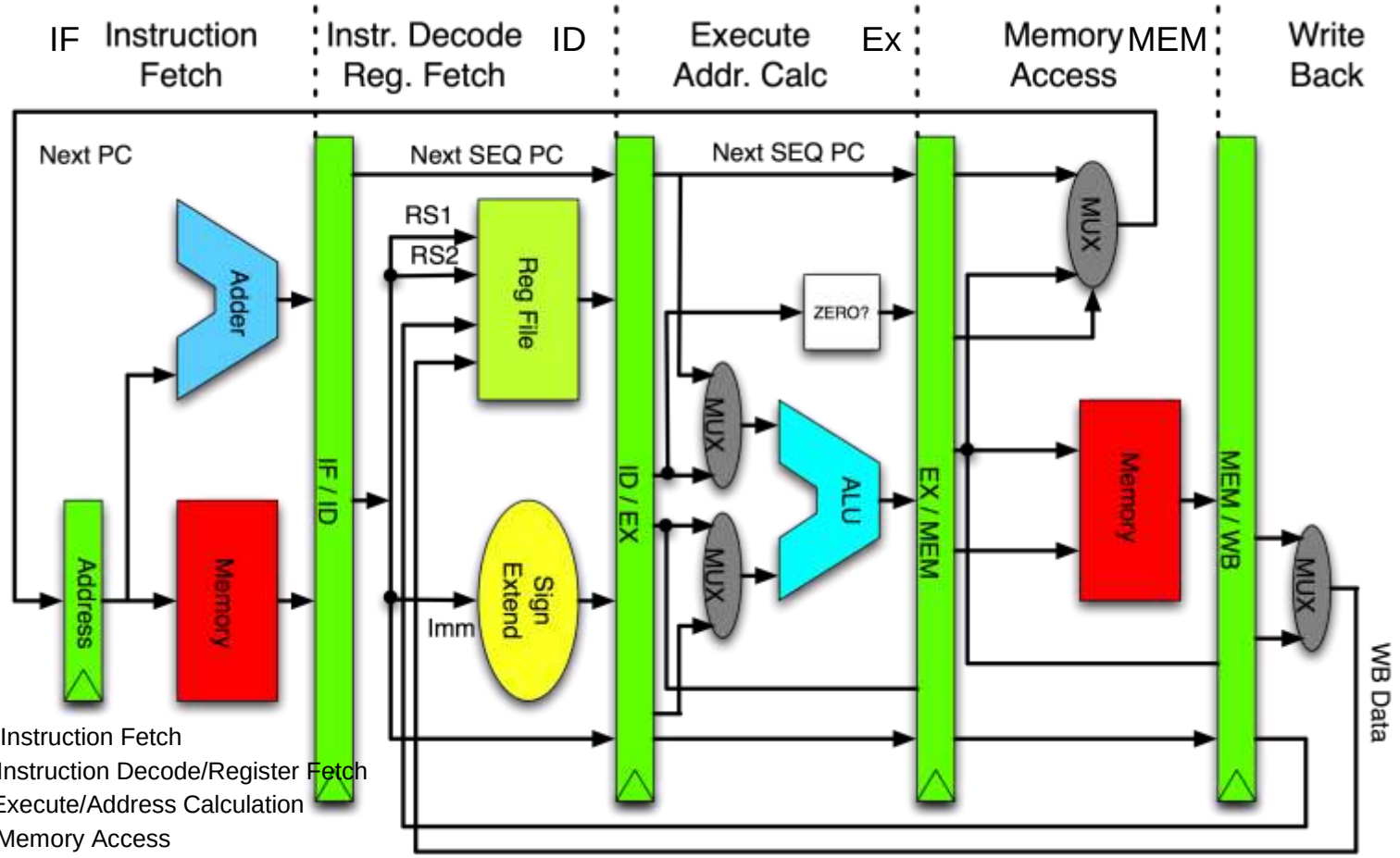
DSP Beispiel: TI TMS320C6678 (I)

- High-End 8 Core-DSP von Texas Instruments
- Fest- und Gleitkomma-arithmetik
- 1 GHz Taktfrequenz
- 4 MB Shared Memory
- Bis zu 8 GB DDR3 adressierbar
- Umfangreiche Peripherie
- 2 Datenpfade
- Je 32 Register mit 32 bit
- 8-fach VLIW-Architektur
- Harvard-Architektur der L1-Speicher
- L1P, L1D und L2 als Cache konfigurierbar



RISC processor 5 stage pipeline

Developed by MIPS Technologies (formerly MIPS Computer Systems, Inc.).



IF -- Instruction Fetch
ID -- Instruction Decode/Register Fetch
EX -- Execute/Address Calculation
MEM -- Memory Access
WB -- Write Back

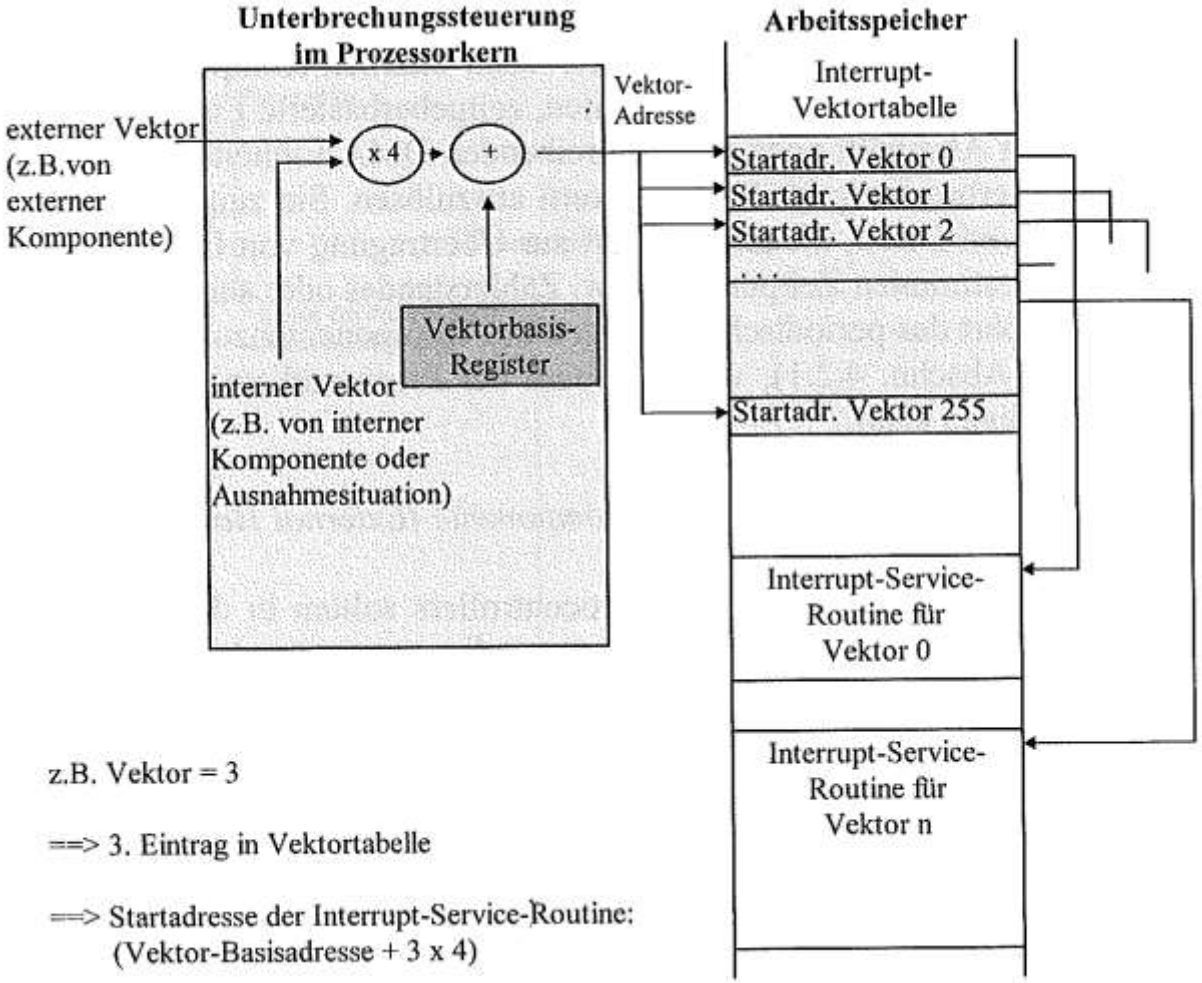
The stage-by-stage architecture of a MIPS microprocessor with a pipeline. Although the memory is shown twice for clarity of the pipeline, MIPS architectures have only one memory bank (i.e. von Neumann architecture).

Basic Pipeline Steps

- Instruction fetch (IF):
  - The instruction pointed to by the PC is fetched from memory into the instruction register of the CPU, and the PC is incremented to point to the next instruction in the memory.
- Instruction decode/register fetch (ID):
  - The instruction is decoded, and in the second half of the stage the operands are transferred from the register file into the ALU input registers (here meaning: latches).
- Execution/effective address calculation (EX):
  - The ALU operates on the operands from ALU input registers and eventually puts the result into ALU output register. The contents of this register depend on the type of instruction. If the instruction is:
    - Register-register (e.g. arithmetic/logical): the ALU outputs the result of the operation into the ALU output register;
    - Memory reference (e.g. load/store), the ALU output register contains an effective memory address;
    - Control transfer (e.g. branch on equal), then the ALU produces the jump / branch target address (which is stored in the ALU output register) and, at the same time, the branch direction.
- Memory access/branch completion (MEM):
  - Only for load, store, and branch instructions. If the instruction is:
    - register-register: the content of the ALU output register is transferred to the ALU result register.
    - load: the data is read from memory (as pointed to by the ALU output register) and is placed in the load memory data register;
    - store: the data in the store value register is written into the D-cache (as pointed to by the ALU output register);
    - control transfer: for jump and branch that is taken: the PC is replaced by the ALU output register content; otherwise, the PC remains unchanged (in both cases, the next step WB is skipped);
- Write back (WB):
  - The result of the instruction execution (register-register or load instruction) is stored into the register file in the first half of the phase.
    In particular, the load memory data register or the ALU result register is written into the register file.

Ermittlung Startadresse ISR durch Interrupt Controller

Eine Unterbrechungs-routine (interrupt service routine, ISR) ist ein Programmstück, das von einem Hauptprozessor (CPU) ausgeführt wird, wenn er durch eine Unterbrechungs-anforderung (engl. Interrupt request, IRQ) gezwungen wird, den gegenwärtigen Programmablauf zu unterbrechen und einen Interrupt auszuführen.



Aus Brinkschulte, Ungerer
„Mikrocontroller, Mikroprozessoren“
Springer-Lehrbuch