Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund

Programmierung verteilter eingebetteter Systeme

Teamprojekt –Einführung und Grundlagen Teil 2

Stephan Rottmann, Ulf Kulau, Felix Büsching

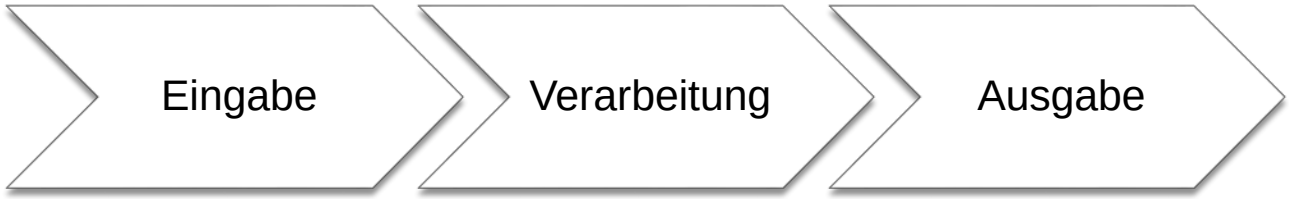
Summer Term 2014

Ablauf

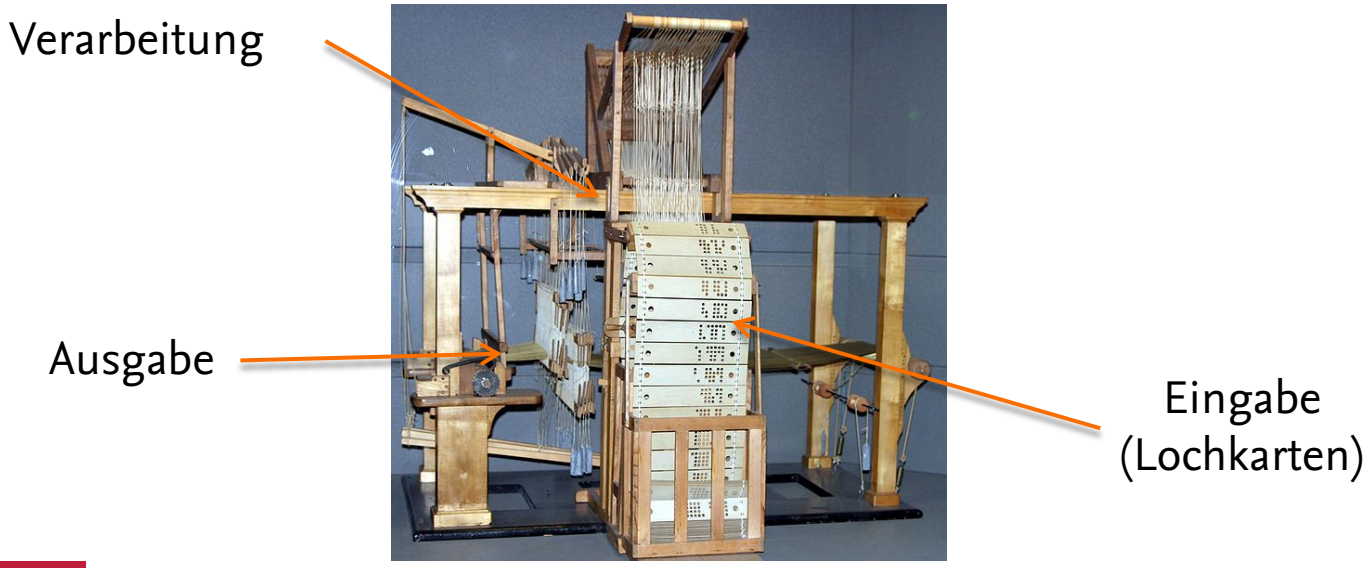
- Organisatorisches
- Grundlagen 1
- Praktikumshardware
- Aufgabe 1
- Grundlagen 2
- Aufgabe 2

Mikroprozessor - Prinzip

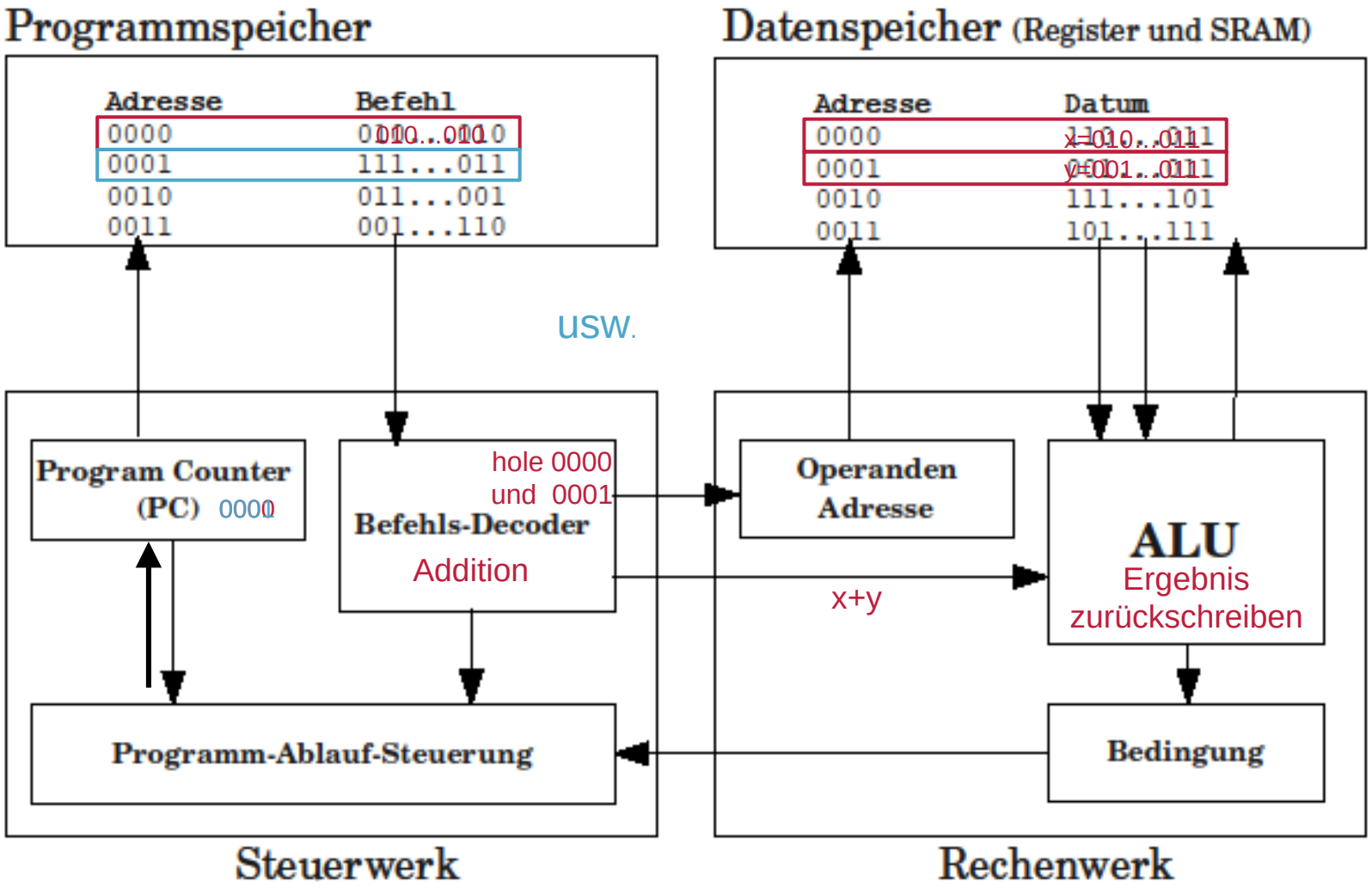
EVA Prinzip:



Beispiel: Mechanischer Webstuhl von 1805



Mikroprozessor Prinzip: Steuer und Rechenwerk



Kurfassung: Ausführung eines binär codierten Befehls

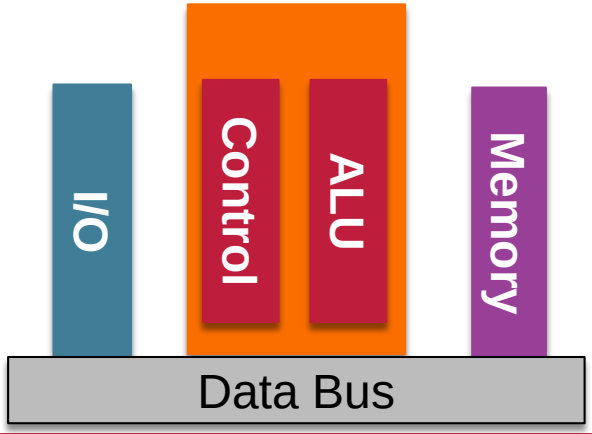
- Befehlsadresse aus dem PC an den Programmspeicher senden
- Befehl an dieser Adresse im Programmspeicher auslesen und dekodieren
- Operationscode (was soll gemacht werden) an die ALU übergeben
- Operanden aus dem Datenspeicher holen und Operation ausführen
- Ergebnis an den Datenspeicher übergeben und
- den nächsten Befehl vorbereiten

C -> Assembler -> Machinecode

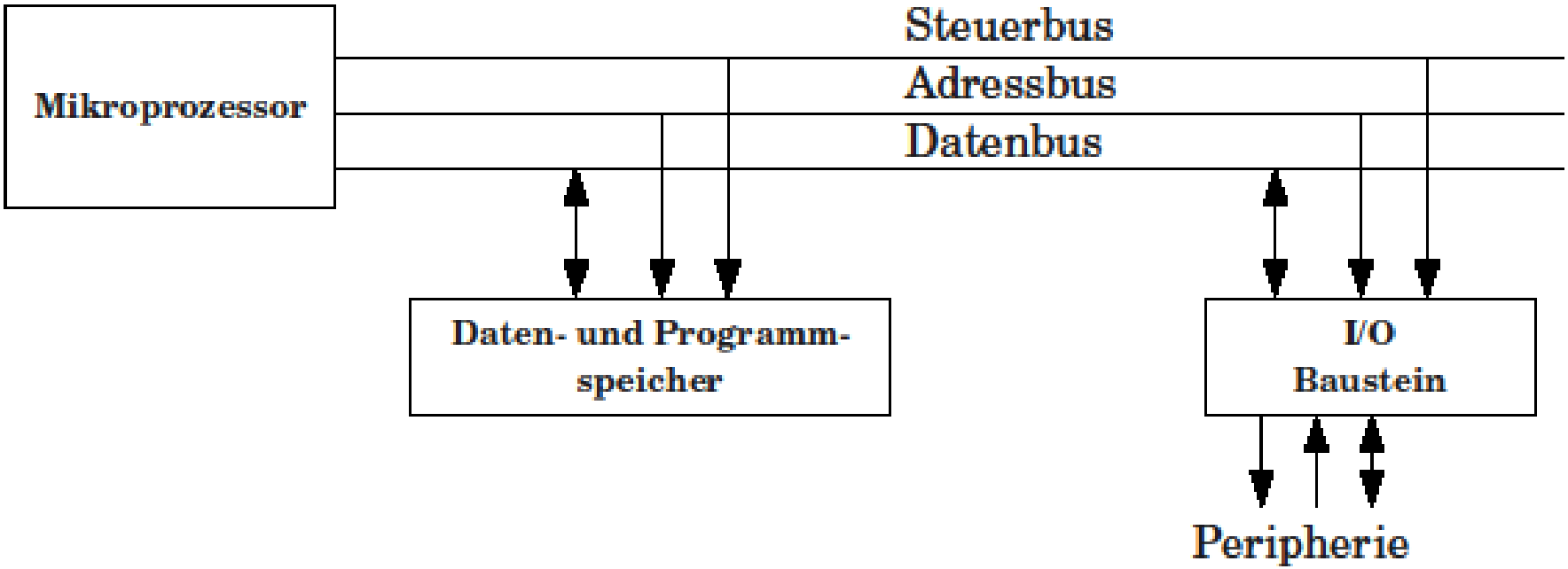
Architekturen: Von Neumann vs. Harvard

Von Neumann Architektur

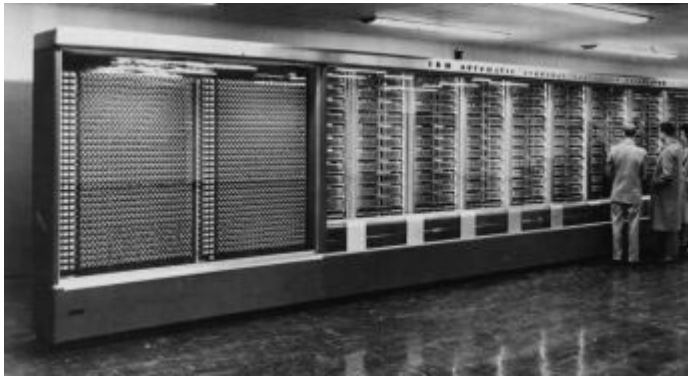
- 1945 entwickelt von John von Neumann
- Einfache Architektur
- Streng sequentiell
- Selber Speicher für Daten und Instruktionen
- Gemeinsamer Bus zwischen Instruktionsspeicher, Datenspeicher und ALU
  - CPU kann entweder Instruktionen lesen oder den Datenspeicher lesen/schreiben
  - “Von Neumann Bottleneck”
  - No race-conditions



Von-Neumann-Architektur

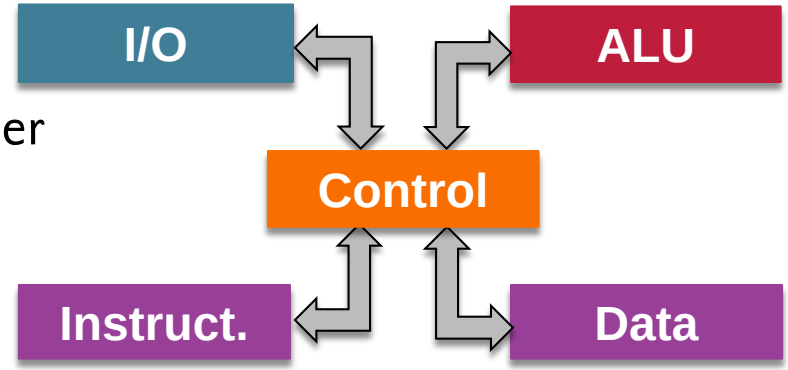


Architekturen: Von Neumann vs. Harvard

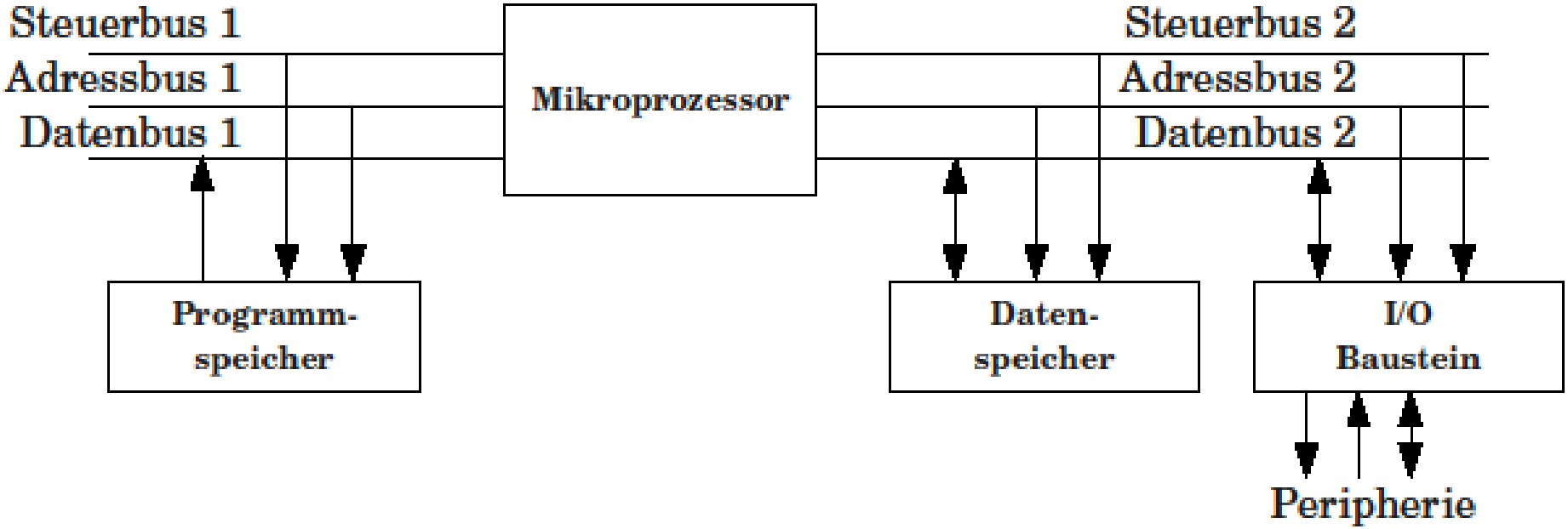


Harvard Architektur

- Benannt nach dem Harvard Mark I (1944)
  - IBM Automatic Sequence Controlled Calculator
- Daten- und Instruktionsspeicher sind physikalisch getrennt
  - Lesen von Instruktionen und Zugriff auf Datenspeicher kann gleichzeitig erfolgen (erlaubt Pipelining)
- Modifikationen:
  - Direkte Pfade zwischen Instruktionsspeicher und ALU (z.b. für Konstanten)



Harvard-Architektur



Die Sprache der Mikroprozessoren

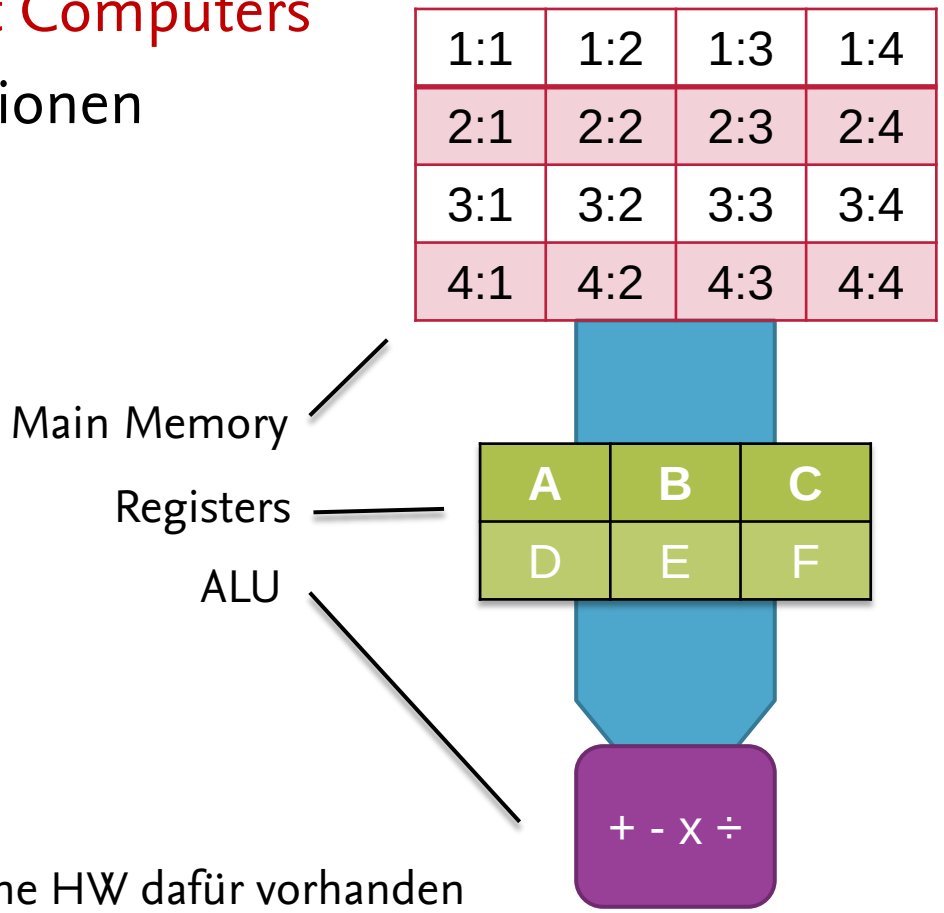
- Prozessor implementiert in HW Funktionalitäten
  - Z.B. arithmetische, logische, Sprung, Ein/Ausgabe
- Befehlssatz:
  - Menge an Maschinenbefehlen, um diese Funktionalitäten zu nutzen
  - Allgemeines Format von Befehlen:

```
<Opcode> {<Operand>, {<Operand>}}
```

Befehlssätze: CISC

CISC = Complex Instruction Set Computers

- Ermöglicht komplexe Instruktionen
  - Mikroprogrammierung
- Geringe Codegröße
  - Dank Microcode
- Potentiell mehrere Takte pro Instruktion nötig



Multiplikation

- MULT 1:3, 2:3
  - MULT Befehl möglich, auch wenn keine HW dafür vorhanden

Fallstricke CISC - Codebeispiel

Shift-Operation:

- Als Micro Instruction ist auf dem CISC lediglich vorhanden

```
x << 1;
```

- Ein `x << n` ist somit als komplexe Instruktion implementiert

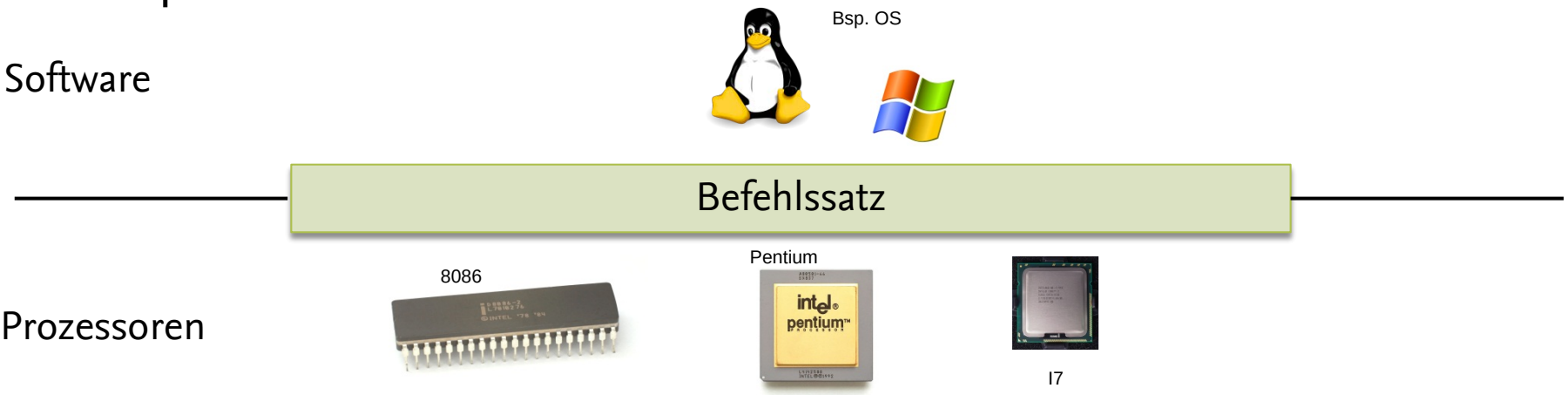
```
for (i = 0; i < n; i++){
        x << 1;
}
```

- Führt zu unterschiedlichen Ausführungszeiten!
  - Beispiel. SW-UART

Motivation CISC

Flexibilität:

- Erlaubt Befehle, welche nicht in HW realisiert sind (z.B. MULT)
- Befehlssatz kann auf Softwareebene erweitert/korrigiert werden
- Emulation von Befehlssätzen zum erhalt der Kompatibilität
- Beispiel: x86 Befehlssatz



Völlig unterschiedliche Technologien, gleicher Befehlssatz

Befehlssätze: RISC

RISC = Reduced Instruction Set Computer

- Ein Takt pro Instruktion
- Register to register
  - LOAD and STORE independent instructions
- Kleiner Befehlssatz
- Emphasis on Software
- RISC verarbeitet pro Taktzyklus mehr Instruktionen als CISC
- Performance

$$\frac{time}{program} = \frac{time}{cycle} * \frac{cycles}{instruction} * \frac{instructions}{program}$$

Multiplikation

- LOAD A, 1:3
- LOAD B, 2:3
- PROD A, B
- STORE 3:3

Ansatz: Braucht man komplexe Befehle?

- Studie von IBM 1975
  - 10 Befehle machen etwa 2/3 des gesamten Codes aus!
- Weniger Befehle, dafür Ausführung in einem Takt
- Keine aufwendige Dekodiereinheit für Befehle
  - Geringere Leistungsaufnahme
- Ideal für den Einsatz in energieeffizienten eingebetteten System




Microprocessors and Architectures: RISC vs. CISC

Performance

$$\frac{time}{program} = \frac{time}{cycle} * \frac{cycles}{instruction} * \frac{instructions}{program}$$

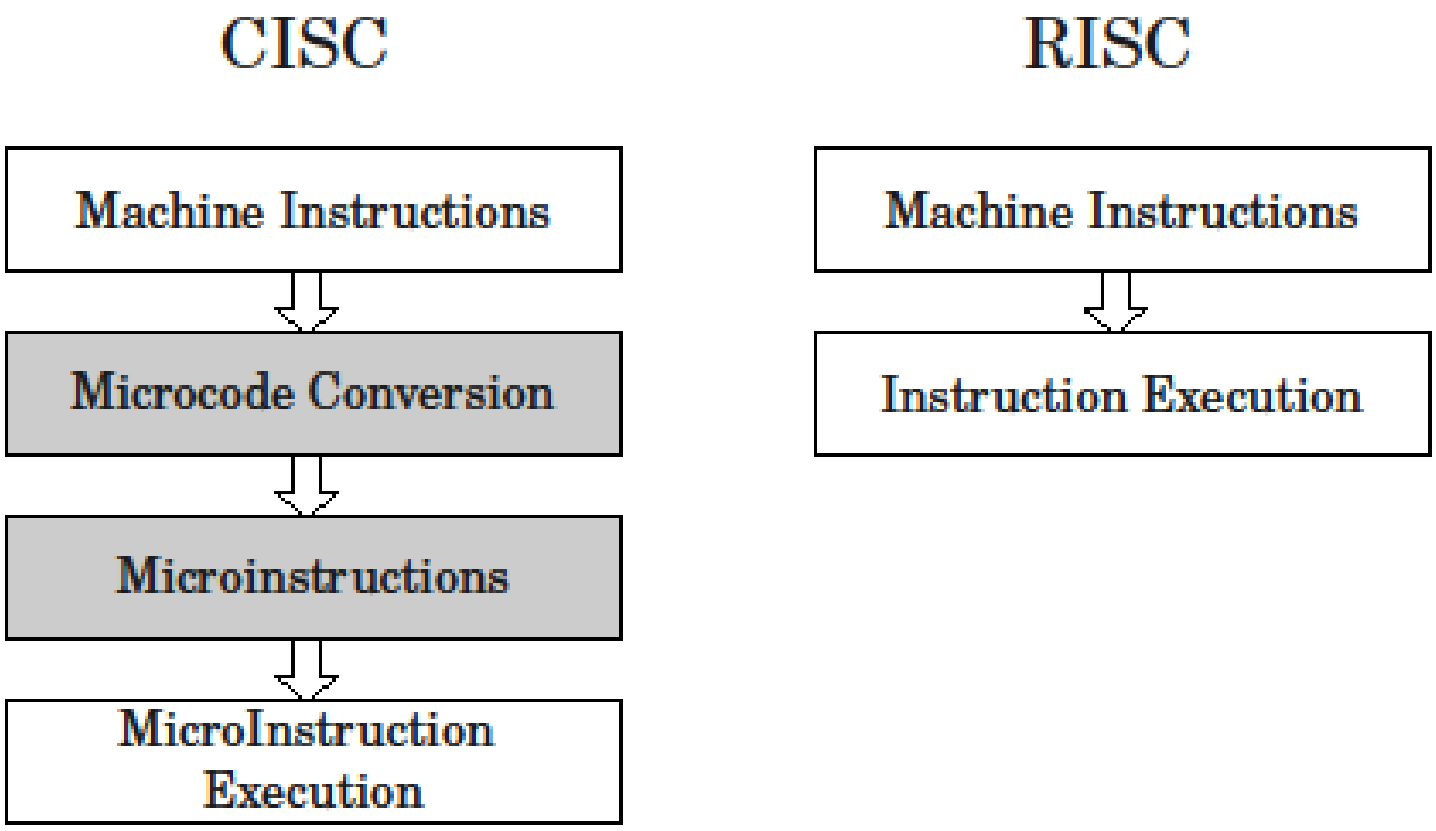
↓RISC ↓CISC

- CISC minimizes instructions / program
  - Less memory needed
  - More cycles per instructions needed
- RISC minimizes cycles / instruction
  - More memory needed
  - Less cycles per instruction needed

Memory is cheap – nowadays even in microcontrollers!

High clock rates → more energy needed

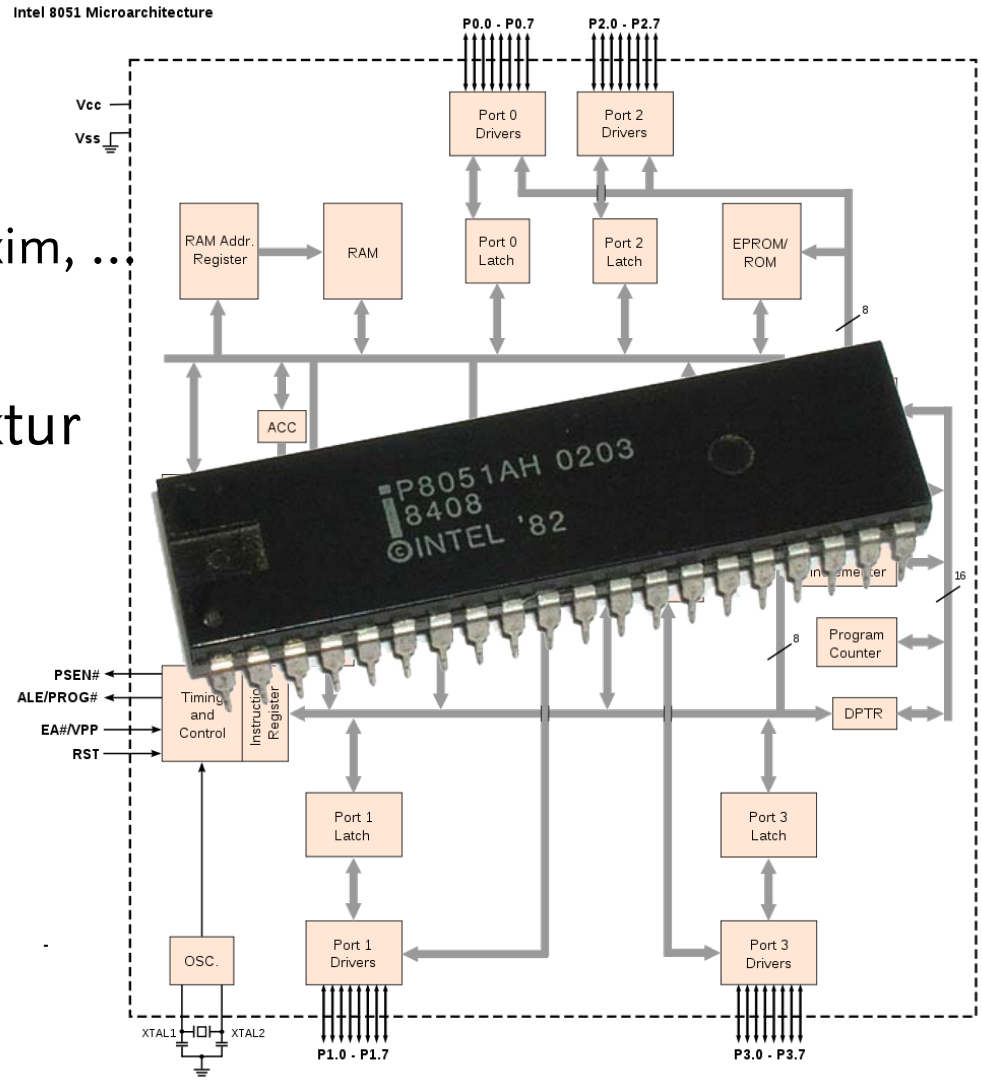
CISC vs. RISC: Zusammenfassung



Intel 8051 – “der” Mikro-Controller

MSC-51 (8051, 8031, 8751)

- 1980 eingeführt von Intel
  - Auch von AD, Atmel, Infineon, Maxim, …
- 8-bit, CISC
- (modifizierte) Harvard Architektur
  - 4 Register-Bänke
- ≥ 128 Byte internerRAM
- Externer ROM, RAM
- 16 bit Adressbus → ≤64 Kbyte
- Diverse Peripherie
  - i/o-Ports, UART, …



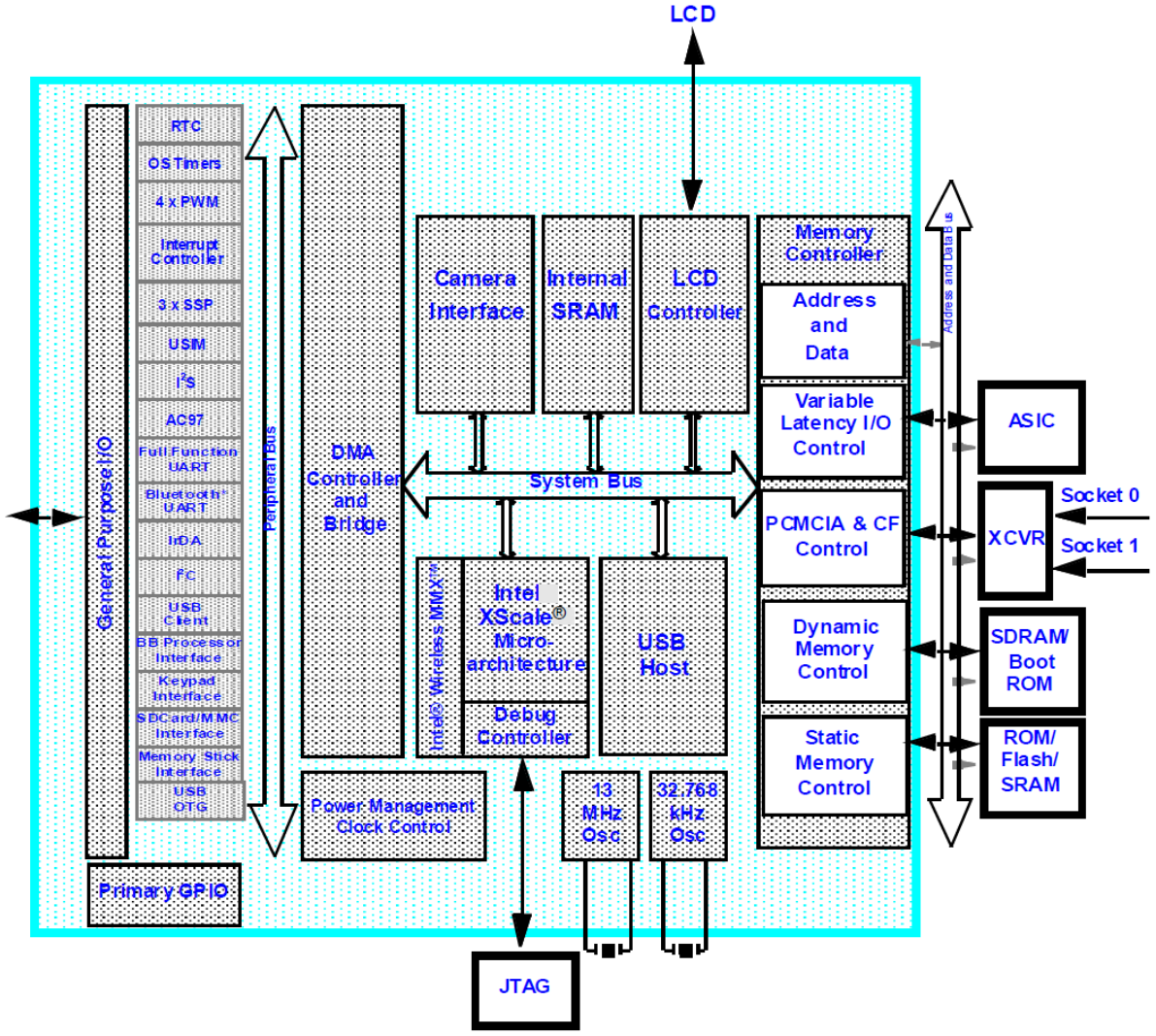
Advanced RISC Machine (ARM)

- Vorgestellt 1987
- Im Embedded Bereich meist genutzte Architektur
  - Hohe Enrgieeffizienz
- 32 bit, RISC
  - XNU-Kernel (Mac OS X/iOS), Linux-Kernel, Windows NT Kernel

XScale PXA271 (ARMv5 architecture)

- Released 2004
- 13 - 416 MHz
- 32 MB Flash, 32 MB RAM
- USB (Client & Host), AC'97 Audio
- 13 MHz Active Power = 44.2 mW

XScale PXA271 (ARMv5 Architektur)



Atmel AVR Series – ATMega 128A

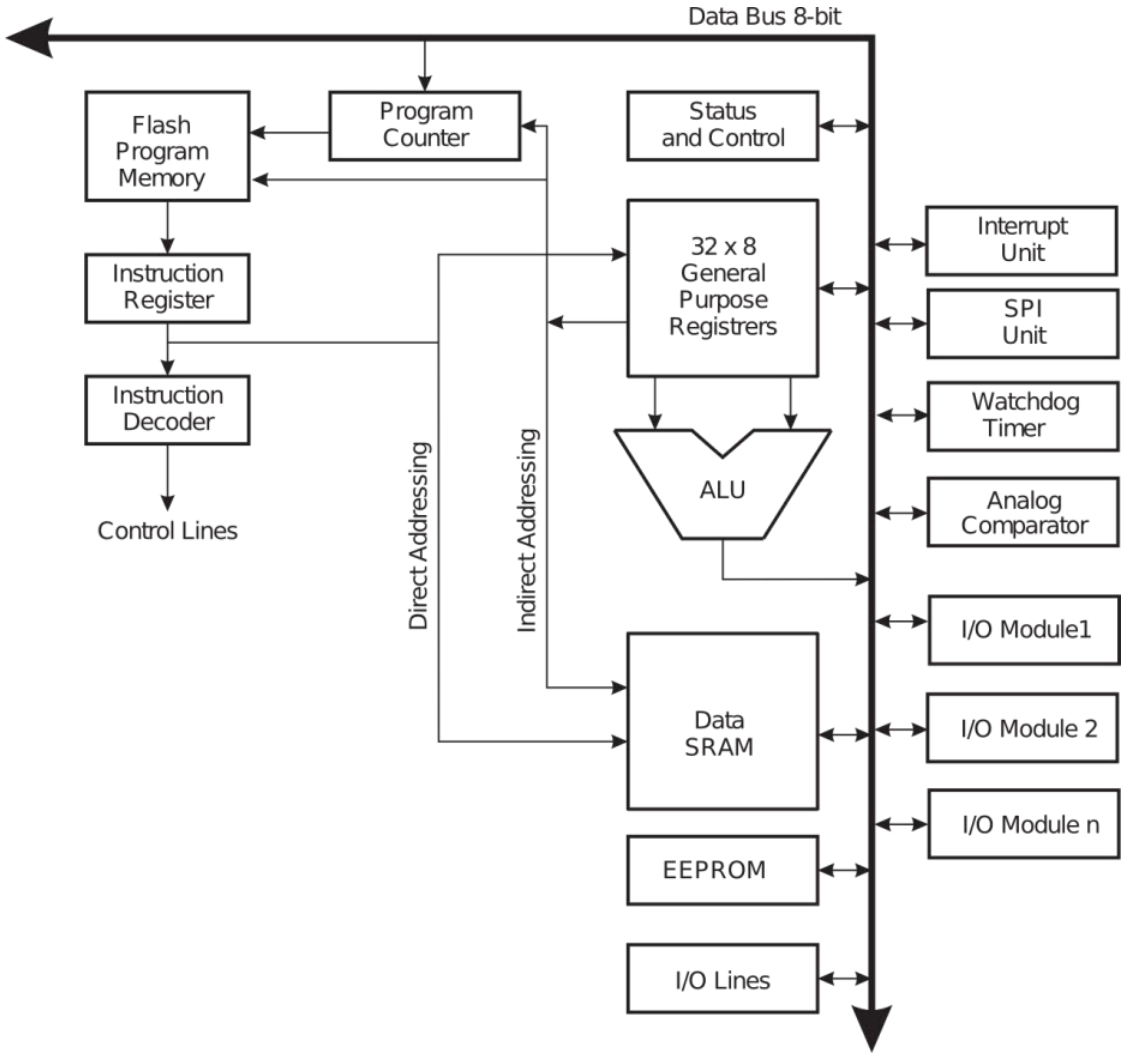
AVR Series

- Entwickelt 1996
- 8-bit, RISC
- (modified) Harvard Architektur

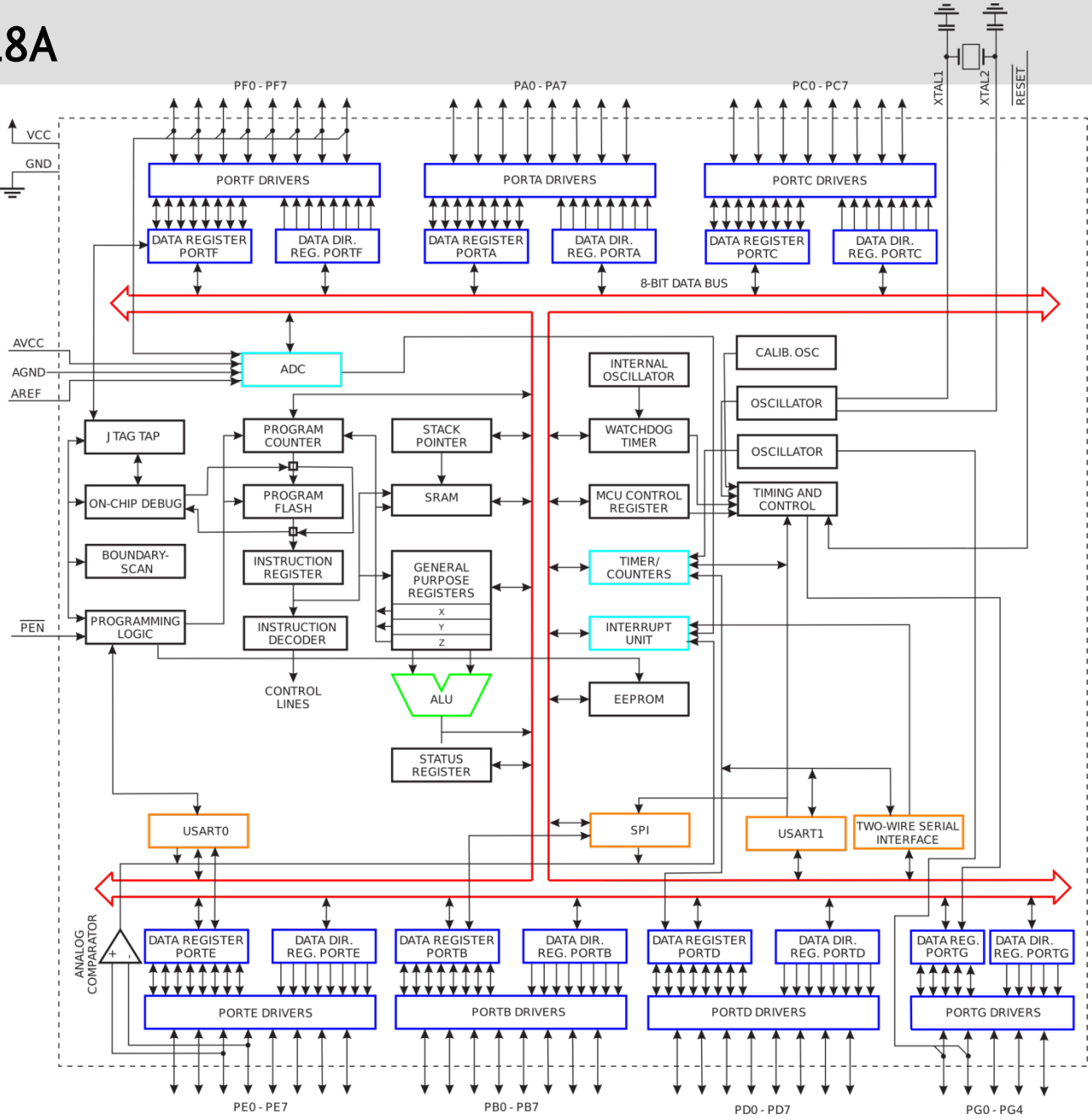
ATMega 128A (Praktikumshardware)

- 128 kB Flash, 4 kB SRAM, 4 kB EEPROM
- 2 USARTs, separated buses for SPI, I^2C
- 8 Kanal10-bit ADC (Multiplex)

Architektur der Atmel AVR Serie



Atmega 128A

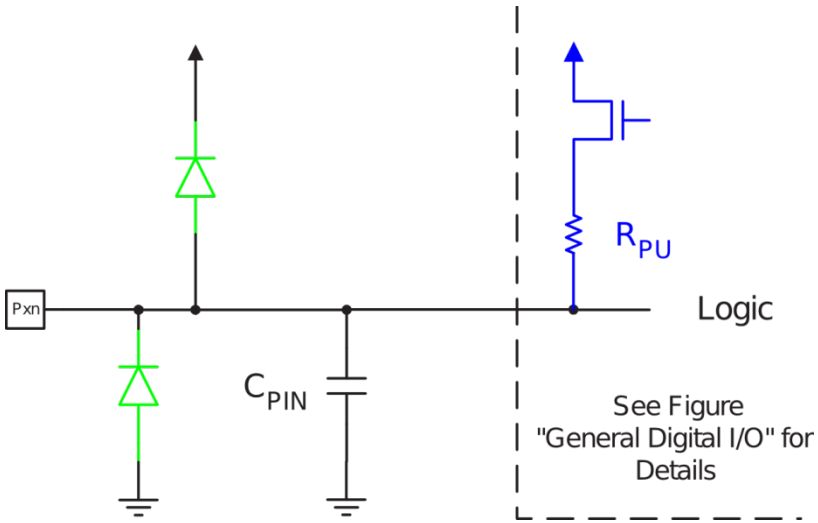

PF0 - PF7
PA0 - PA7
PC0 - PC7
XTAL1
XTAL2
RESET
VCC
GND
PORTF DRIVERS
PORTA DRIVERS
PORTC DRIVERS
DATA REGISTER PORTF
DATA DIR. REG. PORTF
DATA REGISTER PORTA
DATA DIR. REG. PORTA
DATA REGISTER PORTC
DATA DIR. REG. PORTC
8-BIT DATA BUS
AVCC
AGND
AREF
ADC
INTERNAL OSCILLATOR
CALIB. OSC
OSCILLATOR
J TAG TAP
PROGRAM COUNTER
STACK POINTER
WATCHDOG TIMER
OSCILLATOR
ON-CHIP DEBUG
PROGRAM FLASH
SRAM
MCU CONTROL REGISTER
TIMING AND CONTROL
BOUNDARY-SCAN
INSTRUCTION REGISTER
GENERAL PURPOSE REGISTERS
X
Y
Z
TIMER/ COUNTERS
PEN
PROGRAMMING LOGIC
INSTRUCTION DECODER
INTERRUPT UNIT
CONTROL LINES
ALU
EEPROM
STATUS REGISTER
USART0
SPI
USART1
TWO-WIRE SERIAL INTERFACE
ANALOG COMPARATOR
DATA REGISTER PORTE
DATA DIR. REG. PORTE
DATA REGISTER PORTB
DATA DIR. REG. PORTB
DATA REGISTER PORTD
DATA DIR. REG. PORTD
DATA REG. PORTG
DATA DIR. REG. PORTG
PORTE DRIVERS
PORTB DRIVERS
PORTD DRIVERS
PORTG DRIVERS
PE0 - PE7
PB0 - PB7
PD0 - PD7
PG0 - PG4

Peripherie des Controllers

- Digitale Ein- und Ausgänge
- Schnittstellen
  - U(S)ART
  - I²C
  - SPI
- Analoge Eingänge
  - Analog-Digital Wandler
  - Komparator
- PWM

I/O-Ports

- I/O-Pins können als Ein- oder Ausgang verwendet werden
- Richtung wird mit *Data Direction Register* bestimmt
- Rudimentäre Schutzbeschaltung vorhanden
- Verwendung zum Beispiel für
  - LEDs
  - Taster
  - Steuerleitungen
  - …

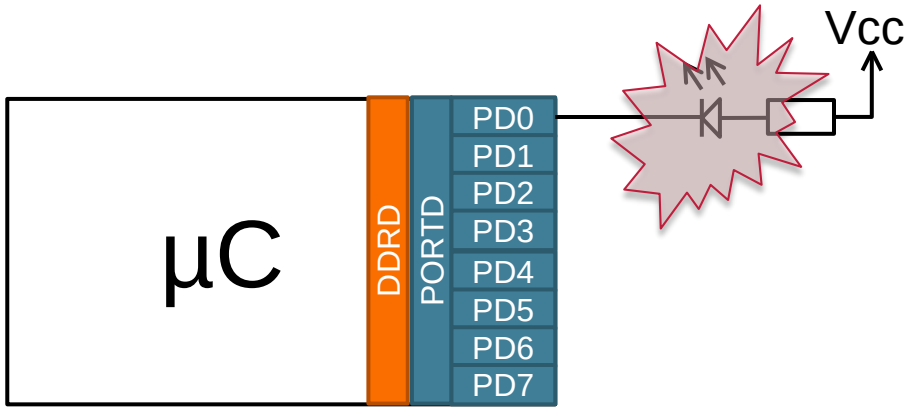


I/O-Ports - Beispiel

LED an Port-Pin PD0

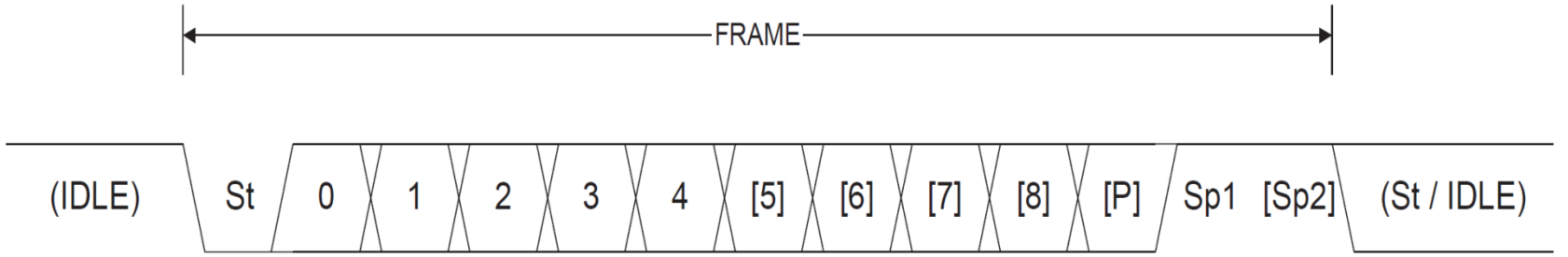
```
DDRD |= (1<<DDD0);          //setze PD0 als Ausgang

While(1){
        PORTD &= ~(1<<PD0); //setze PD0 auf 0
        PORTD |= (1<<PD0);  //setze PD0 auf 1
}
```



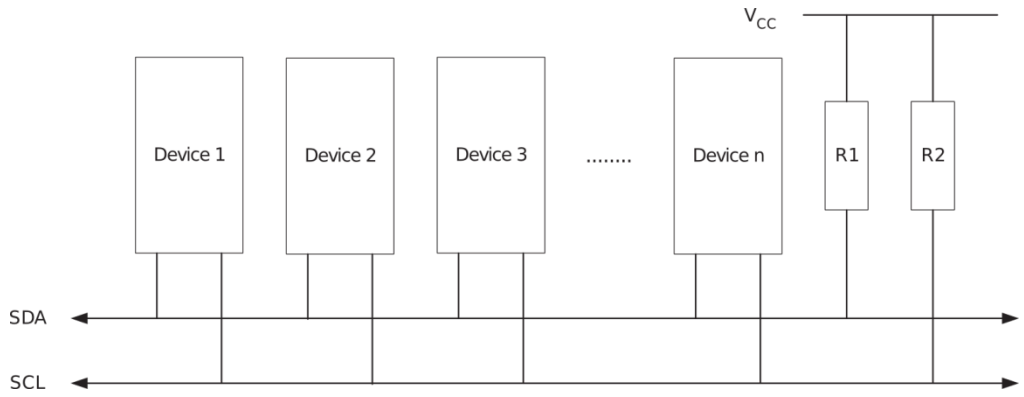
U(S)ART

- Seriell:
  - Datenbits werden *nacheinander* über eine Leitung gesendet
  - U(S)ART:
    - Universal synchronous/asynchronous Receiver/Transmitter, a.k.a. „COM-Port“
  - Asynchron: ohne Taktleitung, synchron: mit

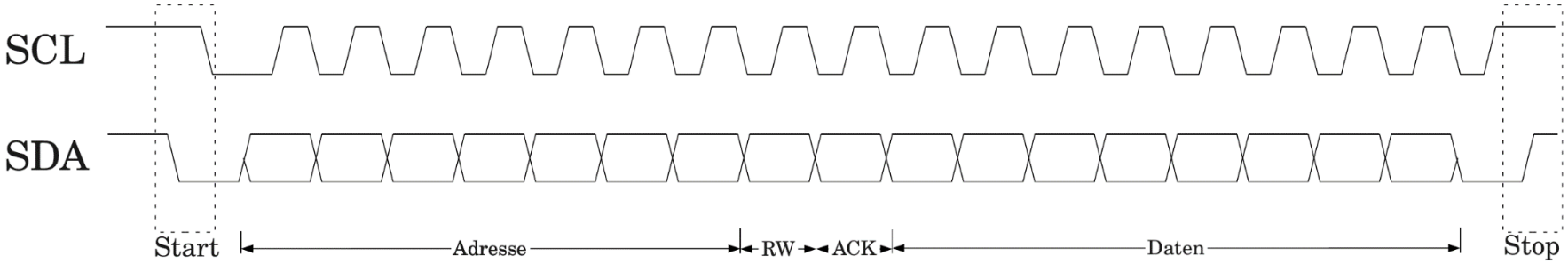


I²C

- Inter- Integrated Circuit bus
  - „Fernseher-Bus“
- Master/Slave-Bus:
  - Ein Master gibt Takt an
  - Viele Bausteine können adressiert werden
  - Pro Bus hat jeder Baustein eindeutige ID



I²C - Beispiel

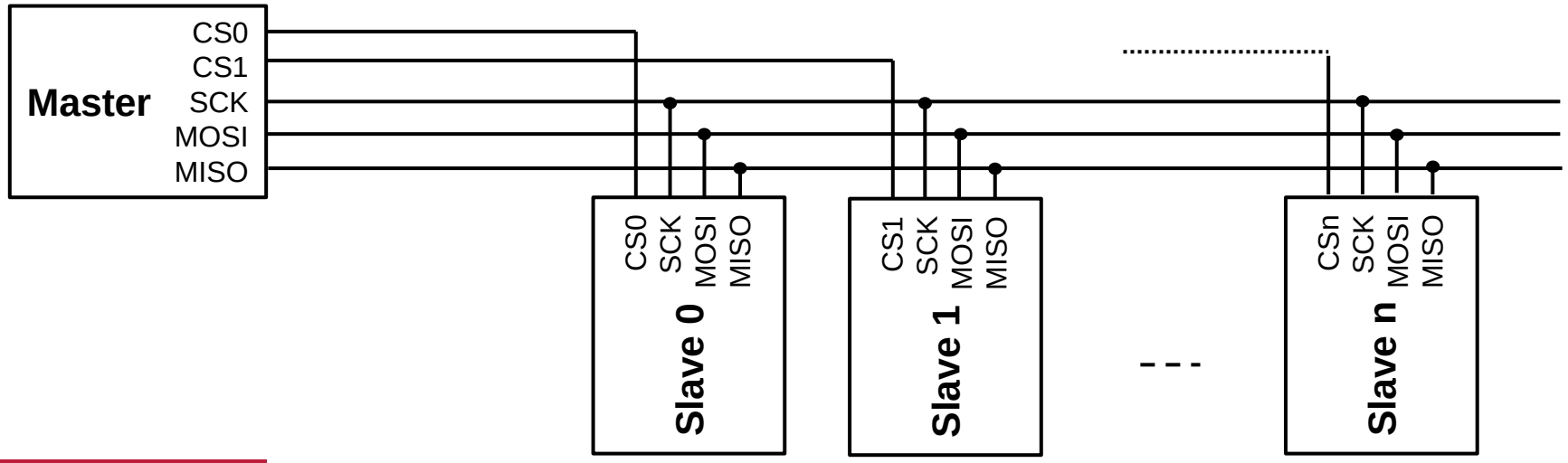


Ablauf:

- Startsignal (SDA -> low während SCL = high)
- 7 Adressbits (Adresse des gewählten Bausteins)
- 1 Bit read/write
- Slave mit gewählter Adresse bestätigt mit ACK
- Daten werden synchron zum Takt geschrieben/gelesen
- Stopsignal (SDA -> high während SCL = high)

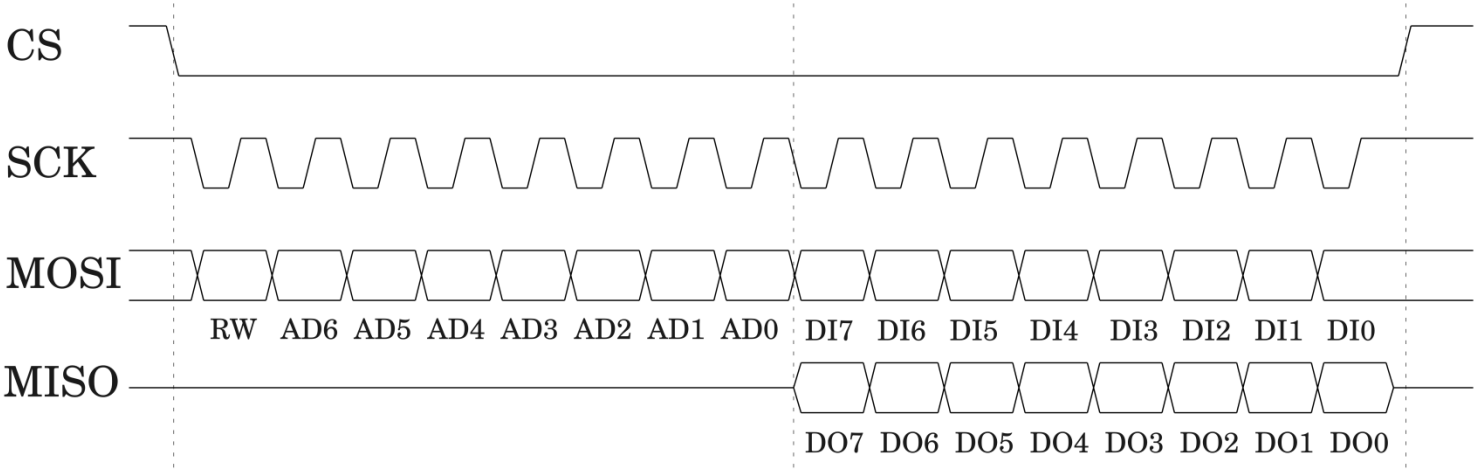
SPI

- Serial Peripheral Interface
- Master/Slave System
  - Master adressiert Slave über ChipSelect-Leitung
- 1-2 Datenleitungen, Clock-Leitung, CS
- SPI Topologie:



SPI – Beispiel (4 Wire-Mode)

- Auswahl des Bausteins vie Chip Select (CS)
- 1 Bit read/write
- 7 Adressbits der Register des Bausteins (z.B. Sensordaten)
- Synchron zum Takt:
  - Daten in Register(Adresse) im Baustein schreiben (MOSI)
  - Daten aus Regsiter(Adresse) im Baustein lesen (MISO)



Analog-Digitalwandler

- Unterschiedliche Verfahren
  - Flash-Wandler
    - Hoher Hardwareaufwand
    - schnell
  - Sukzessive Approximation:
    - Ein Komparator
    - Vergleichsspannung wird nachgeregelt
    - Langsamer