Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

- `switch` **Anweisung:**

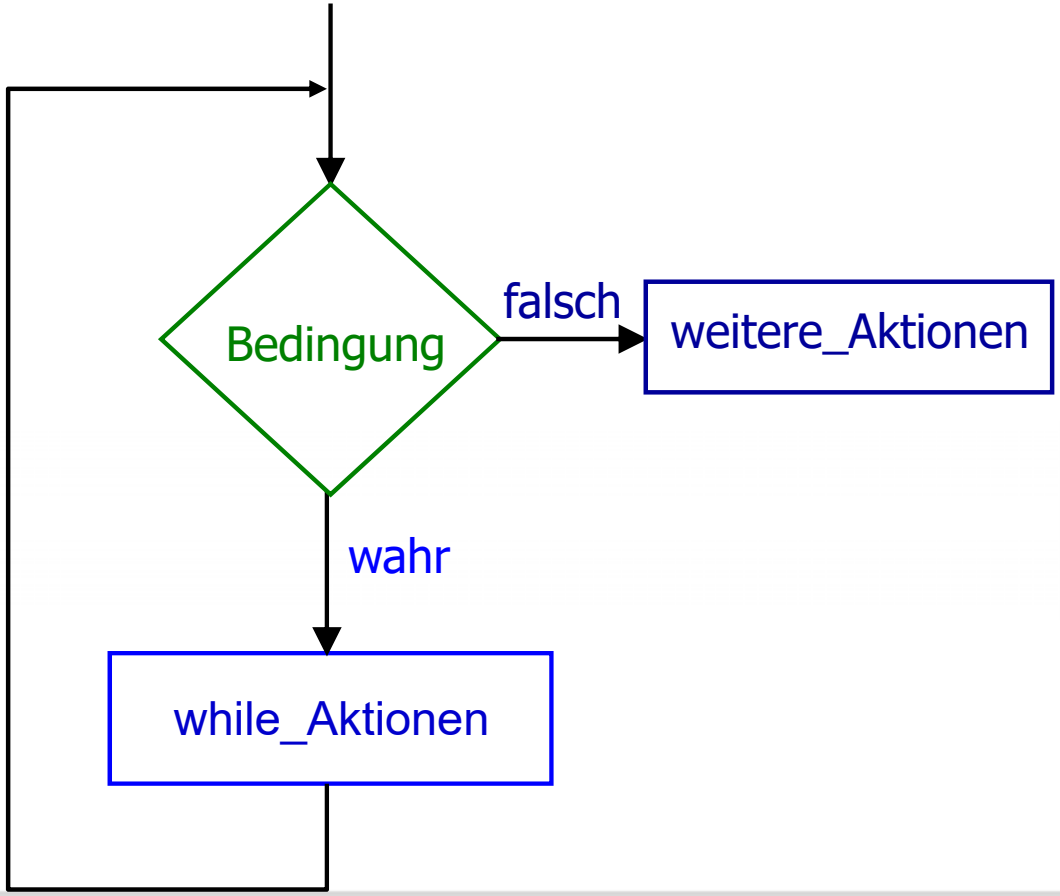
```
switch(ausdruck)
{
   case a: a Anweisungen    /* falls ausdruck == a */
   break;

   case b: b Anweisungen    /* falls ausdruck == b */
   break;

   case c: c Anweisungen    /* falls ausdruck == c */
   break;

   defaut: default Anweisungen    /* sonst */
   break;

}
```

Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

- **`while` Schleifen:**

```
while (Bedingung)
{
        while_Aktionen;
}
weitere_Aktionen;
```

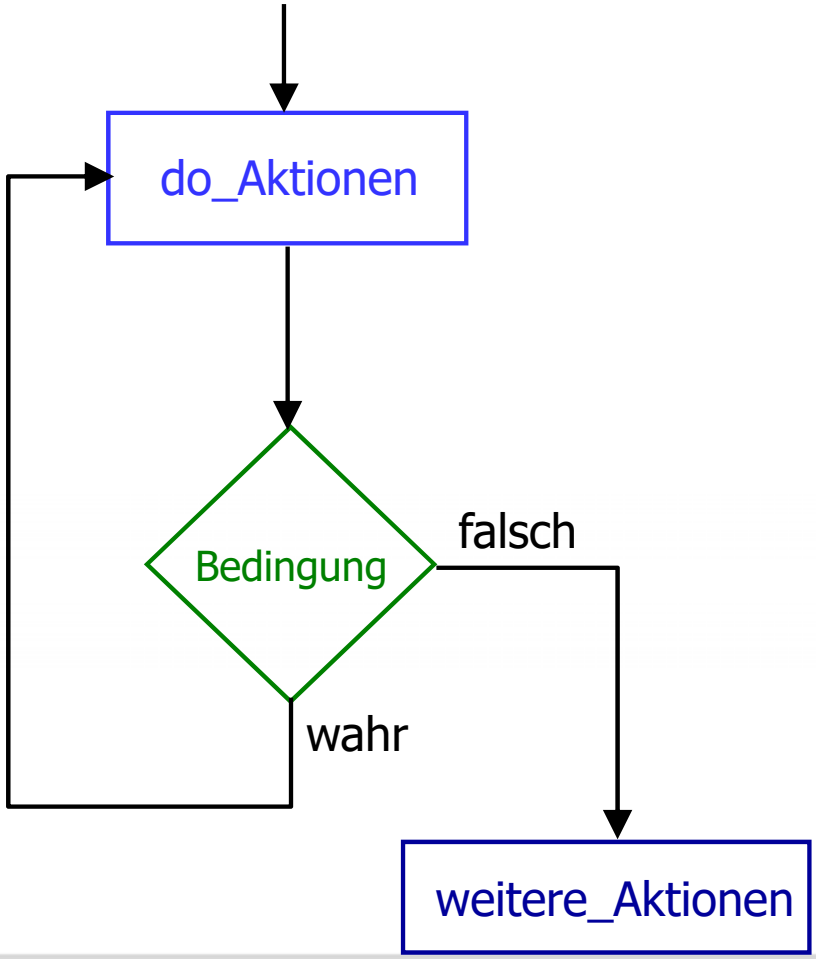


Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

- **`do while` Schleifen:**

```
do {
        do_Aktionen;
} while (Bedingung);

weitere_Aktionen;
```

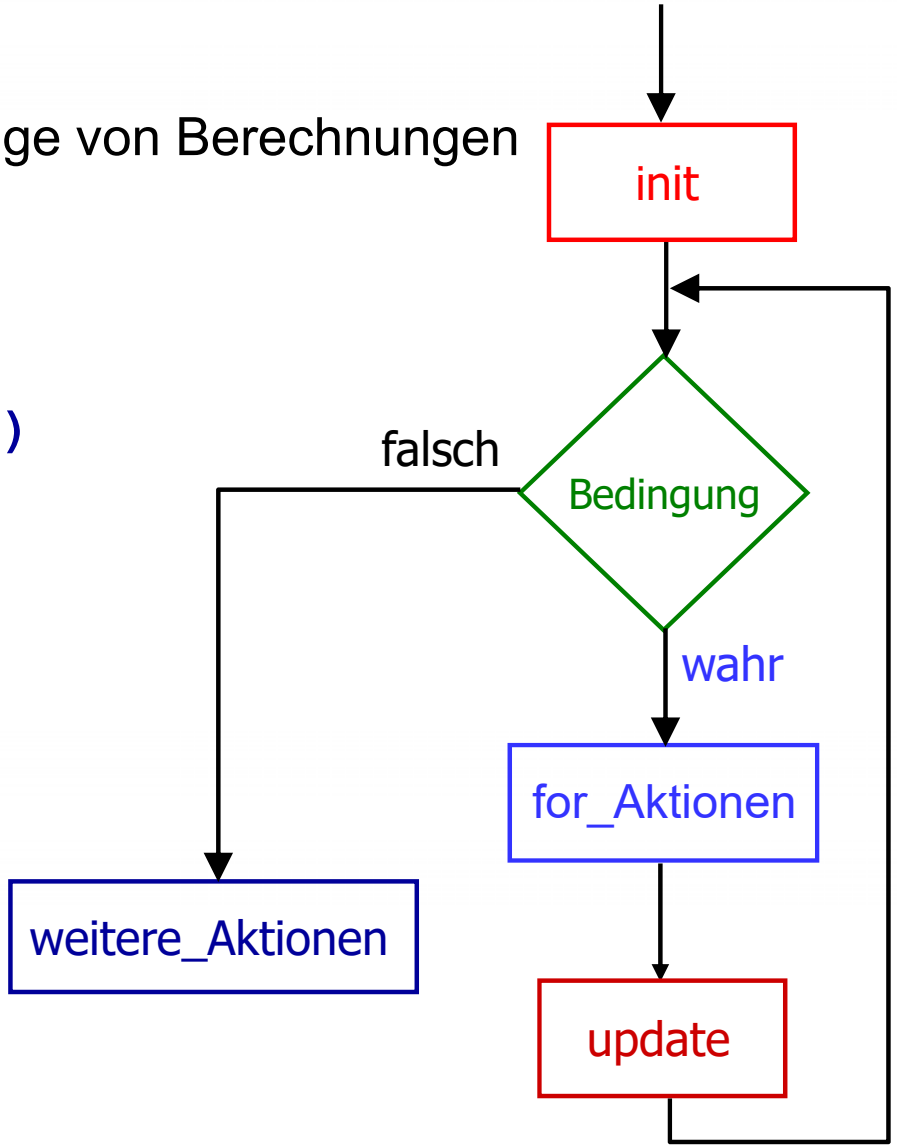


Grundlagen: Kontrollstrukturen

Kontrollstrukturen definieren die Reihenfolge von Berechnungen

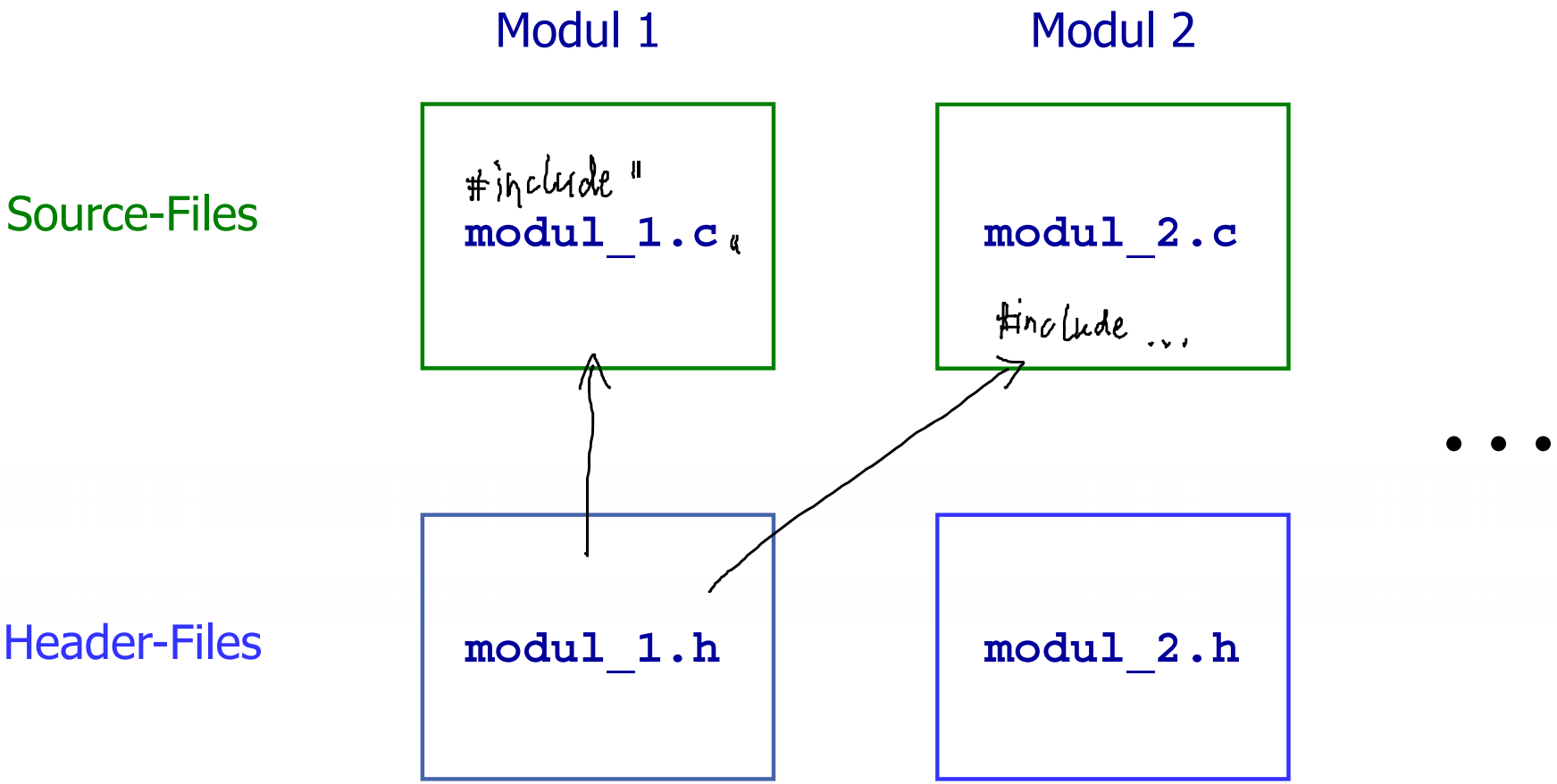
- **`for` Schleifen:**

```
for (init; Bedingung; update)
{
    for_Aktionen;
}
weitere_Aktionen;
```



2.5 Funktionen und Programmstruktur (1)

- Aufbau eines Programms



Funktionen und Programmstruktur (2)

- Aufbau eines Programms

Preprozessor Anweisungen

```
#include <stdio.h>          /*  Einbinden einer Bibliothek (I/O-Funktionen) */
#include "modul_1.h"        /*  Einbinden eines Moduls „modul_1"  */
#define EYE_COLOR blau      /* Globale Textersetzung im Programmtext, Makrodef. */
```

Globale Deklarationen und Definitionen:

```
extern int i;  /* Deklaration: es wird nur mitgeteilt, dass es
die Variable irgendwo gibt. Später muss der Linker überprüfen,
dass er in einer anderen Objektdatei (aus einer anderen
Quellcode-datei compiliert) den Speicherbereich dazu findet. */

int j;  /* Definition: Variable bekannt machen und Speicher-
bereich zuteilen, z.B. auf Stack oder Heap. Die Variable hat
jetzt schon einen Wert, nämlich den, der an dieser Stelle
(zufällig) im Speicher steht. */

j=42;  /* Initialisierung */

int fakultaet (int n);  /* Funktionsprototyp (nur Dekl.) */
```

Funktionen und Programmstruktur (3)

```
int fakultaet (int n) {    /* Funktionsdefinition */
   ...
}
```

```
void main () {
   printf("Hallo KIT \n");
}
```

- Jedes Programm enthält eine Funktion „main"
- Unterprogramme werden in C in Form von **Funktionen** definiert, die **Parameterlisten** und **Ergebnistypen** haben
- Die Abarbeitung eines Programms beginnt stets mit der Ausführung von „main"
- Innerhalb von „main" können die weiteren definierten Funktionen aufgerufen werden

Funktionen und Programmstruktur (4)

- Funktionen werden global definiert
- Rekursive Funktionsaufrufe sind zulässig: eine Funktion darf sich selbst aufrufen
  - Beispiel (Fakultätsberechnung):

```
int fakultaet(int n){
  if ( n == 1 )
    return 1;
  else
    return n * fakultaet(n-1);
}
```

Parameterübergabe an Funktionen

- **Call by Value**
  - Normalfall in C
  - Es wird eine Kopie des aktuellen Parameters an die Funktion übergeben
  - Funktionen können den Wert der Kopien ändern, ohne Auswirkungen auf den Wert des aktuellen Parameters beim Aufrufer
- **Call by Reference**

Globale und Lokale Variablen

- **Globale Variablen** sind im gesamten Programm einschließlich aller Unterprogramme bekannt (sollte nicht zu häufig verwendet werden)

- **Lokale Variablen** sind innerhalb einer Funktion oder eines „Blockes“ deklarierte Variablen

```
#include <stdio.h>
int global = 0;  /* Das ist eine globale Variable */

void main () {
  int lokal = 1;       /* Das ist eine lokale Variable in main */
  printf("Global %d, Lokal %d\n", global, lokal);
    {
      int lokal = 2;  /* Das ist eine lokale Variable im Block */
      printf("Global %d, Lokal %d\n", global, lokal);
    }
  printf("Global %d, Lokal %d\n", global, lokal);
}
```

Speicherklassen

- Definieren die Lebensdauer und den Gültigkeitsbereich (Sichtbarkeit) von Variablen und/oder Funktionen
- In C gibt es die Speicherklassen:
  - **auto** wird für lokale Variablen eingesetzt; Objekte gelten nur solange wie der Bereich, in dem sie definiert wurden
  - **register** ist ein Hinweis (!) an den Compiler, dass er die folgende lokale Variable in einem Register der CPU behalten soll
    - Sollte nur für Variablen verwendet werden, auf die schnell/häufig zugegriffen werden muss, z.B. bei Zählern
  - **static** wird statisch Speicherplatz zugewiesen
    - Quasi wie globale Variablen, aber nur mit lokaler Sichtbarkeit
    - Kann genutzt werden um Zustand vom einen Funktionsaufruf in den nächsten zu retten
  - **extern** deklariert eine globale Variable, die in irgendeinem anderen Programm-Modulen definiert wurde

2.6 Zeiger und Vektoren (1)

- Mächtigstes Werkzeug der Sprache C zur maschinennahen Programmierung
- Effiziente Programmierung durch Vermeidung von Kopieroperationen
- Pointerarithmetik → direkter Zugriff auf Elemente strukturierter Daten
- Fortgeschrittene Verwendung von Pointern kann zu kryptischem Code und Unleserlichkeit führen ☹
- Macht das Programmieren mit C fehleranfällig

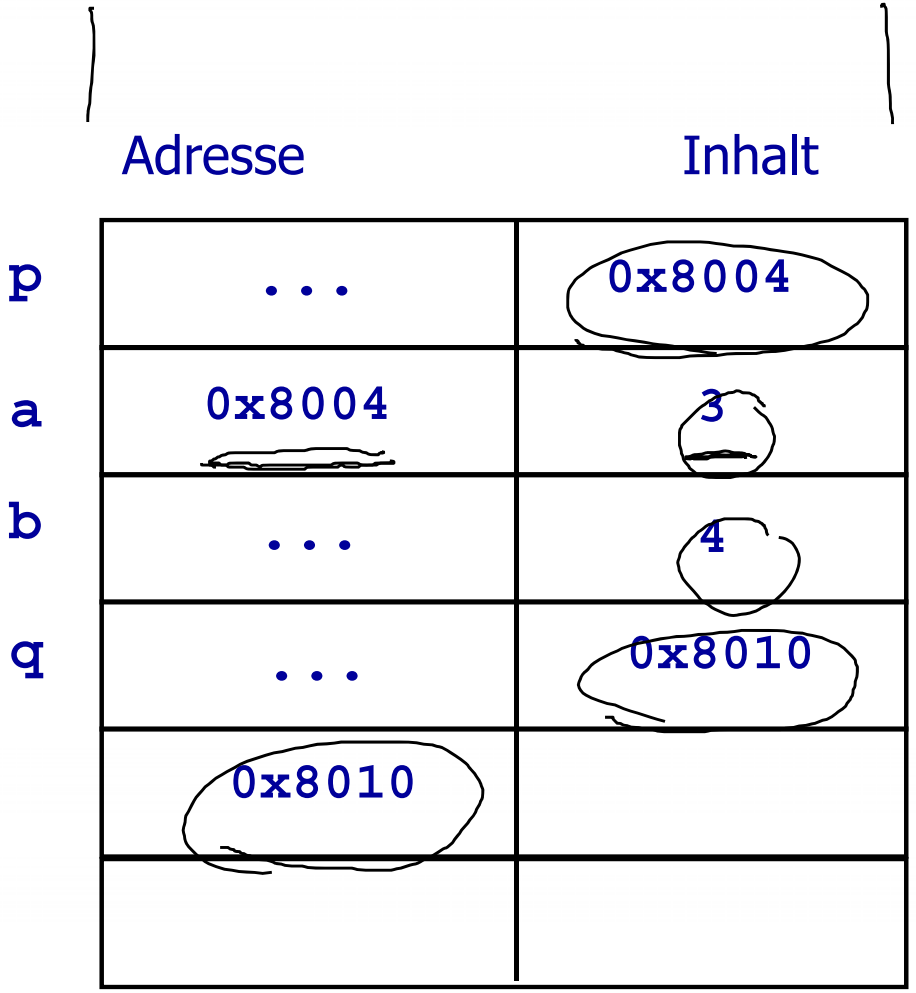
Zeiger und Vektoren (2)

- Ein **Zeiger „pointer“** enthält eine Adresse, die auf Daten verweist

```
int *p;                 /* „p“ ist vom Typ „Zeiger auf int“
                           „*p“ (Dereferenzierung) ist vom Typ „int“
*/
int *q;
int a = 3;
int b;

p = &a;                 /* der unäre Operator „&“ liefert die
                           Adresse von a */
b = *p + 1;             /* der unäre Operator „*“ liefert den
                           Wert, auf den p zeigt */
q = (int*) 0x8010       /* Typumwandlung einer Adresse in
                           Hexadezimalschreibweise auf
                           „Zeiger auf int“ */
```

Zeiger und Vektoren (3)

```
int *p;
int *q;
int a = 3;
int b;

p = &a;
b = *p + 1;
q = (int*) 0x8010
```



Zeiger und Vektoren (4)

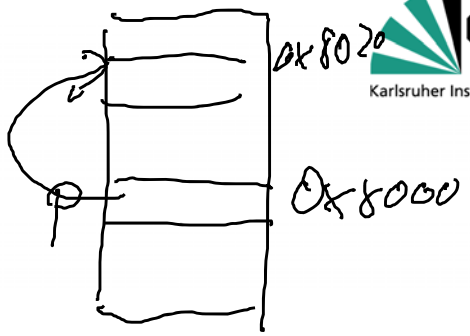
```
int n[] = {1, 2, 3};
char c[] = {'a', 'b', 'c'};
char s[] = "de";
s[1] = 'f';
int *i = n;   // i = &(n[0]);
*i++ = 12;    // *(i++) = 12;
*i++ = 13;    // *i = 13; ++i;
```

	Adresse	Inhalt
n[0]	0x8004	1
n[1]	0x8008	2
n[2]	0x800C	3

	Adresse	Inhalt
c[0]	0x8014	'a'
c[1]	0x8015	'b'
c[2]	0x8016	'c'

	Adresse	Inhalt
s[0]	0x8024	'd'
s[1]	0x8025	'f'
s[2]	0x8026	'\0'

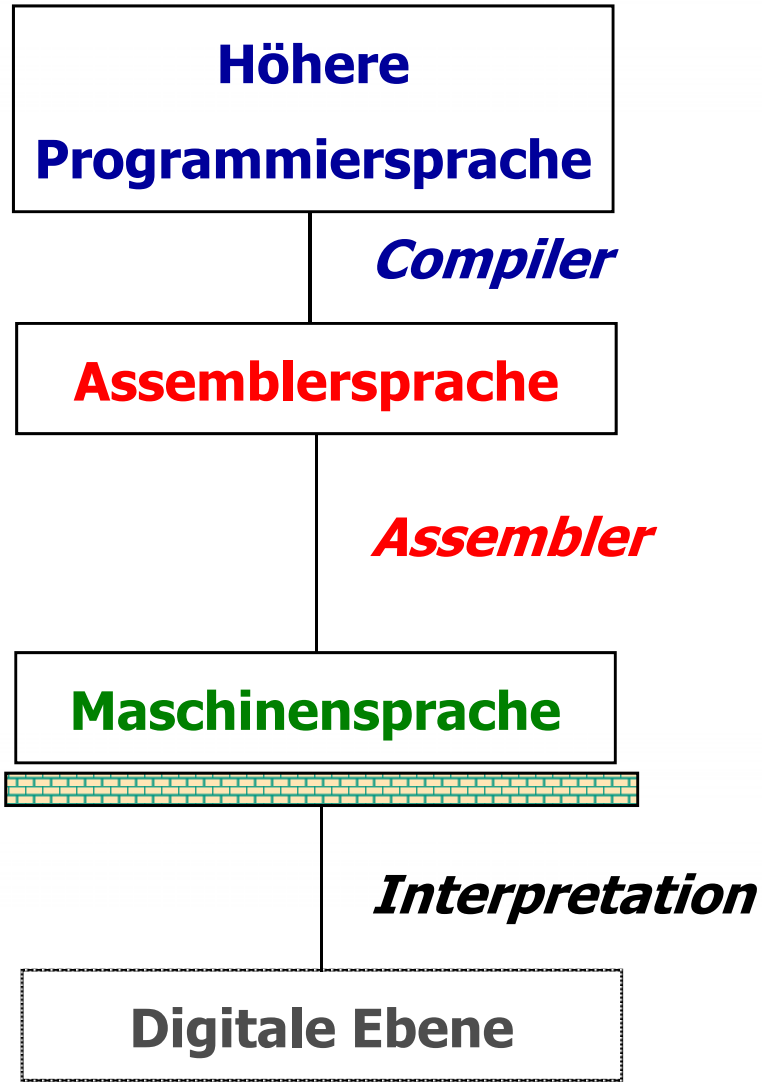
Parameterübergabe an Funktionen



- **Call by Value**
  - Normalfall in C
  - Es wird eine Kopie des aktuellen Parameters an die Funktion übergeben
  - Funktionen können den Wert der Kopien ändern, ohne Auswirkungen auf den Wert des aktuellen Parameters beim Aufrufer

- **Call by Reference**
  - In C nur indirekt mit Hilfe von Zeigern realisierbar
  - Der Übergabeparameter ist eine Variable und die aufgerufene Funktion erhält die Speicheradresse dieser Variablen
  - Wenn die Funktion den Wert des Parameters verändert, ändert sie den Inhalt der zugehörigen Speicherzelle → der Wert der Variablen beim Aufrufer der Funktion ändert sich auch

Hierarchie




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $15,  0($2)
lw    $16,  4($2)
sw    $16,  0($2)
sw    $15,  4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

```
ALUOP[0:3] ← InstReg[9:11] & MASK
```

C – Entwicklungswerkzeuge (1)

- Texteditor
- Compiler
  - Linux: gcc (Gnu Compiler Collection)
  - OS X: gcc, clang
  - Windows: msvc (MS Visual Studio)
- Buildtool (bei mehr als einer C-Datei)
  - CMake
  - Integrated Development Environment (IDE)
    - Eclipse, QtCreator, ...
    - Windows: VisualStudio

C – Entwicklungswerkzeuge (2)

Ubuntu-Linux: Installation

- `sudo apt-get install build-essential`
  - dpkg-dev
  - gcc
  - libc-dev
  - make
- `sudo apt-get install cmake`

Ubuntu-Linux: Übersetzung

- `gcc -o program program.c`

C – Entwicklungswerkzeuge (3)

OS X: Installation

- Xcode installieren
  - https://developer.apple.com/xcode/index.php
  - Kommandozeilentools mitinstallieren
- CMake installieren
  - http://www.cmake.org

OS X: Übersetzung

- `gcc -o program program.c`
- `clang -o program program.c`

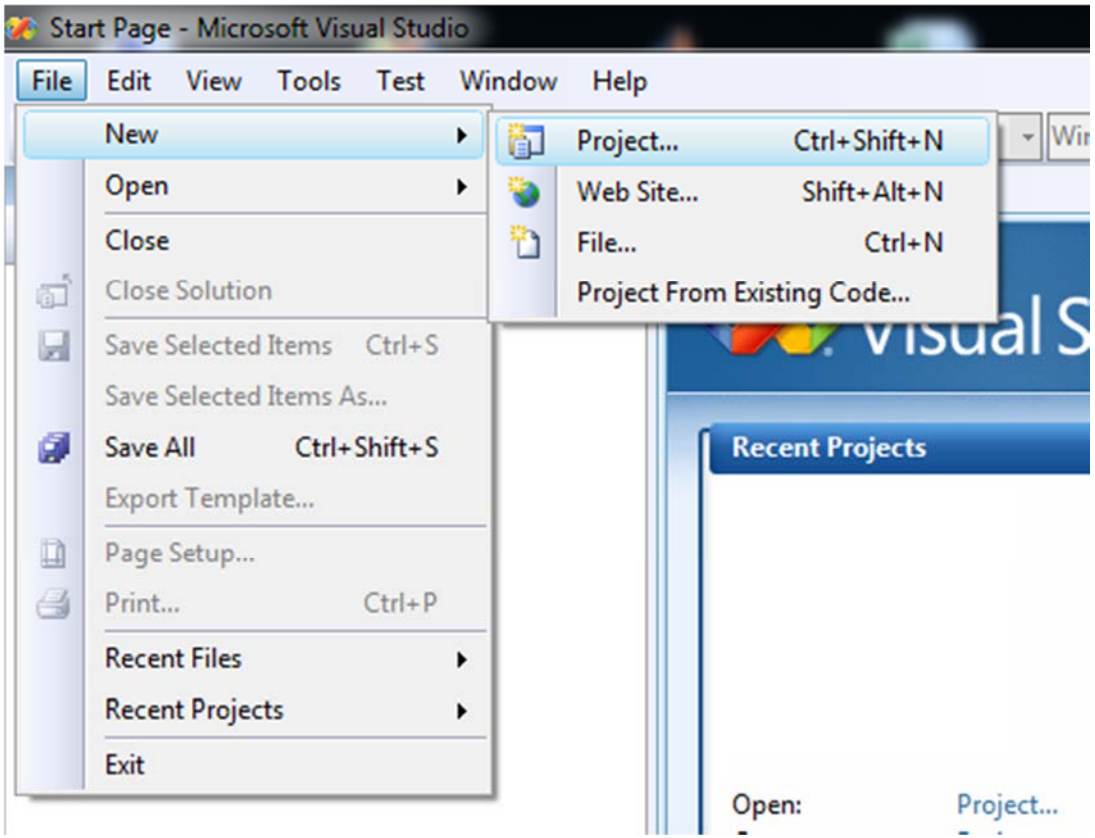
C – Entwicklungswerkzeuge (4)

Windows Installation

- VisualStudio installieren
  - Testversion: https://www.microsoft.com/germany/visualstudio/
  - Vollversion (kostenlos für Studenten): http://www.scc.kit.edu/dienste/7929.php
- CMake installieren
  - http://www.cmake.org

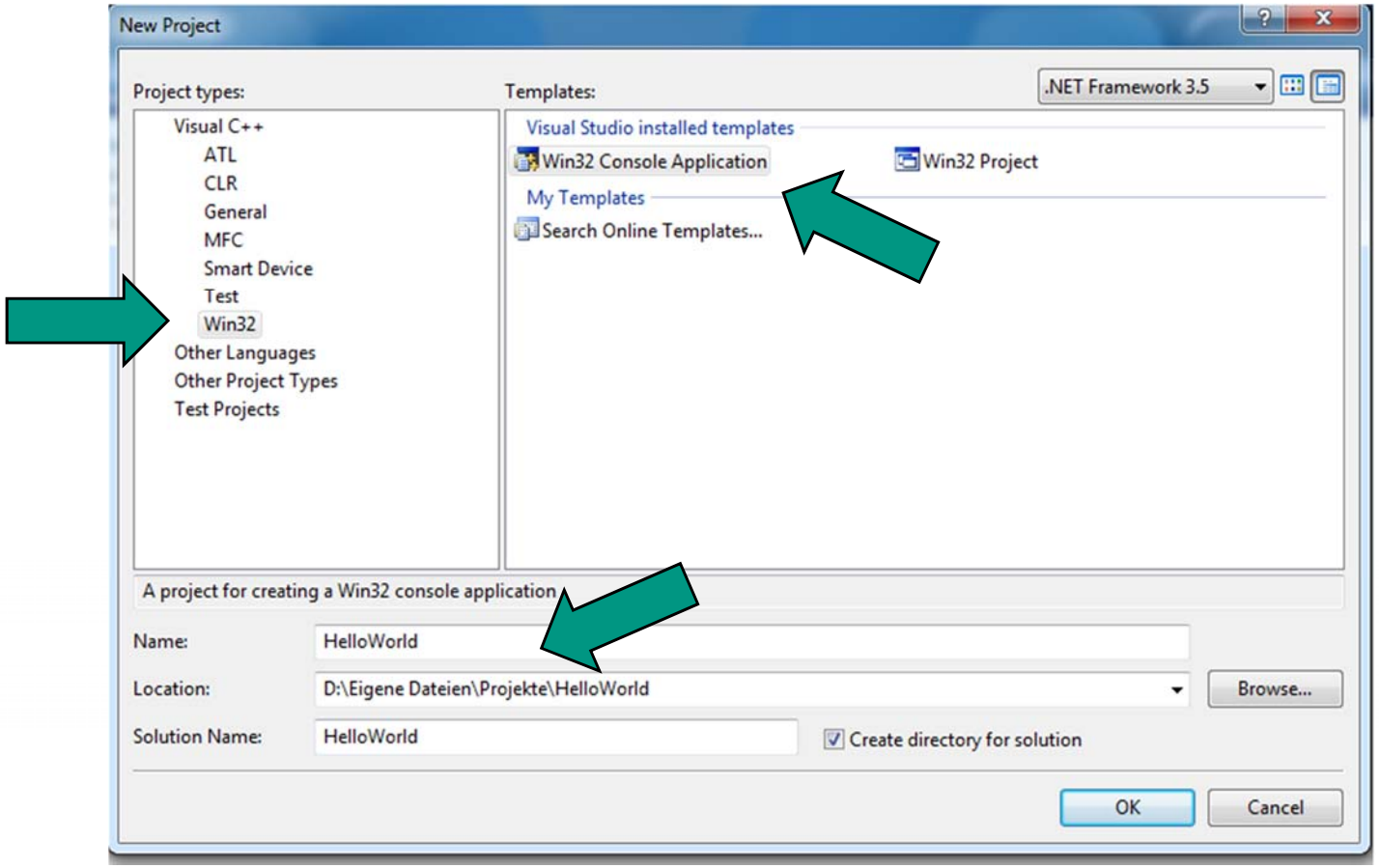
HelloWorld in VisualStudio (1)

- Visual Studio installieren
- Neues Projekt erstellen:



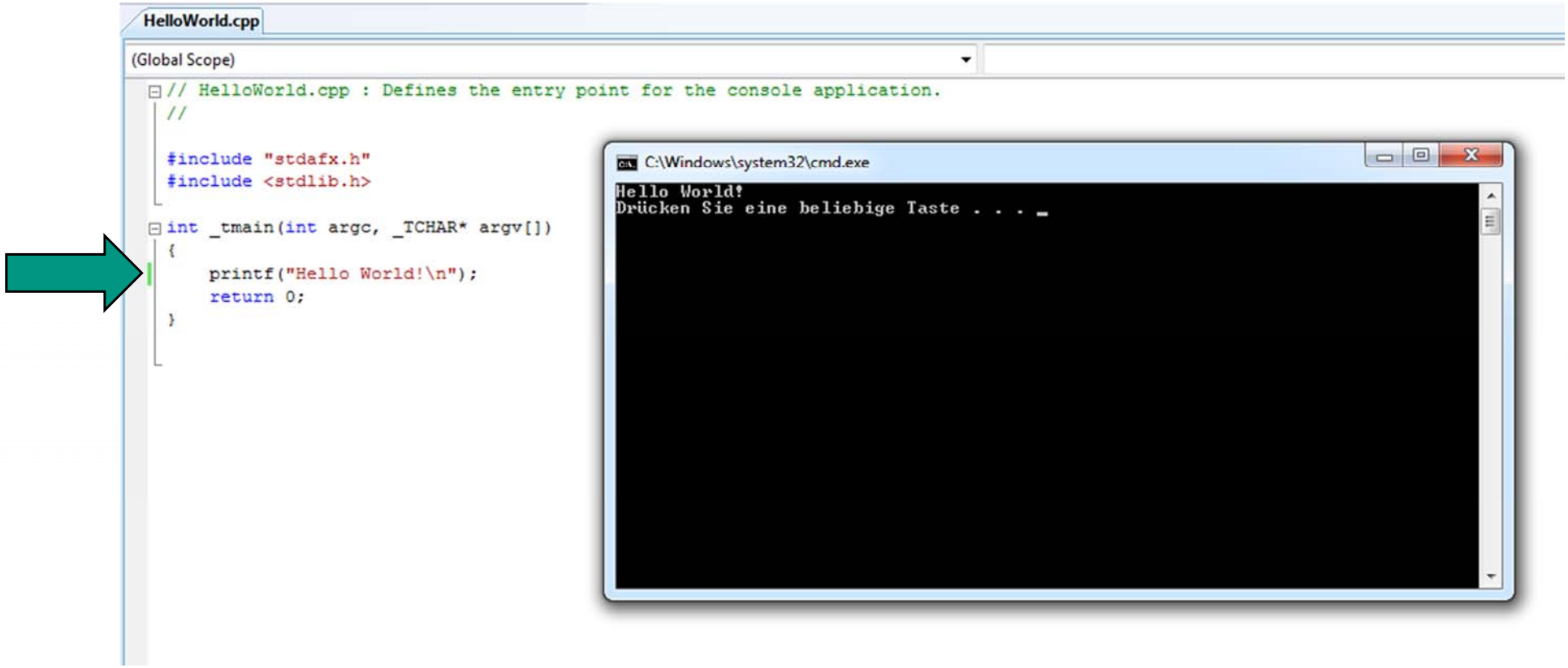
HelloWorld in VisualStudio (2)

- HelloWorld Projekt erstellen und bestätigen:



HelloWorld in VisualStudio (3)

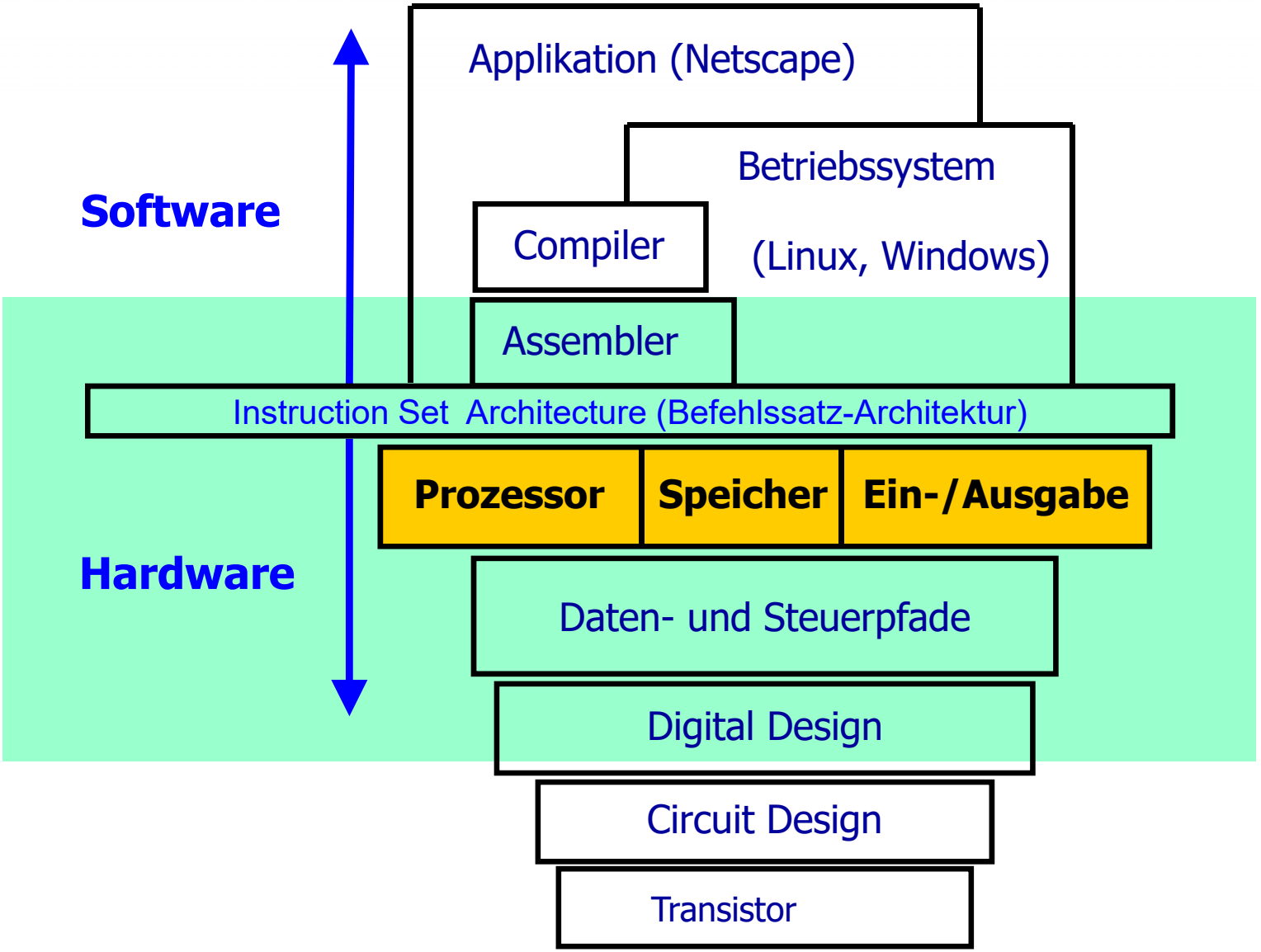
- Einfügen der „Hello World“ Ausgabe
- Compilieren mit F7, Ausführen mit Ctrl+F5



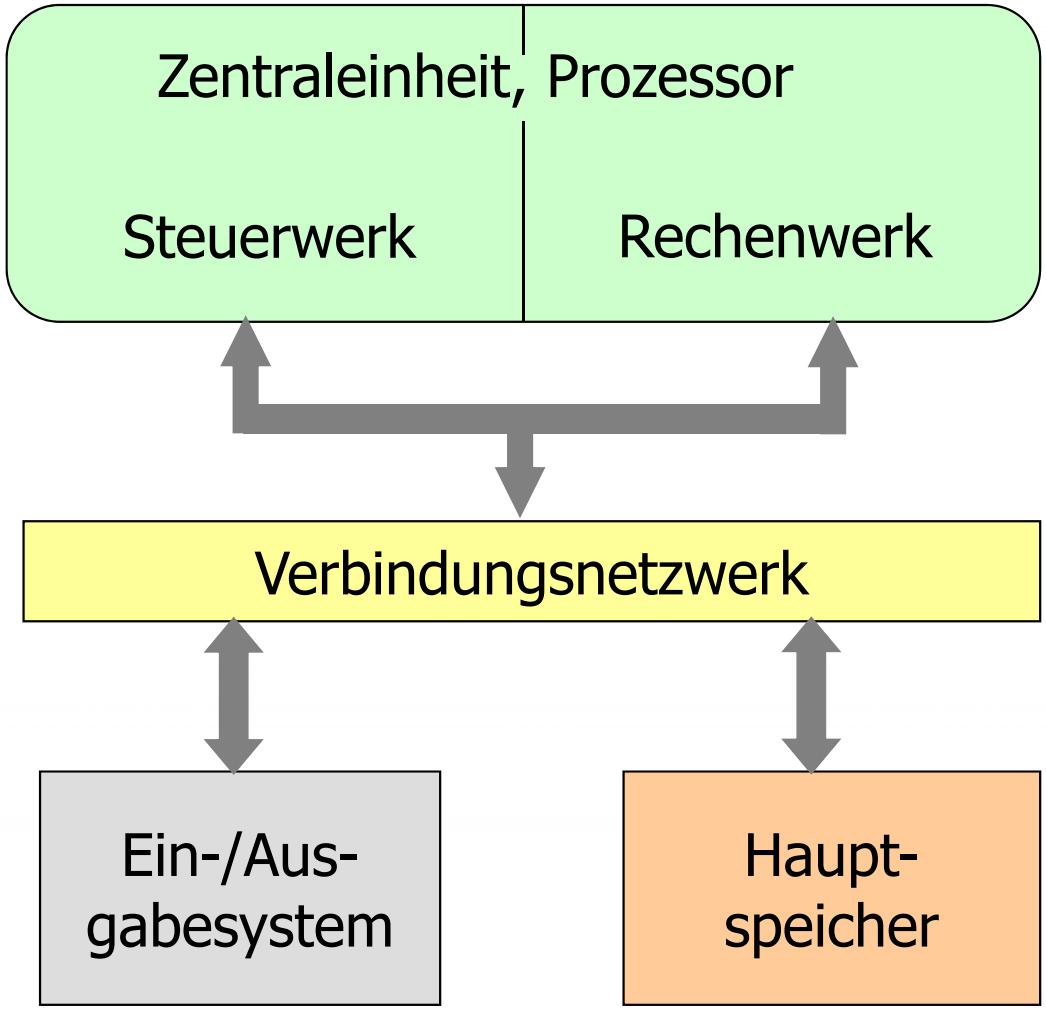
Kapitel 3

- Programmierbarer Universalrechner
  - Organisationsprinzip programmierbaren Universalrechners
  - Aufbau eines einfachen Mikroprozessors
    - Steuerwerk (Leitwerk)
    - Rechenwerk
    - Speicherwerk
    - Ein-Ausgabewerk
    - Verbindungsstrukturen
  - Maschinenbefehlszyklus

Rechnerorganisation



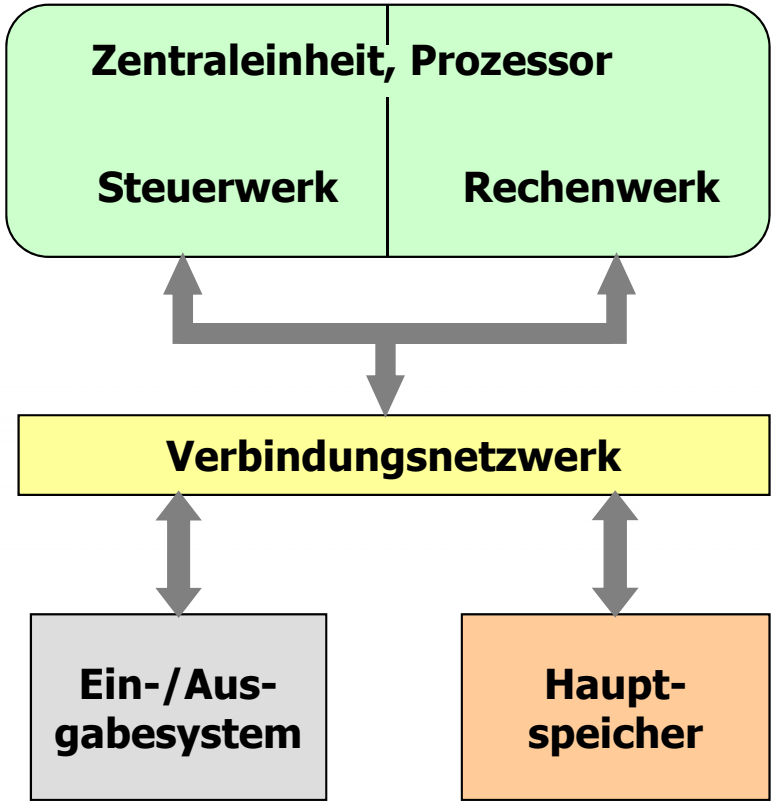
3.1 Organisationsprinzip des programmierbaren Universalrechners



Komponenten des programmierbaren Universalrechners

- Zentraleinheit (central processing unit, CPU, Prozessor)
  - Verarbeitet Daten gemäß eines Programms
  - Besteht aus Steuerwerk und Rechenwerk

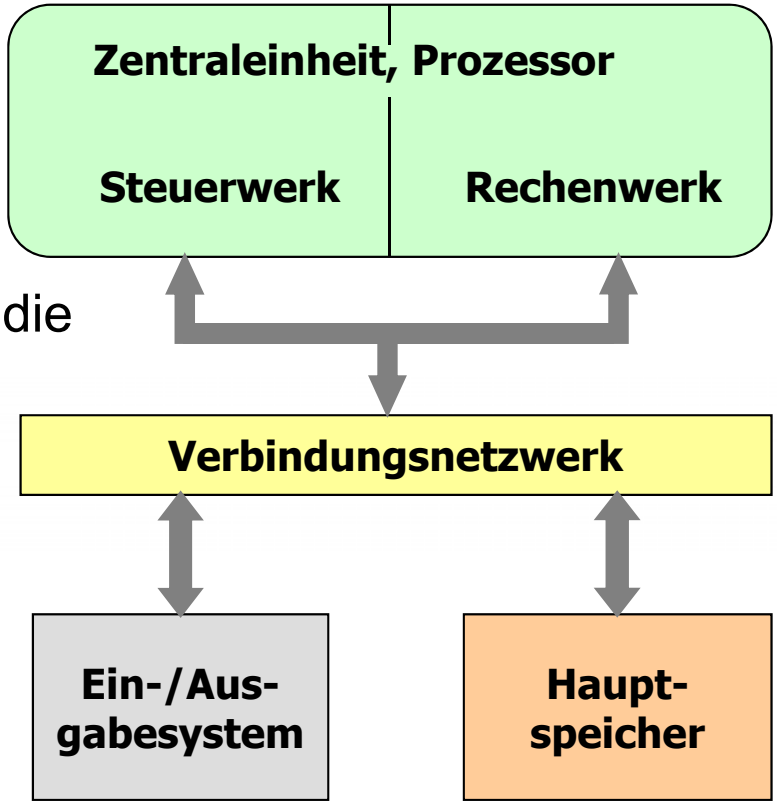
- Steuerwerk (Leitwerk, control unit, CU)
  - Holt die Befehle eines Programms aus dem Speicher
  - Dekodiert sie
  - Steuert ihre Ausführung in der verlangten Reihenfolge durch Steuer- und Status-Signale



Komponenten des programmierbaren Universalrechners

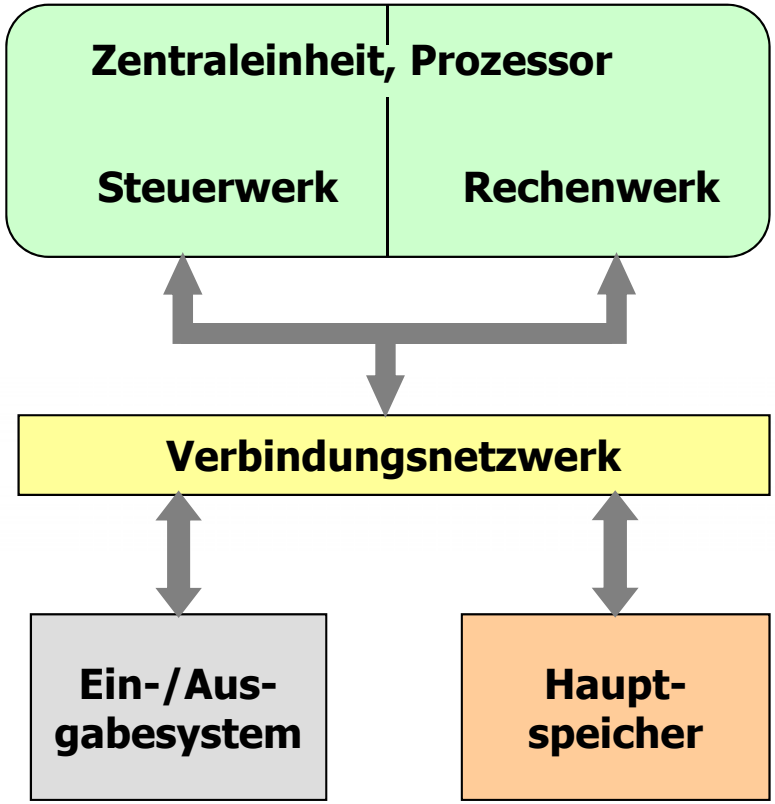
- Zentraleinheit (central processing unit, CPU, Prozessor)
  - Verarbeitet Daten gemäß eines Programms
  - Besteht aus Steuerwerk und Rechenwerk

- Rechenwerk (Operationswerk, Ausführungseinheit, ALU)
  - Führt die Rechenorperationen (z.B. arithmetisch/logische Operationen aus)
  - Auszuführende Operationen werden durch die Steuersignale bestimmt
  - Liefert seinerseits Statussignale an das Steuerwerk zurück



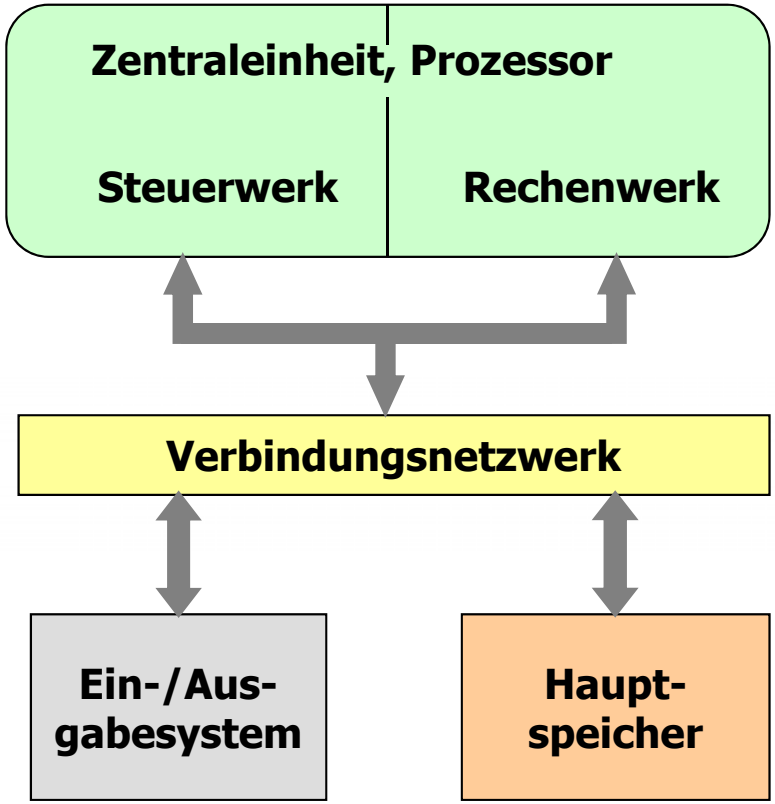
Komponenten des programmierbaren Universalrechners

- Hauptspeicher
  - Jede Speicherzelle ist eindeutig durch ihre Nummer (Adresse) identifizierbar
  - Dort werden Programme und Daten abgelegt (von-Neumann-Konzept)
  - Alternativ: Harvard-Architektur mit getrenntem Programm- und Datenspeicher



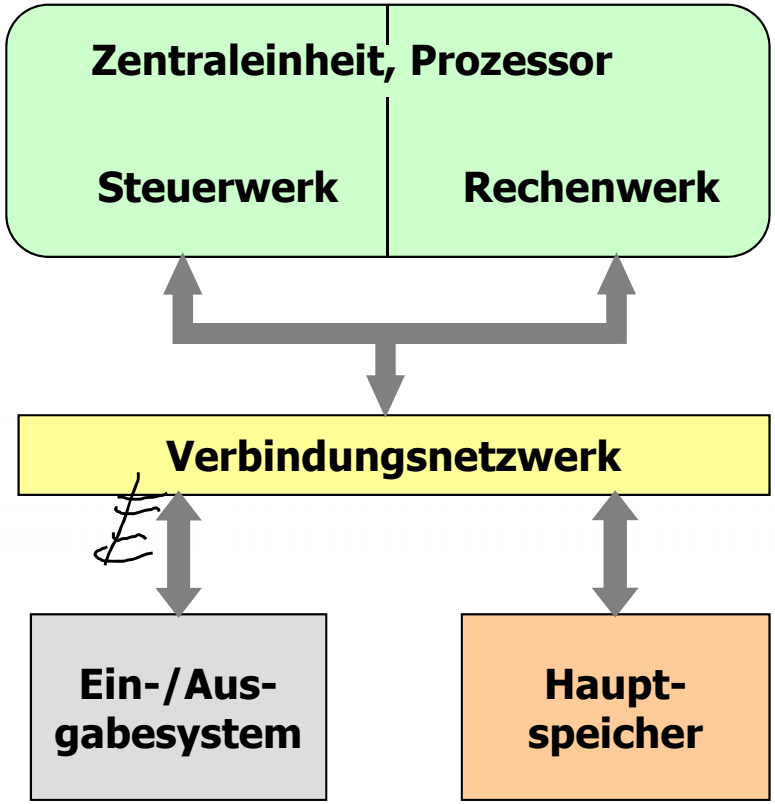
Komponenten des programmierbaren Universalrechners

- Verbindungsnetzwerk
  - Verbindet die Komponenten für den Austausch von Daten
  - Bus:
    - Adressleitungen: Leitungen, auf denen die Adressinformation transportiert wird (unidirektional)
    - Datenleitungen: Transportieren Daten und Befehle von/zum Prozessor (bidirektional)
    - Steuerleitungen: Geben Steuer- / Status informationen von/zum Prozessor (uni- oder bidirektional)



Komponenten des programmierbaren Universalrechners

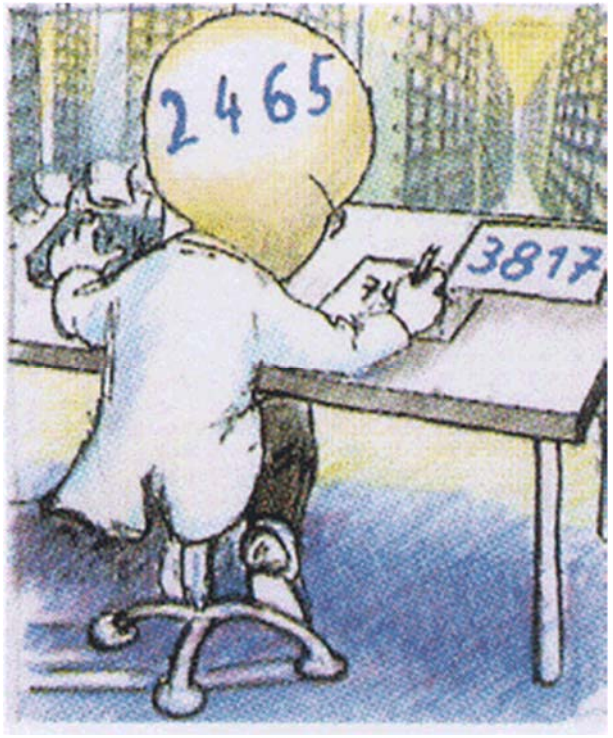
- Ein-/Ausgabesystem
  - Geräte zur Eingabe von Daten und Programmen und zur Ausgabe der verarbeiteten Daten (Bildschirme, Drucker, Terminals, ...)
  - Die peripheren Geräte sind über Ein-/Ausgabe-Schnittstellen mit dem Rechner verbunden



Phasen der Befehlsausführung

Holphase

Decodierphase



Ausführungsphase

Phasen der logischen Befehlsausführung

Holphase:

den nächsten Befehl aus dem Speicher in das Befehlsregister laden

Dekodierphase:

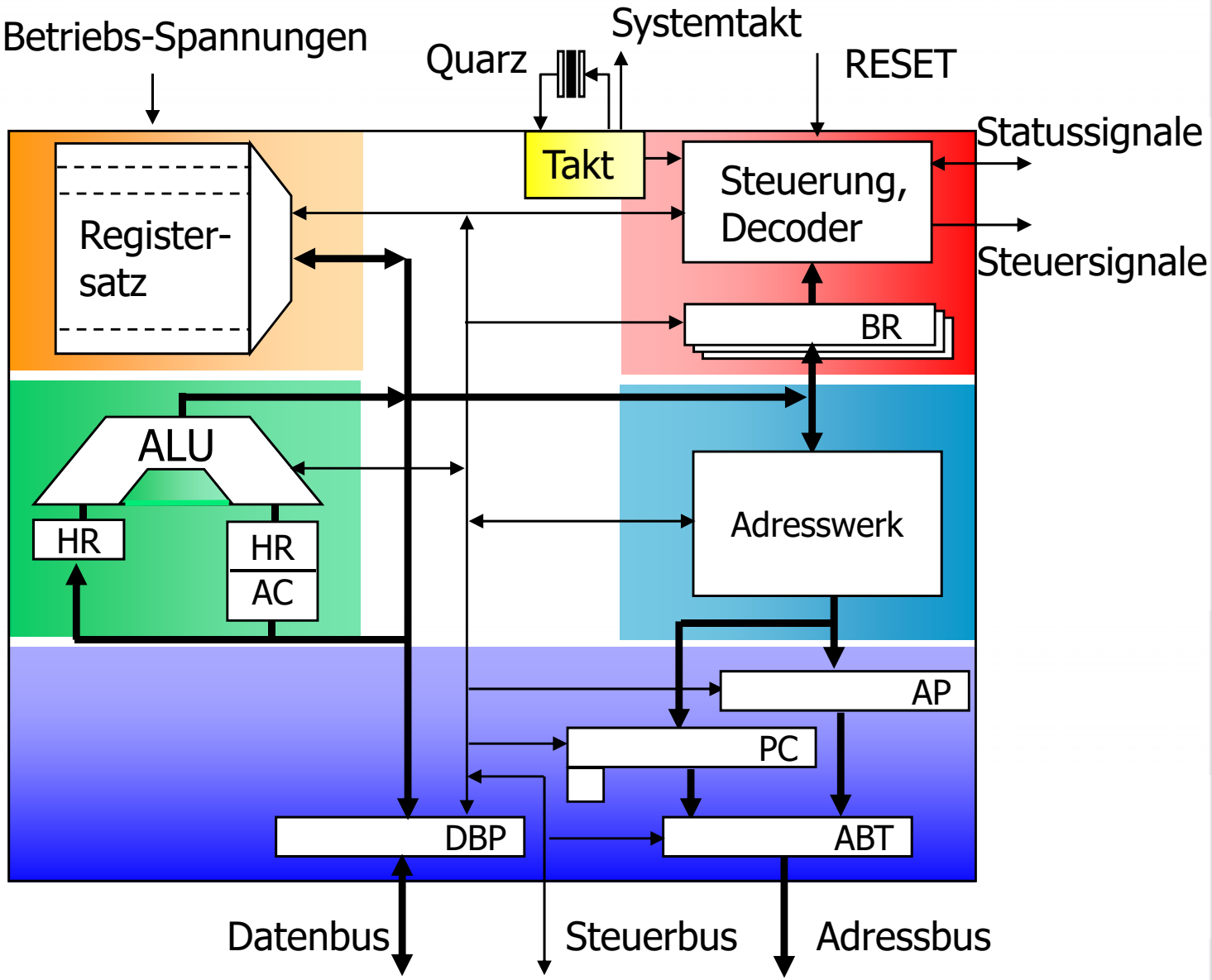
Dekodierung des Maschinenbefehls

Ausführungsphase:

Ausführung des Befehls

3.2 Aufbau eines einfachen µP

- HR: Hilfsregister
- AC: Accumulator
- BR: Befehlsregister
- AP: Adress Puffer
- PC: Program Counter
- ABT: Adressbus Puffer
- DBP: Datenbus Puffer



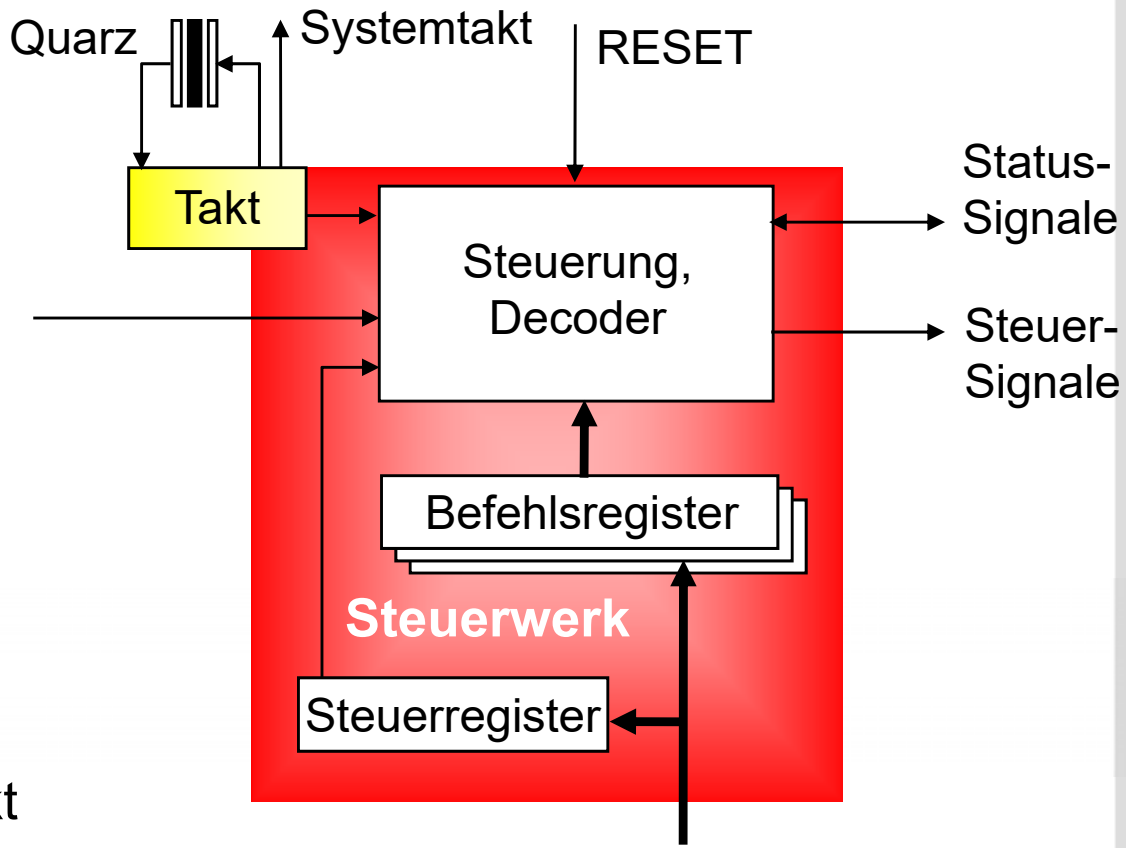
Aufbau eines einfachen µP

- **Steuerwerk**
- **Rechenwerk**
- **Registersatz**
- **Adresswerk**
- **Systembusschnittstelle**
- **Interne Busse**

3.2.1 Steuerwerk

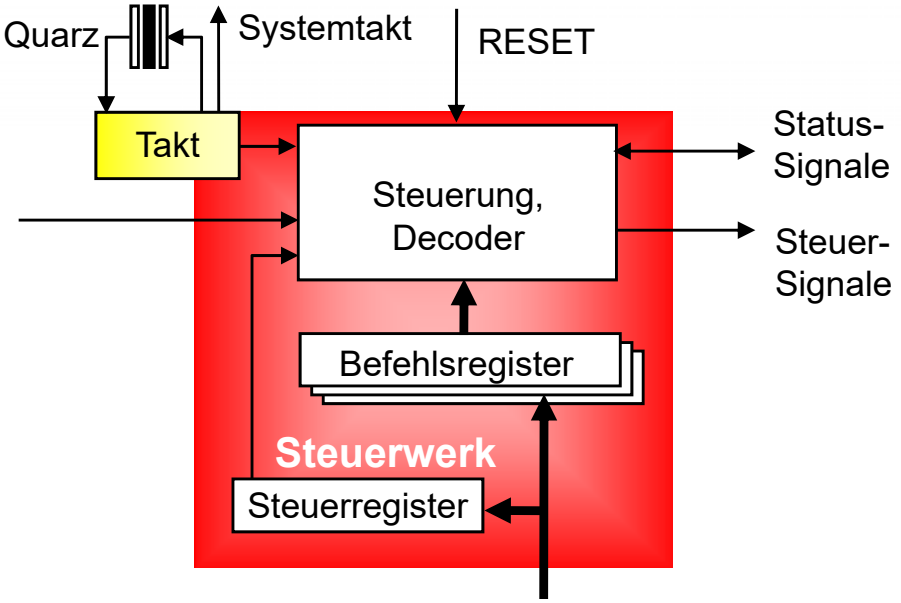
- Steuert die Systemkomponenten

- Befehlsregister
  - Enthält den gerade ausgeführten Befehl
- Dekoder
  - Dekodiert Befehlsworte
  - Als mikroprogrammiertes Schaltwerk realisiert (im Beispiel)
- Taktgenerator
  - Erzeugt den vom externen Quarz festgelegten Systemtakt



Taktgenerator

- Aufgaben des Taktgenerators
  - Taktfrequenz herstellen
  - Erzeugung eines mit dem Prozessortakt synchronisierten Rücksetzsignals
  - Taktgenerator ist bei modernen Mikroprozessoren on Chip (mit externem Quarz verbunden)



- Beim Rücksetzen kann das Steuerwerk ggf. eine längere Initialisierungsroutine durchlaufen (untypisch)
  - Dann muss das Rücksetzsignal genauen zeitlichen Spezifikationen genügen

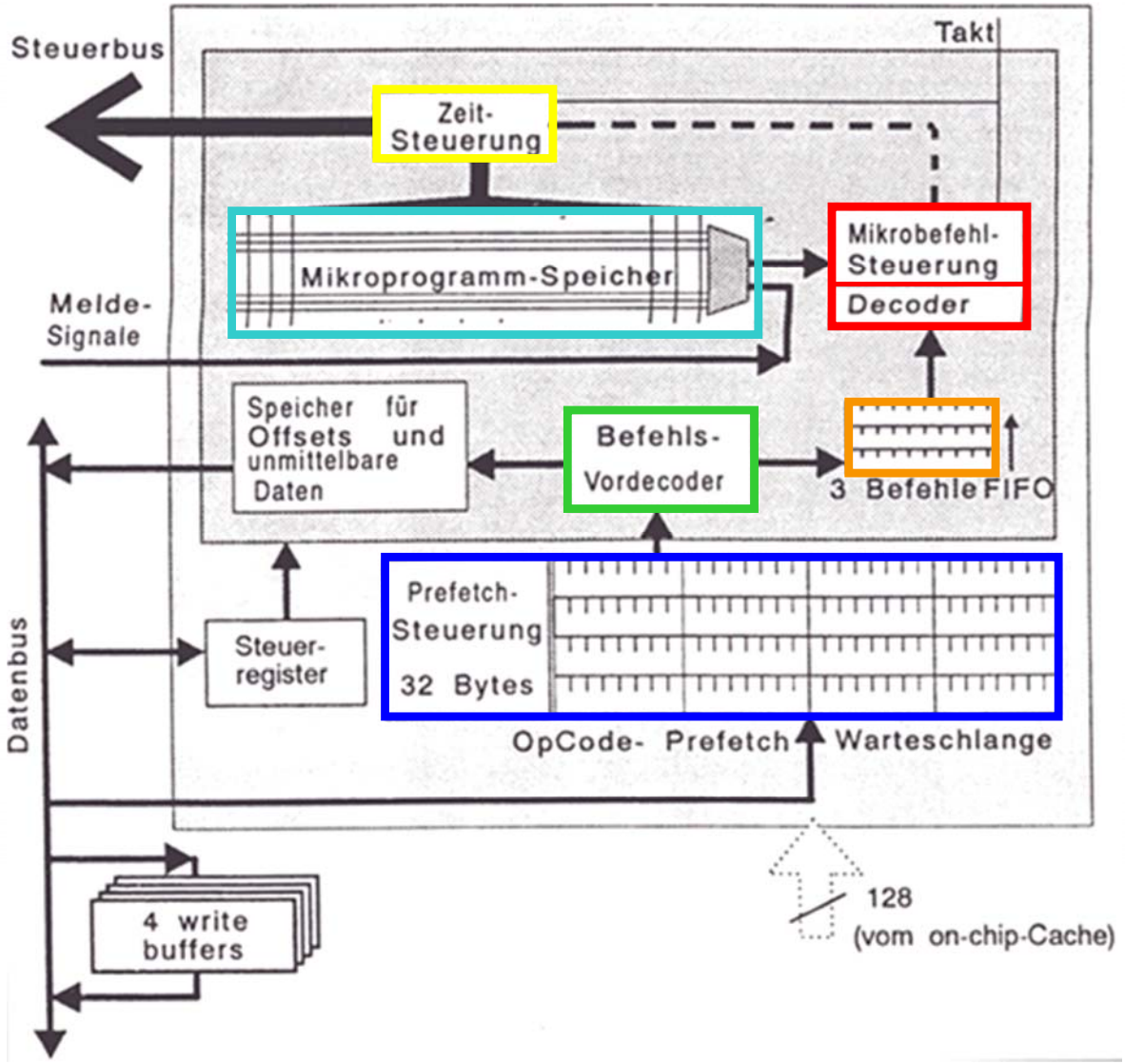
3.2.1 Steuerwerk

Das **Befehlsregister** besteht aus mehreren Registern:

Gründe:

- Variables Befehlsformat:
  - die Befehle sind unterschiedlich lang
    (Befehle sind ein Vielfaches von einem Byte, ...)
- Vorabladen von Befehlen (Opcode-Prefetching):
  - zur Steigerung der Verarbeitungsgeschwindigkeit werden bereits mehrere folgende Befehle in das Befehlsregister geladen, während der aktuelle Befehl gerade dekodiert wird
    - Opcode prefetch queue, Warteschlange, Pipelining

Steuerwerk: Fallstudie (1)



Steuerwerk: Fallstudie (2)

- Prefetch Queue (Opcode Prefetch Warteschlange)
  - FIFO-Speicher mit 32 Byte
  - Prefetch Steuerung sorgt für weitestmögliche Füllung
  - Füllung kann meist aus dem on-chip Cache über einen 128 Bit breiten Datenpfad erfolgen
  - Muss die Füllung jedoch aus dem Arbeitsspeicher erfolgen (Cache Miss), haben diese Zugriffe höchste Priorität
  - Eventuelle gleichzeitig auf den Bus auszugebende Daten werden in Schreibpuffern (write buffers) zwischengespeichert

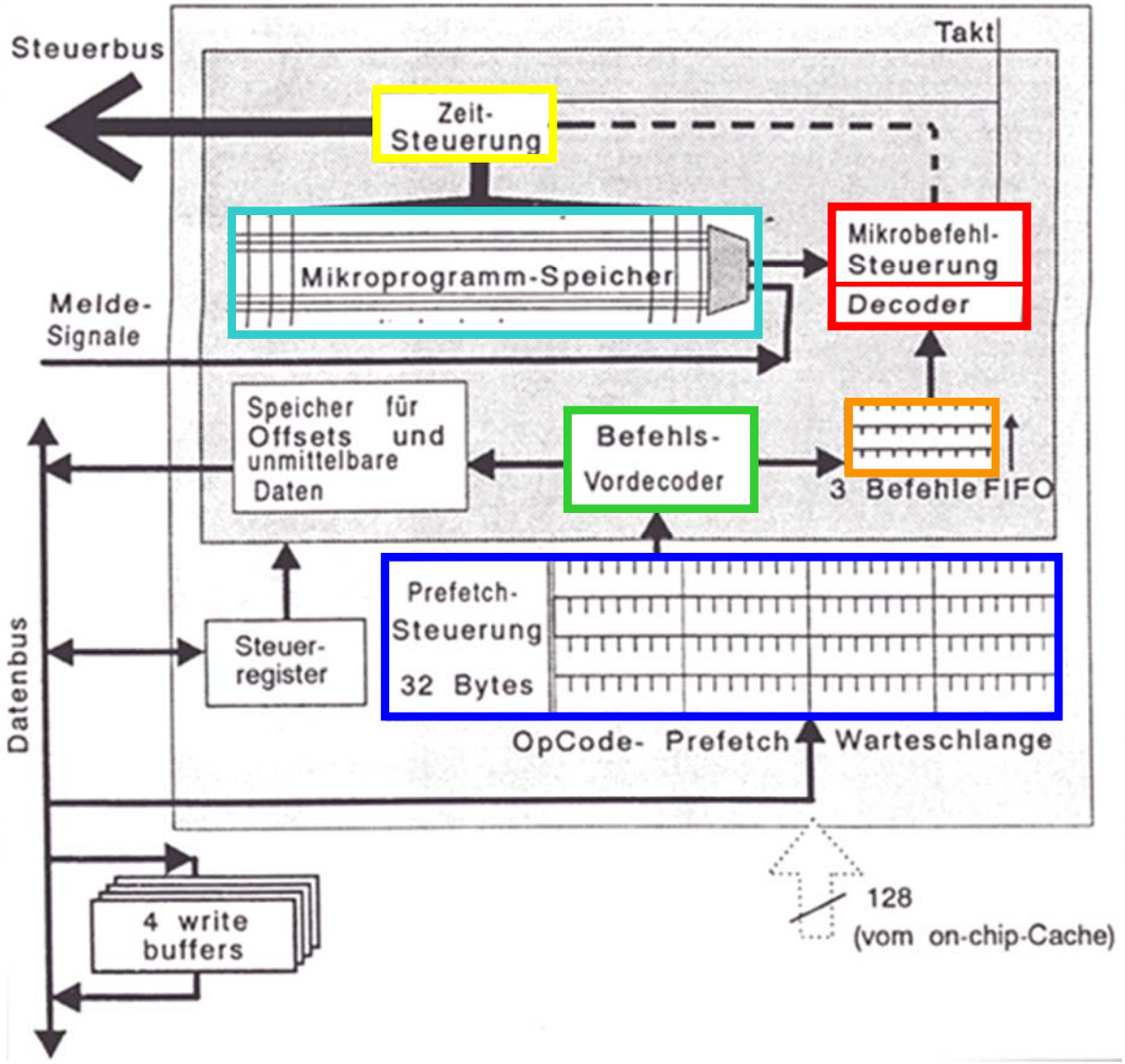
Steuerwerk: Fallstudie (3)

- Befehls-Vordecoder:
  - Aus der Prefetch-Queue gelangen die Befehle in den Befehls-Vordecoder
  - Vorbearbeitung der Befehle
  - Direkt im Befehl angegebene Operanden (unmittelbare Daten) sowie Adressdistanzen (Offsets) werden abgezweigt und separat gespeichert
- Befehls-FIFO:
  - Vordekodierte Befehle gelangen in den Befehls-FIFO
  - Platz für 3 Befehle

Steuerwerk: Fallstudie (4)

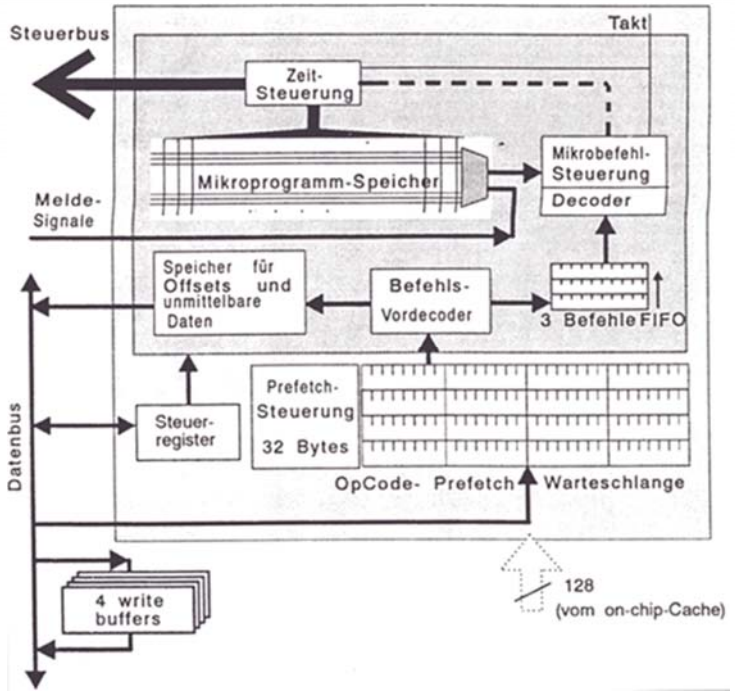
- Befehls-Decoder:
  - Entnimmt den obersten vordecodierten Befehl aus dem Befehls-FIFO
  - Ermittelt die Startadresse des zugehörigen Mikroprogramms
- Mikrobefehls-Steuerung, Mikrobefehls-Speicher:
  - Synchrones mikroprogrammiertes Schaltwerk, erzeugt die Steuersignale, interpretiert die Meldesignale
- Zeit-Steuerung:
  - Synchronisiert die erzeugten Steuersignale mit dem Systemtakt

Steuerwerk: Fallstudie (1)



Mikroprogrammsteuerwerk

- Im Festwertspeicher liegt für jeden Befehl ein Mikroprogramm
  - Mikroprogramm ≡ Folge von Mikrobefehlen
- Einzelne Bits eines Mikrobefehls ≡ Mikrooperationen ≡ Auswahl- und Freigabesignale für die benötigten Komponenten
- Aufbau eines Mikrobefehls:



Folge-Adresse	Register	Rechenwerk		Systembus Schnittstelle	Adresswerk	externe Steuersignale
		ALU	AC,HR			

- Horizontale Mikroprogrammierung: Steuerungsfeld im Mikrobefehl für jedes Steuersignal im Rechner ➔ Kein weiterer Dekoder erforderlich
- Vertikale Mikroprogrammierung: Zusammenfassung von Steuersignalen zu Feldern die erst dekodiert werden müssen