

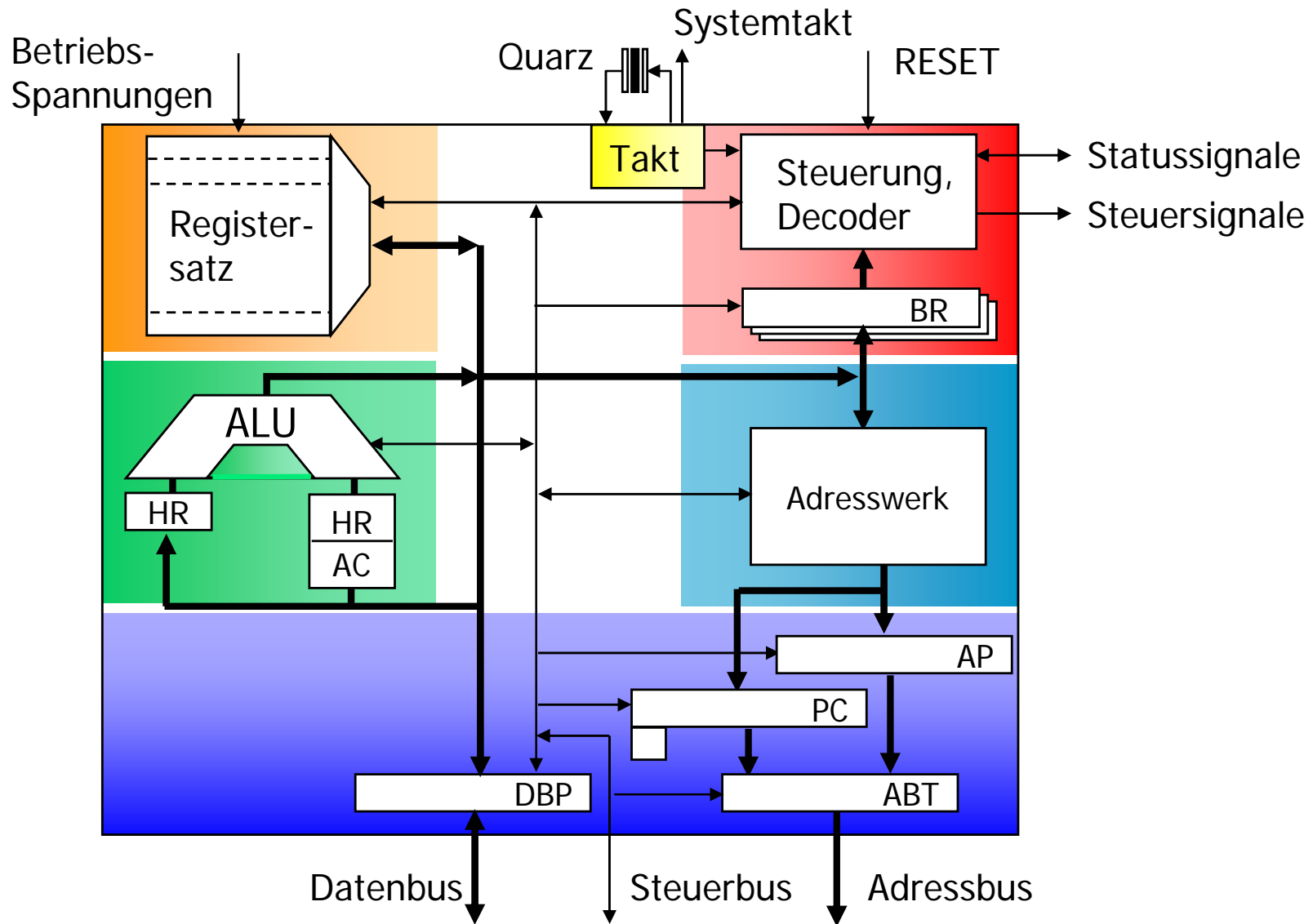
# Aufbau eines einfachen $\mu\text{P}$

---

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



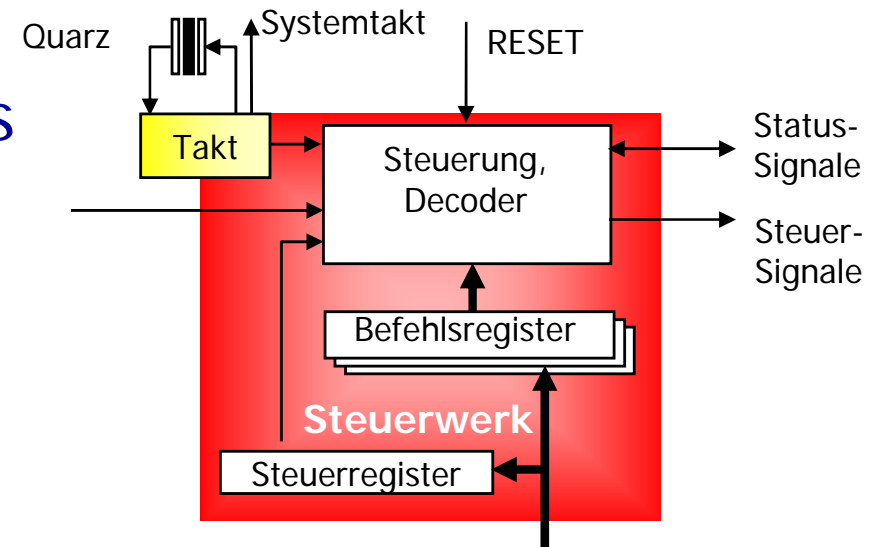
# 3.2 Aufbau eines einfachen $\mu$ P



# Das Steuerregister

Mit Hilfe des **Steuerregisters** kann die aktuelle Arbeitsweise des Steuerwerks beeinflusst werden

Die Bedeutung der Bits des Steuerregisters hängen vom jeweiligen Prozessor ab



Beispiele für die Bedeutung von Bits des Steuerregisters:

- **Interrupt enable Bit**  
bestimmt, ob auf eine Unterbrechungsanforderung am INT-Eingang reagiert wird



# Das Steuerregister

---

- **User/System Bit**

bestimmt ob der Prozessor im User-Modus (nur beschränkter Teil des Befehlsvorrats nutzbar) oder im Systemmodus (alle Befehle verfügbar, i. A. für das Betriebssystem reserviert) arbeitet

- **Trace Bit**

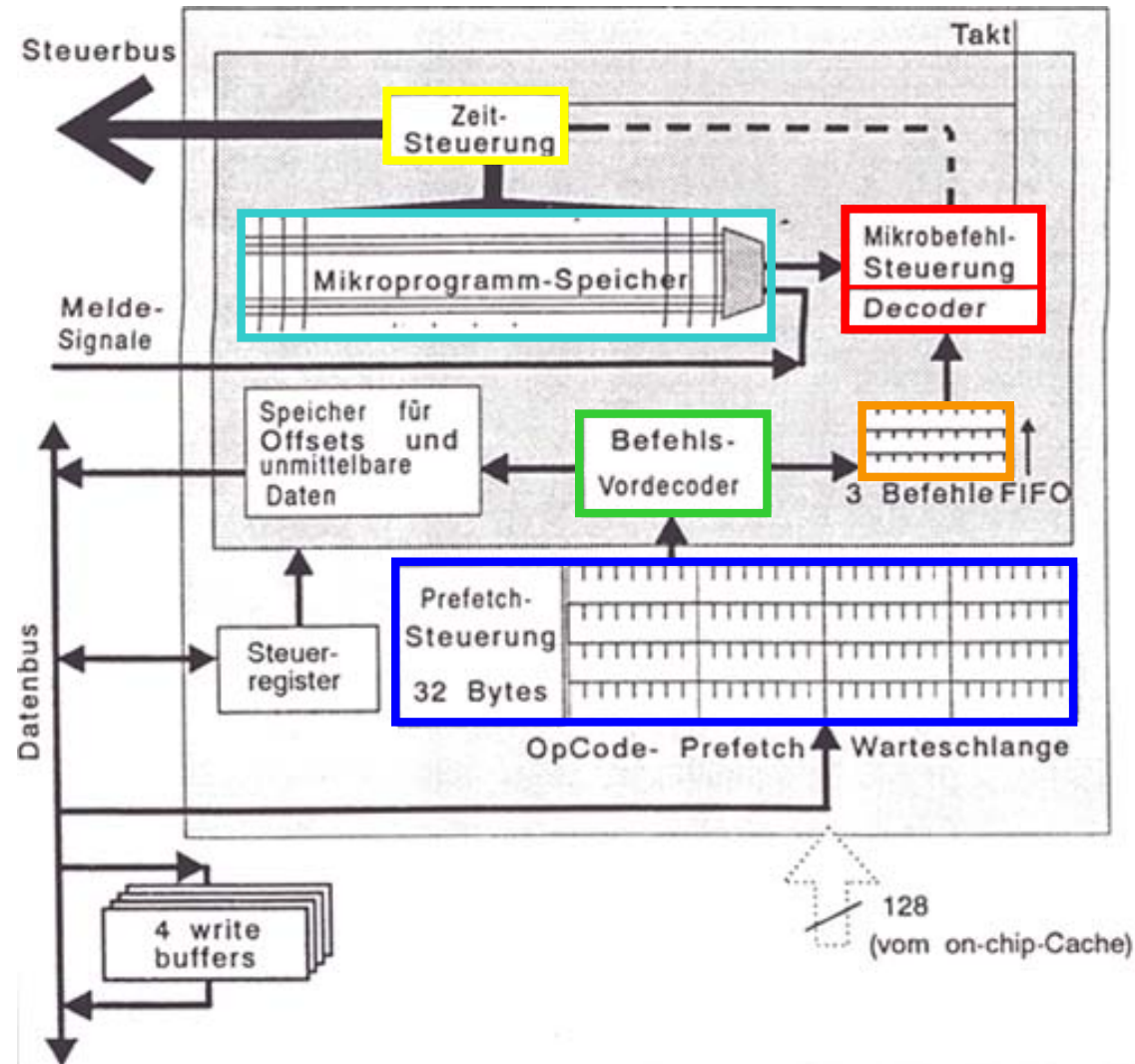
erlaubt Befehlsabarbeitung im Einzelschritt (Single Step Mode), d. h. nach jeder Befehlsausführung wird eine Unterbrechungsroutine gestartet → Debugging

- **Decimal Bit**

entscheidet, ob Dual oder BCD gerechnet wird



# Steuerwerk des Intel 80486



# Steuerwerk des Intel 80486

---

## Befehlsregister-Block: Prefetch-Queue

- 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 des Intel 80486

---

## Befehls-Vordecoder:

- Aus der Prefetch-Queue gelangen die Befehle in den Befehls-Vordecoder
- Vorbereitung der Befehle
- direkt im Befehl angegebene Operanden (unmittelbare Daten) sowie Adressdistanzen (Offsets) werden abgezweigt und separat gespeichert



# Steuerwerk des Intel 80486

---

## Befehls-FIFO:

- vordekodierte Befehle gelangen in den Befehls-FIFO
- Platz für 3 Befehle

## Befehls-Decoder:

- entnimmt den obersten vordecodierten Befehl aus dem Befehls-FIFO
- ermittelt die Startadresse des zugehörigen Mikroprogramms



# Steuerwerk des Intel 80486

---

## **Mikrobefehls-Steuerung, Mikrobefehls-Speicher:**

synchrones mikroprogrammiertes Schaltwerk, erzeugt die Steuersignale, interpretiert die Meldesignale

## **Zeit-Steuerung:**

synchronisiert die erzeugten Steuersignale mit dem Systemtakt



# Steuerwerk: Zusammenfassung

---

- ❑ Das Steuerwerk interpretiert die Maschinenbefehle und setzt sie unter Berücksichtigung der Statusinformation in Steuerkommandos für andere Komponenten um.
- ❑ Das Befehlsregister enthält den als nächstes auszuführenden Maschinenbefehl, sofern der vergangene Befehl kein Sprungbefehl ist.
- ❑ Das Steuerregister beeinflusst die aktuelle Arbeitsweise des Steuerwerks



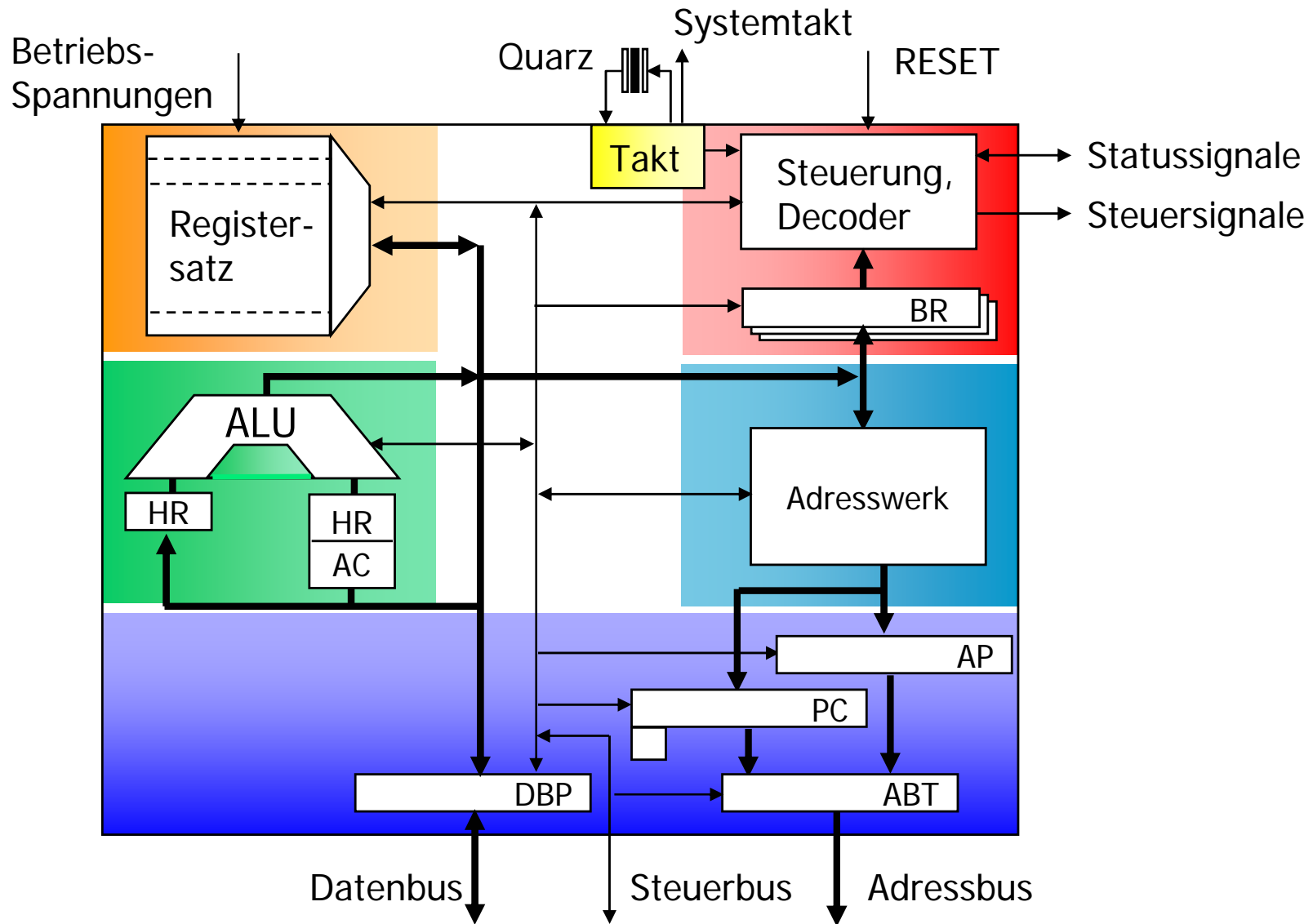
# Aufbau eines einfachen $\mu\text{P}$

---

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

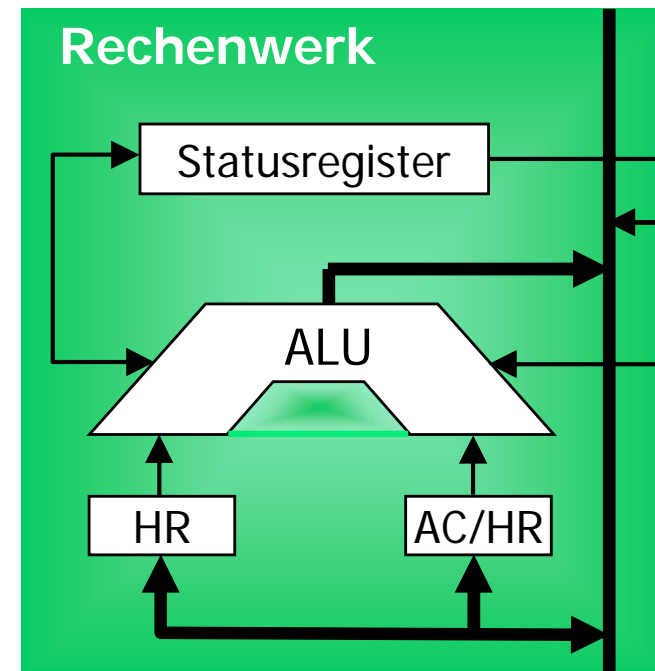


# Aufbau eines einfachen $\mu P$



## 3.2.2 Rechenwerk

- Führt die vom Steuerwerk verlangten logischen und arithmetischen Operationen aus
- **Statusregister** informiert das Steuerwerk über den Ablauf des Ergebnisses (z. B. Carry, Overflow, Zero, Sign, ...)
- Zum Zwischenspeichern von Operanden und Ergebnissen sind **Hilfregister** und **Akkumulatoren** vorhanden



## 3.2.2 Rechenwerk

---

- 2 Eingangsbusse ( 2 Operanden) und 1 Ausgangsbus (Ergebnis). Sie entsprechen in der Breite dem internen Prozessor-Bus und sind mit diesem verbunden
- Die ALU selbst ist ein reines Schaltnetz
- Vor die ALU sind Hilfsregister zur Zwischenspeicherung von Operanden geschaltet
- Die Ergebnisse werden entweder in Prozessor-Registern gespeichert, auf die Eingangsregister der ALU zurückgeführt oder über den externen Datenbus an andere Systemkomponenten übertragen
- Eingänge zur Steuerung der ALU-Operationen



## 3.2.2 Rechenwerk

---

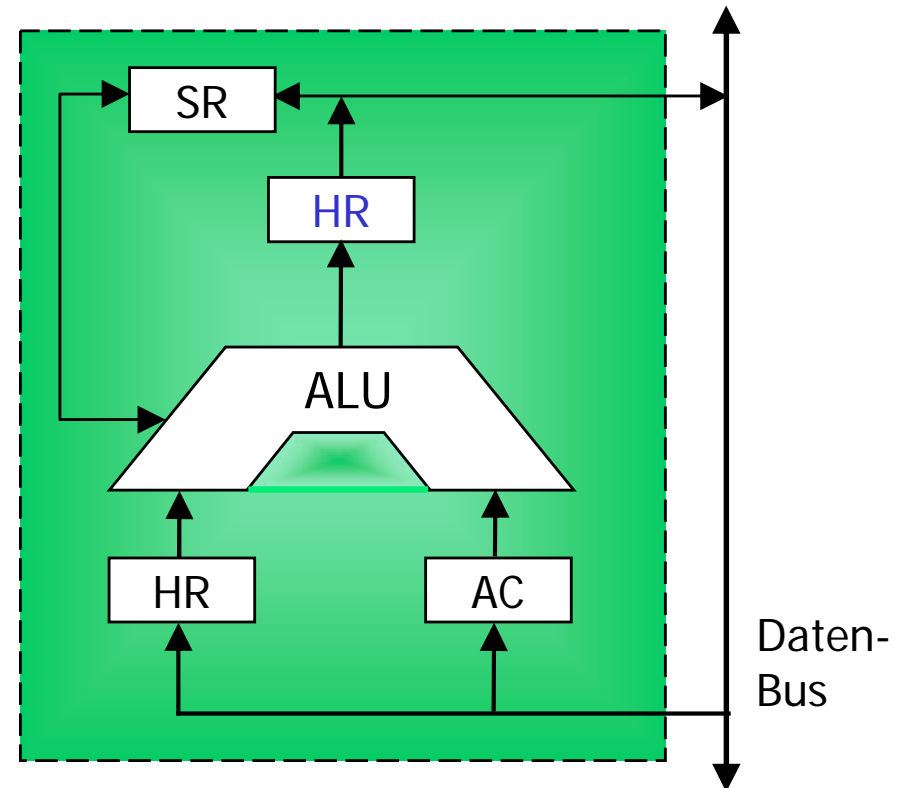
- Melde-Ausgänge für den Status der abgelaufenen Operation (Carry, Overflow, Zero, Sign, ..., usw.)  
Status wird im Status-Register zwischengespeichert
- **Akkumulator (spezielles Register)**  
Bei 8 Bit Mikroprozessoren war das einzige interne Register, das direkt Ergebnisse der ALU aufnehmen kann
  - ➔ alle ALU Ergebnisse werden dort abgelegt
  - ➔ Akkumulator "sammelt" die Ergebnisse auf



# Rechenwerksvarianten

**Variante A:** Hilfsregister des Akkumulators wird hinter die ALU verlegt

**Vorteil:** ALU-Operationen ohne Veränderung des Akkumulators sind möglich

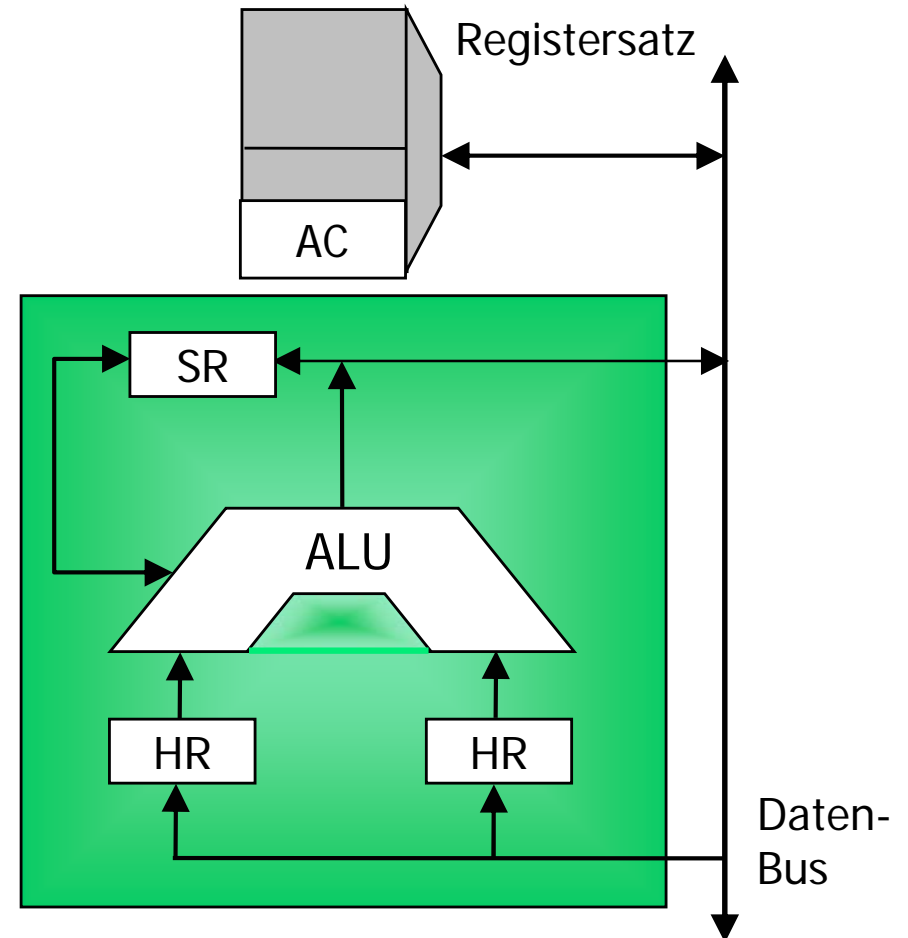


Hauptsächlich in älteren 8-Bit-Prozessoren, bei denen auch die Adressberechnung in der ALU des Rechenwerkes durchgeführt wird

# Rechenwerksvarianten

**Variante B:** Rechenwerk ohne Akkumulator. Der Akkumulator wird in den Registersatz des Prozessors verlegt

**Vorteil:** Mehrere Register des Registersatzes können Akkumulatorfunktion übernehmen  
→ mehrere Akkumulatoren



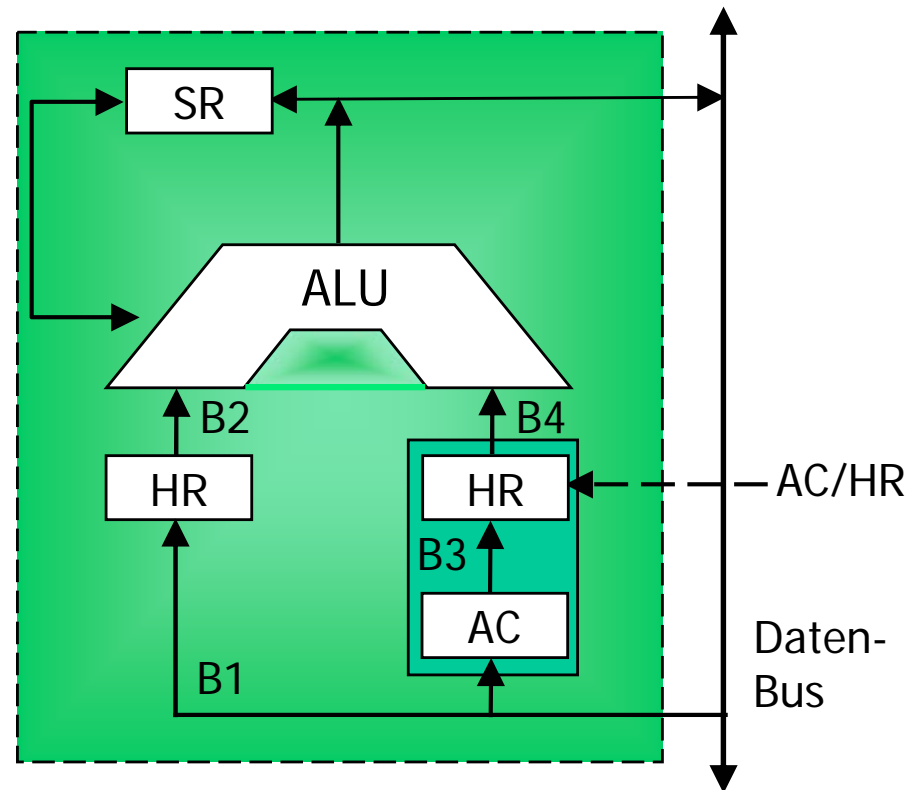
Alle modernen 16/32-Bit-Prozessoren nutzen dieses Konzept



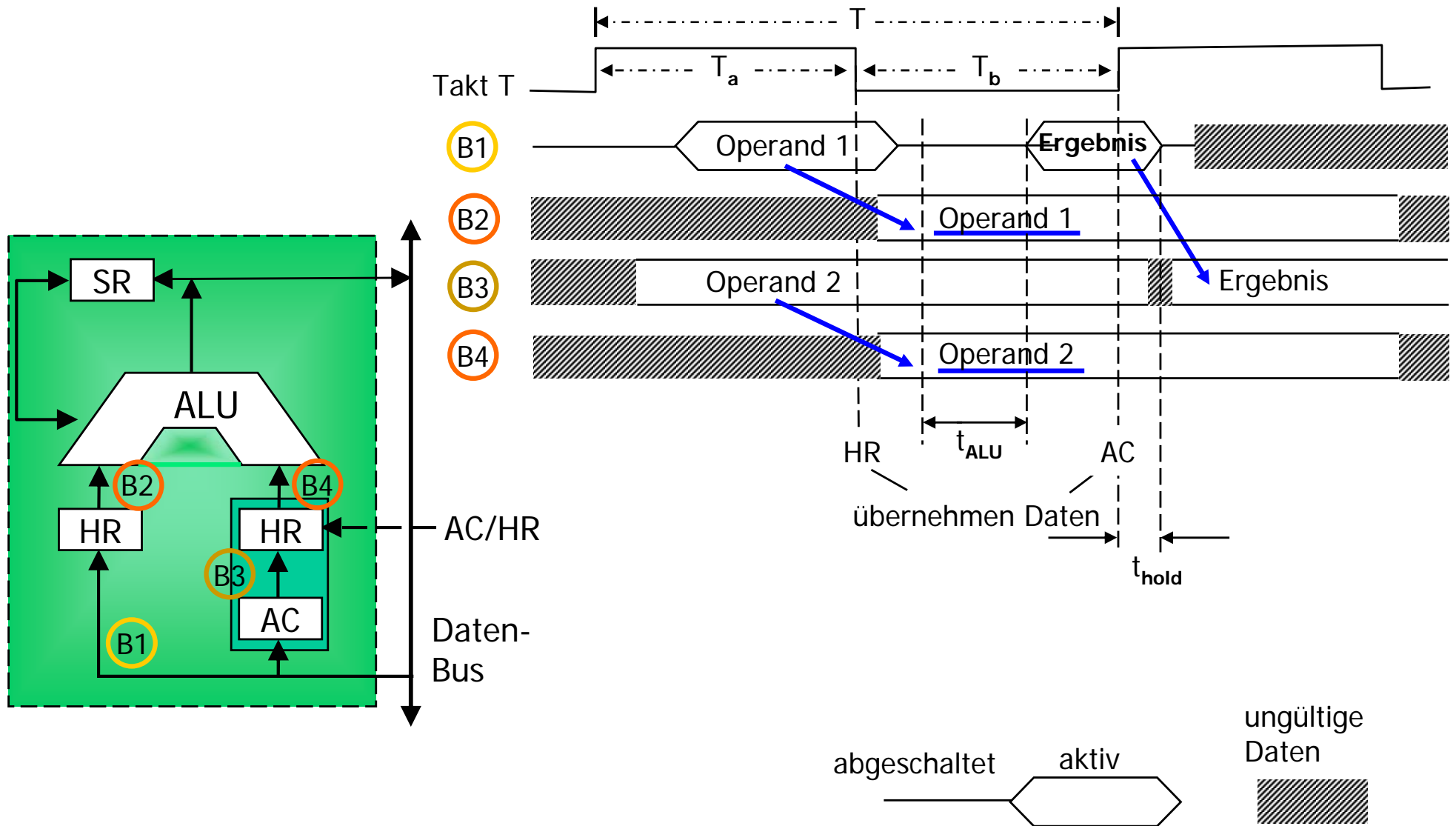
# Aufbau des Rechenwerks

## Akkumulator

meist zweigeteilt, bestehend aus dem eigentlichen Akkumulator-Register **AC** und einem nachgeschalteten Hilfsregister **HR** (Latch)



# Zeitverhalten des Rechenwerks



# Zeitverhalten des Rechenwerks

---

- Während der positiven Takthälfte von  $T$  liegen auf  $B_1$  und  $B_3$  die Operanden
- Diese werden mit der negativen Taktflanke in die Hilfsregister übernommen
- Danach stehen diese auf  $B_2$  und  $B_4$  stabil zur Verfügung, die ALU berechnet in einer Ausführungszeit  $t_{ALU}$  das Ergebnis
- Dieses Ergebnis wird mit der positiven Taktflanke in den Akkumulator übernommen
- Ohne Hilfsregister würden sich die während der ALU-Rechenzeit ergebenden Schwankungen am Ausgang (Hasards, Wettläufe) sofort wieder auf den ALU-Eingang auswirken → ein asynchrones Schaltwerk mit allen bekannten Problemen und Fehl-Ergebnissen würde entstehen.



# Operationsvorrat der ALU

---

## □ Arithmetische Operationen:

- Addieren ohne/mit Übertrag
- Subtrahieren ohne/mit Übertrag
- Inkrementieren/Dekrementieren
- Multiplizieren ohne/mit Vorzeichen
- Dividieren ohne/mit Vorzeichen
- Komplementieren (Zweierkomplement)

## □ Logische bitweise Verknüpfungen:

- Negation
- UND
- ODER
- Antivalenz



# Operationsvorrat der ALU

---

## □ Schiebe- und Rotations-Operationen:

- Links-Verschieben
- Rechts-Verschieben
- Links-Rotieren ohne Übertragsbit
- Links-Rotieren durchs Übertragsbit
- Rechts-Rotieren ohne Übertragsbit
- Rechts-Rotieren durchs Übertragsbit

## □ Transport-Operationen:

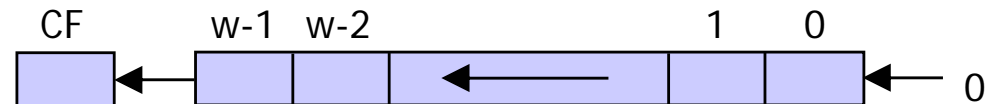
- Transferieren



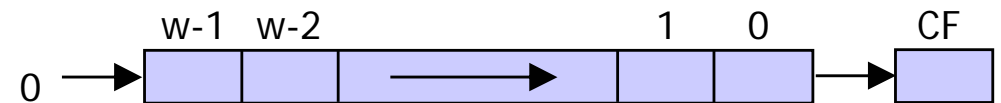
# Schiebeoperationen

- ❑ Dem logischen Linksschieben entspricht die Multiplikation mit 2
- ❑ Dem logischen Rechtschieben entspricht die Division durch 2
- ❑ Dies gilt jedoch nur für positive Zahlen. Bei negativen Zahlen im Zweierkomplement muß zur Vorzeichenerhaltung das höchstwertigste Bit in sich selbst zurückgeführt werden.

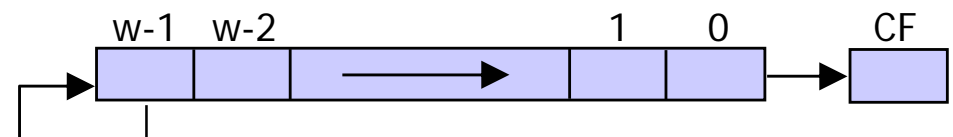
a) logisches und arithmetisches Verschieben nach links



logisches Verschieben nach rechts



b) arithmetisches Verschieben nach rechts



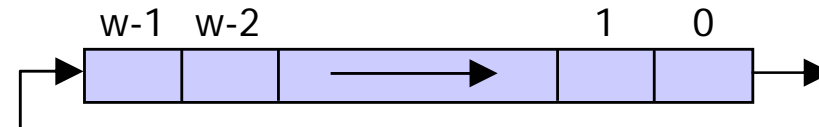
➔ Unterschied logisches/arithmetisches Rechtsschieben



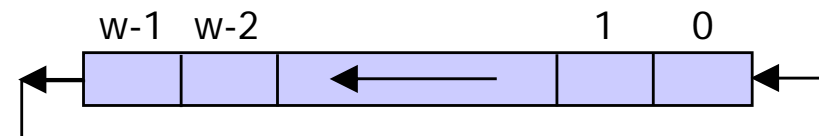
# Rotationsoperationen

- ❑ Register wird als geschlossene Bitkette betrachtet
- ❑ Carry-Flag kann wahlweise mitbenutzt oder als zusätzliches Bit einbezogen werden

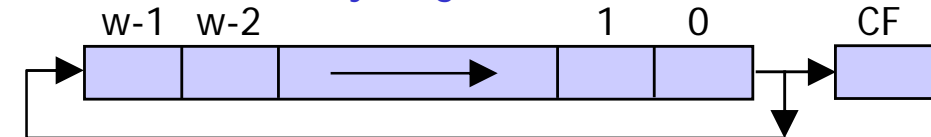
Rotieren nach rechts



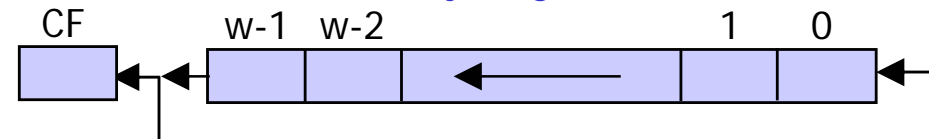
Rotieren nach links



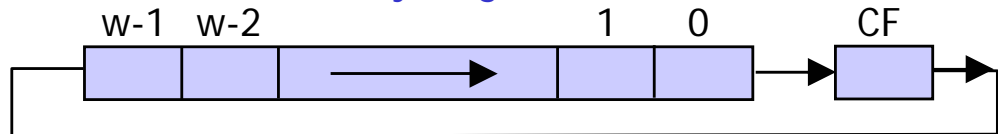
Rotieren nach rechts mit Carry-Flag



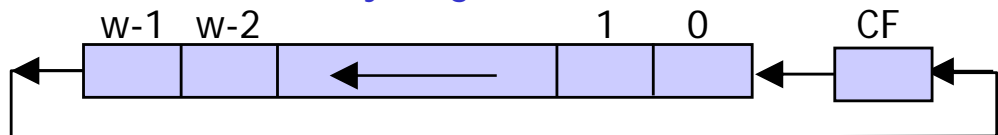
Rotieren nach links mit Carry-Flag



Rotieren nach rechts durch Carry-Flag

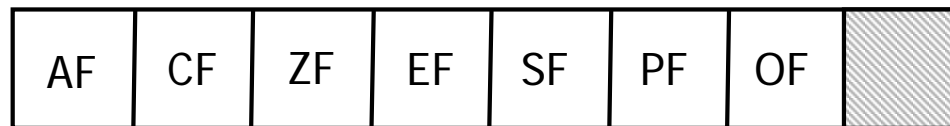


Rotieren nach links durch Carry-Flag



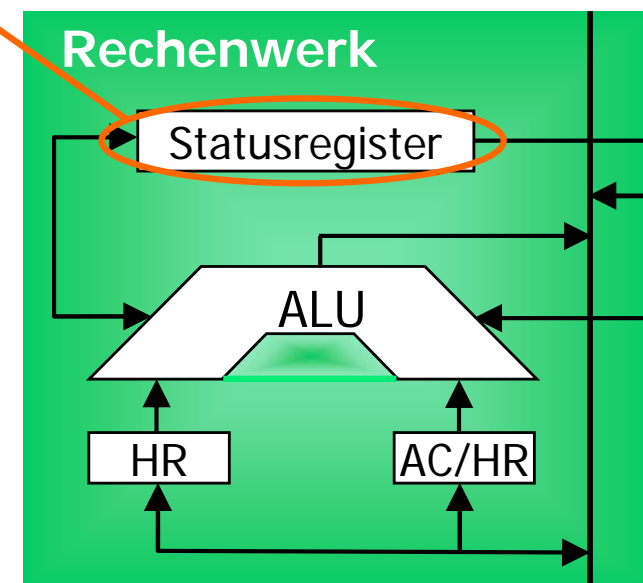
# Statusregister (Zustandsregister, Condition Code Register CCR)

Einzelne Bits, die das Ergebnis einer arithmetischen Operation widerspiegeln werden im Statusregister gespeichert



Das Statusregister enthält meist folgende Bits (Flags):

- Übertragsbit (Carry Flag CF)
- Hilfsübertragsbit (Auxiliary Carry AF)
- Nullbit (Zero Flag ZF)
- Vorzeichenbit (Sign Flag SF)
- Überlaufbit (Overflow Flag OF)
- Even Flag EF
- Paritätsbit (Parity Flag PF)



# Bedeutung der Statusflags

- **Carry Flag (CF):** Übertrag aus dem höchstwertigsten Bit bei Addition oder Subtraktion (Borrow) → sequentielle Addition und Subtraktion mit größerer Wortbreite ist möglich.
- **Aux Carry (AF):** Übertrag von Bit 3 in Bit 4 des Ergebnisses (BCD-Arithmetik).
- **Zero Flag (ZF):** Zeigt an, ob das Ergebnis der letzten Operation gleich 0 war (bedingte Programmverzweigungen, Zählschleifen)
- **Sign Flag (SF):** Zeigt an, dass das Ergebnis negativ ist (Most Significant Bit = 1). Spiegelt das höchstwertige Bit eines Operanden wieder (Bedingte Programmverzweigungen )
- **Overflow Flag:** Bereichsüberschreitung im Zweierkomplement
- **Even Flag (EV):** zeigt an, ob das Ergebnis eine gerade Zahl ist
- **Parity Flag (PF):** Signalisiert ungerade Parität des Ergebnisses,



# Statusregister (Zustandsregister, Condition Code Register CCR)

---

Werte von Statusbits können direkt Einfluß auf die Ausführung des Mikroprogramms haben

→ bedingte Programmverzweigung

Statusregister und Steuerregister werden häufig zur Erleichterung der Adressierung zusammengefaßt betrachten und manipuliert, und meist **Prozessorstatuswort (PSW)** genannt



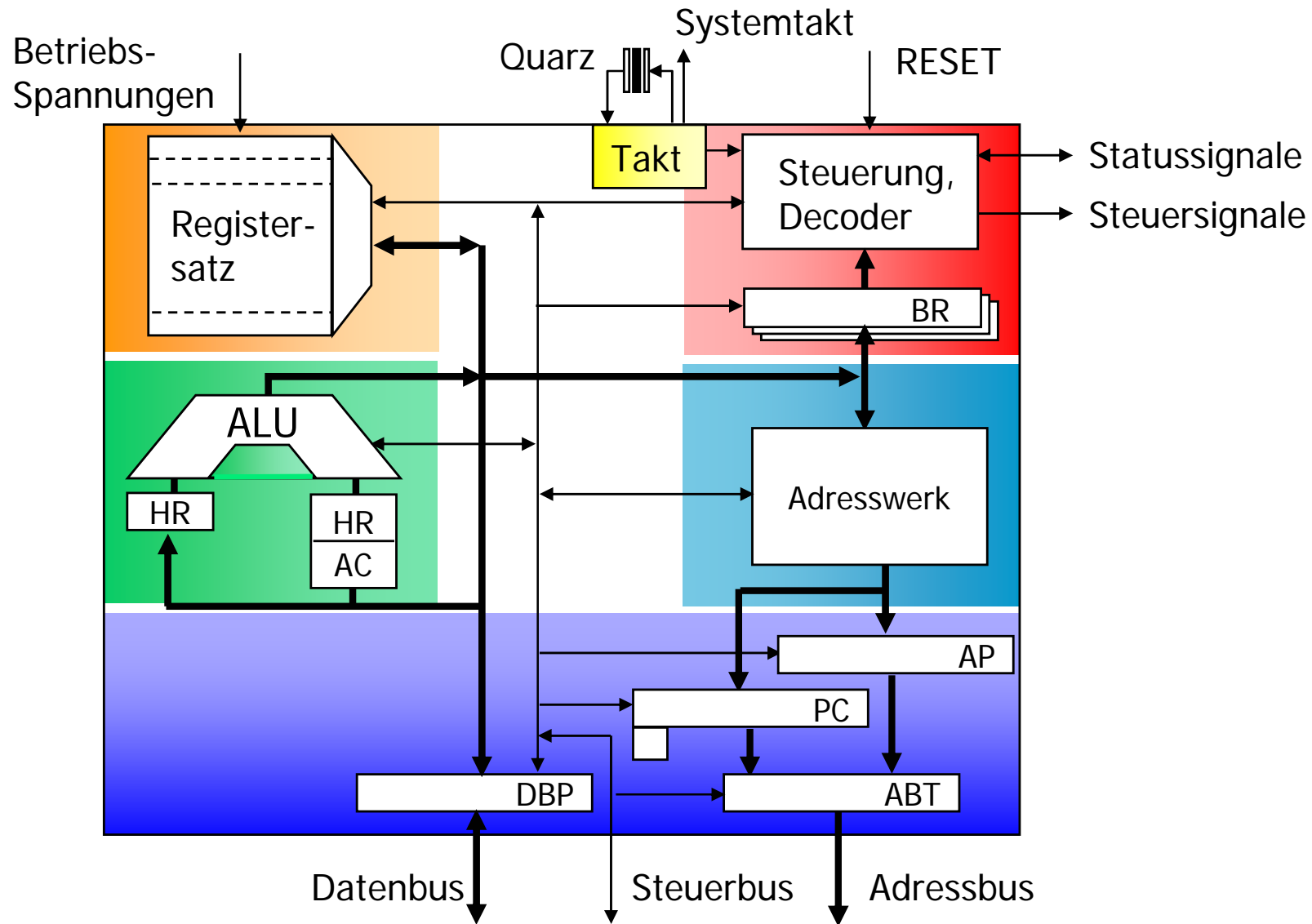
# Aufbau eines einfachen $\mu\text{P}$

---

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



# Aufbau eines einfachen $\mu$ P

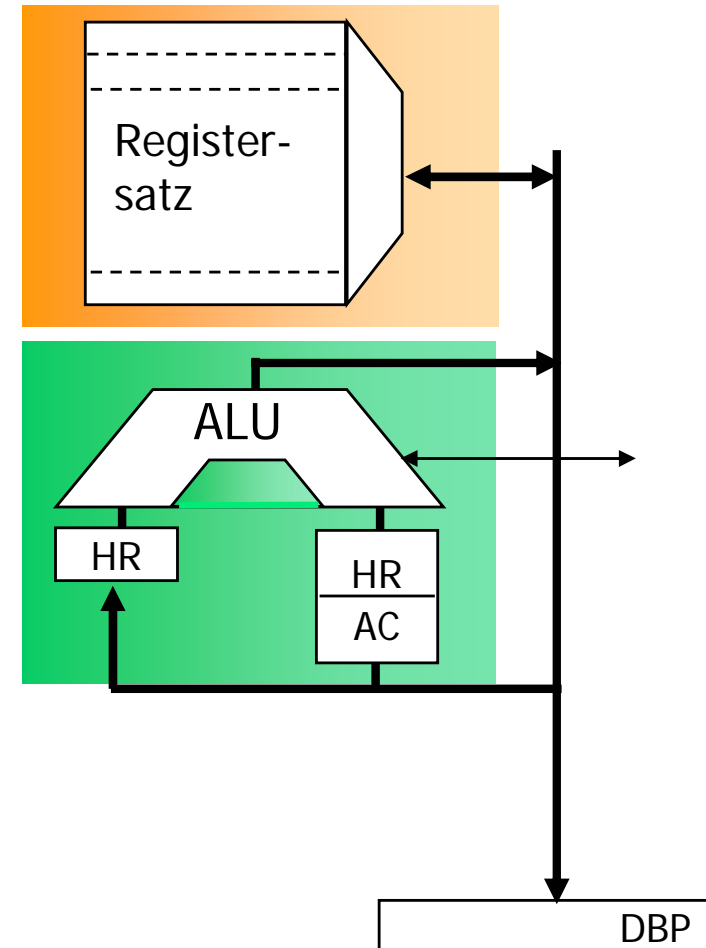


## 3.2.3 Registersatz

Erweiterung des Rechenwerks:

Häufig benutzte Operanden können dort zwischengespeichert werden

→ schnellerer Zugriff als auf den Hauptspeicher



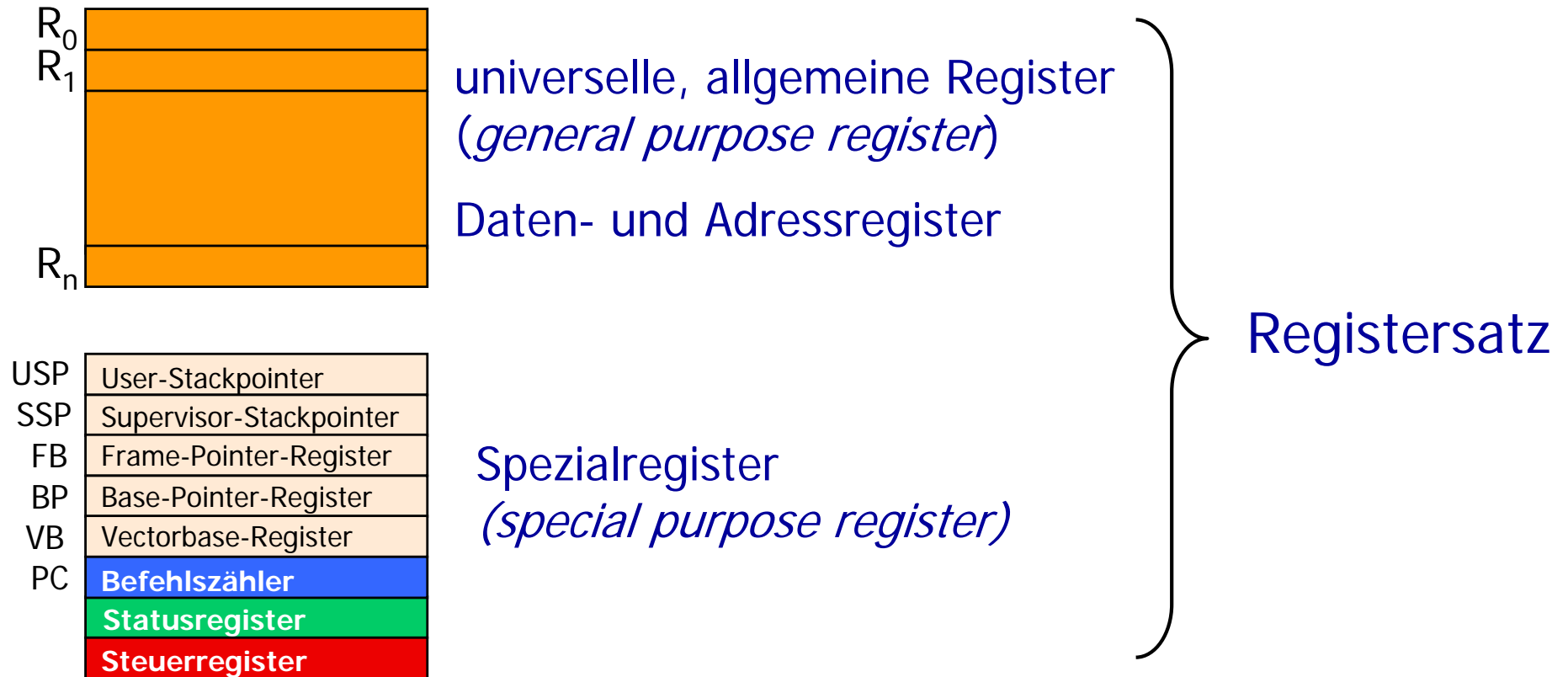
## 3.2.3 Registersatz

---

- Getrennte Ein-/Ausgänge → Dual Port Speicher, zwischen Eingangs- und Ausgangsbuss gehängt
  - Schreiben eines Registers und gleichzeitiges Lesen eines anderen Registers möglich
- Heutige superskalar-Prozessoren: pro Takt mehrere allgemeine Register schreiben und lesen
- Oft dynamische Speicherzellen → Refresh erforderlich
- Register mit Zusatzfunktionen:  
Inkrementieren/Dekrementieren/auf Null setzen/Inhalt verschieben



# 3.2.3 Registersatz



# Daten- und Adressregister

---

## Datenregister:

- Zwischenspeichern von Operanden
- schneller Zugriff auf häufig benutzte Operanden
- bei modernen Prozessoren sind mehrere Datenregister als Akkumulator nutzbar

## Adressregister:

Speichern von Adressen (oder Teile davon) eines Operanden im Hauptspeicher

- Basisregister
- Indexregister



# Spezialregister (special purpose register)

---

Register zur speziellen Verwendung:

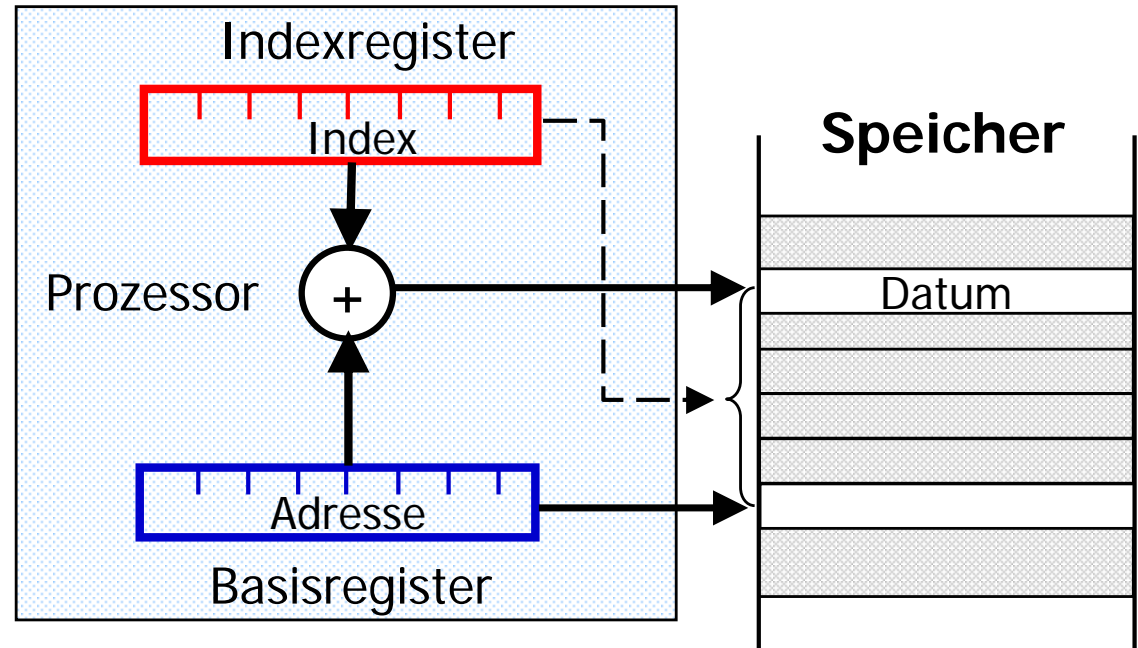
- ❑ Befehlszähler (instruction pointer)
- ❑ Steuerregister
- ❑ Statusregister
- ❑ Register für den Start von Interrupt-Behandlungen (interrupt vector base register)
  
- ❑ **Stackregister** (user und supervisor Stackpointer)



# Funktion von Basis- und Indexregister

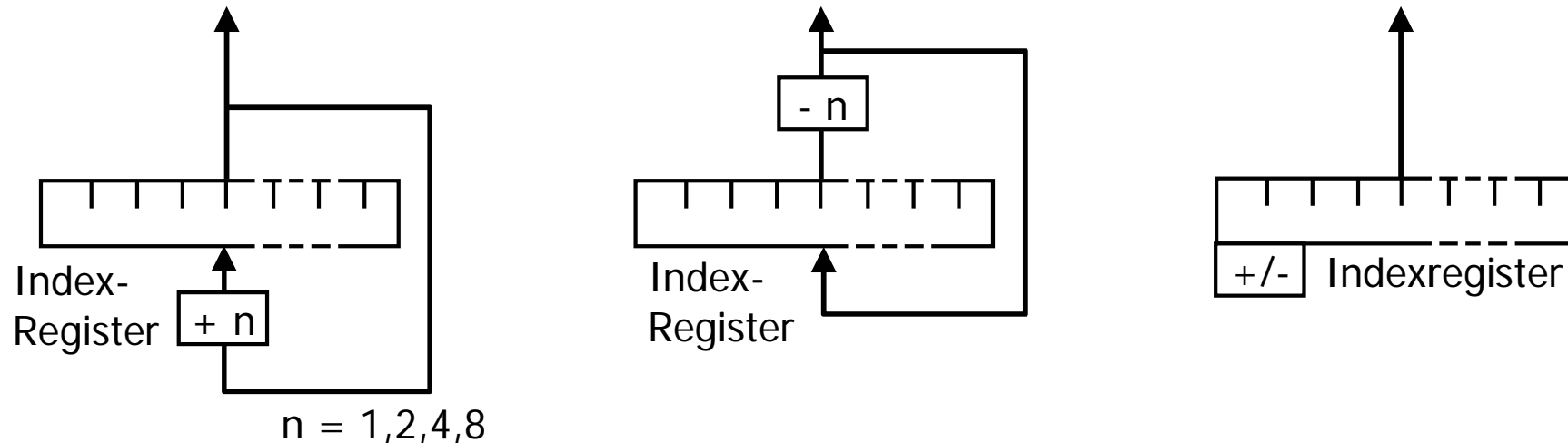
Basisregister enthält die Anfangsadresse eines Speicherbereichs.

Diese bleibt während der Bearbeitung des Speicherbereichs unverändert.



Indexregister enthält eine Distanz (Offset, Displacement) zu einer Basisadresse und dient zur Auswahl eines bestimmten Datums des Speicherbereichs.

# Automatische Modifikation von Indexregistern



## a) Post-Inkrement:

automatische Erhöhung des Registerwerts um  $+n$  nach Adressierung einer Speicherzelle

## b) Pre-Dekrement:

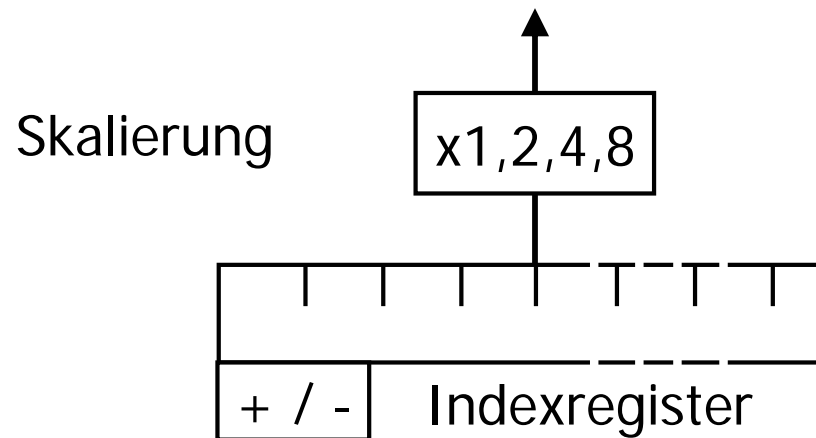
automatische Erniedrigung des Registerwerts um  $-n$  vor Adressierung einer Speicherzelle

## c) Auto-Inkrement / Auto-Dekrement Register



# Register mit Skalierung

Das Indexregister wird vor der Auswertung je nach aktueller Datenlänge (1 Byte, 2 Byte, 4 Byte, 8 Byte) mit dem Faktor 1, 2, 4 oder 8 multipliziert.



## Vorteil:

bessere Ausnutzung der Registerbreite, da das Register selbst nur noch um 1 inkrementiert bzw. dekrementiert werden muss.



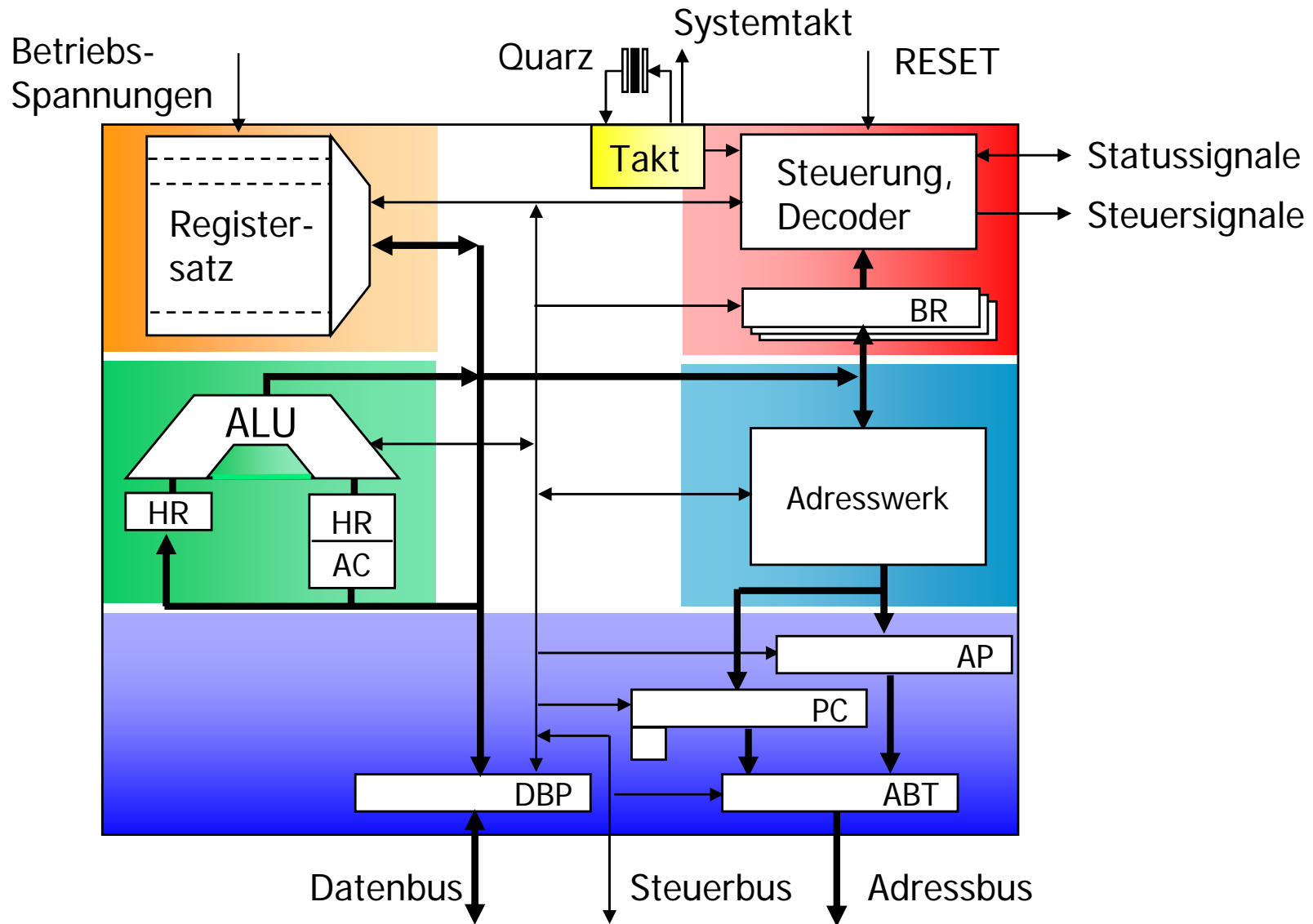
# Aufbau eines einfachen $\mu\text{P}$

---

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



# Aufbau eines einfachen $\mu P$



## 3.2.4 Adresswerk

- Berechnet nach den Vorschriften des Steuerwerks die Adresse eines Befehls oder eines Operanden
- Früher:  
Häufig Bestandteil des Rechenwerks
- Heute:  
Sehr komplex (viele verschiedene komplexe Adressierungsarten) und deshalb eigenständig

