Prozessorentwurf

- Einführung:
  - Leistungsverhalten eines Prozessors wird bestimmt durch:
    - Befehlszahl
    - Taktzykluszeit
    - Taktzyklen/Befehl
  - Compiler und die Befehlssatzarchitektur verantwortlich für die Befehlszahl, die ein bestimmtes Programm benötigt.
  - Implementieren des Prozessors verantwortlich für Taktzykluszeit und Taktzyklen/Befehl

Prozessorentwurf (2)

- Einführung:
  - im weiteren wird der Entwurf des
    - **Datenpfades**
    - **Steuerwerkes**

    eingeführt.

Schaltungstechnische Grundlagen

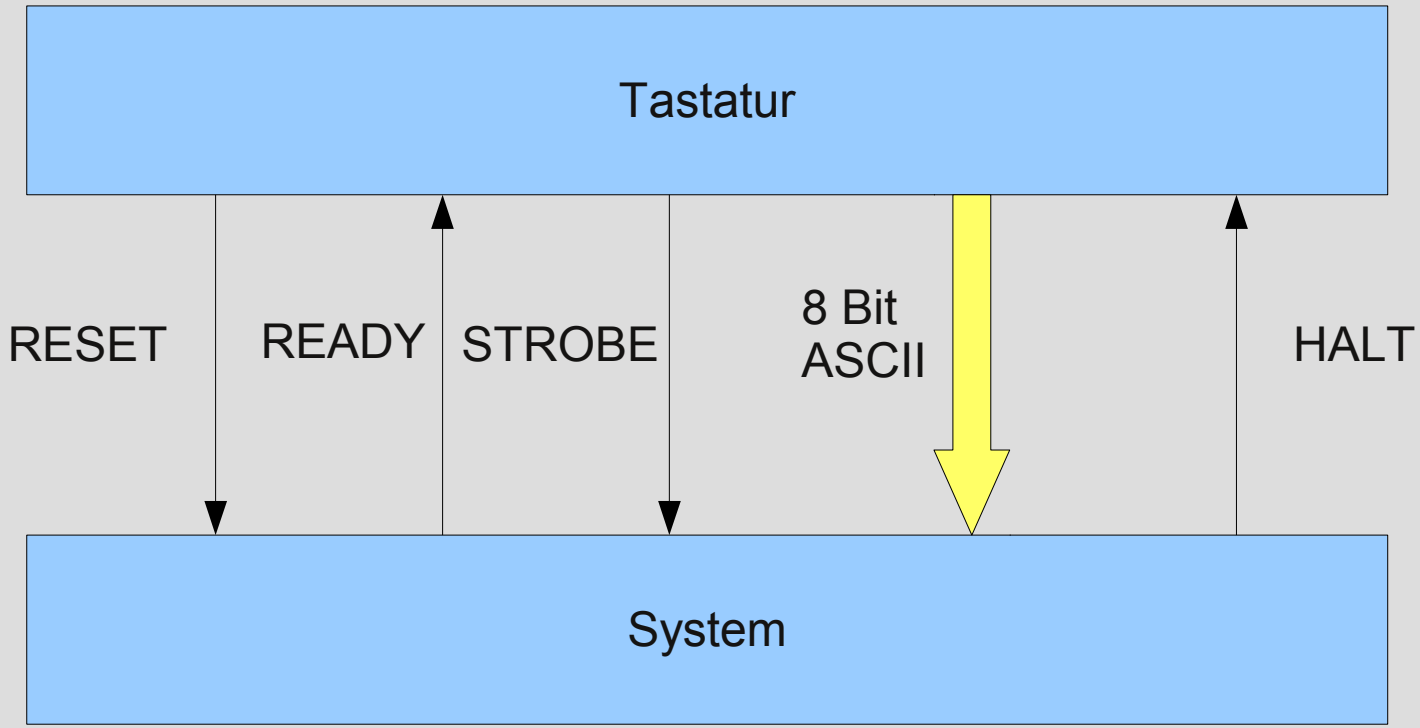
- Aufgabenstellung:
  - Eingabe einer Integerdezimalzahl (16-Bit) über die Tastatur
  - Umwandlung in die interne Darstellung als Dualzahl

Schaltungstechnische Grundlagen (2)

- Teilaufgaben
  - „Implemetierung“ der Tastatur
  - „Ablegen“ der Dualzahl in den internen Speicher des Systems

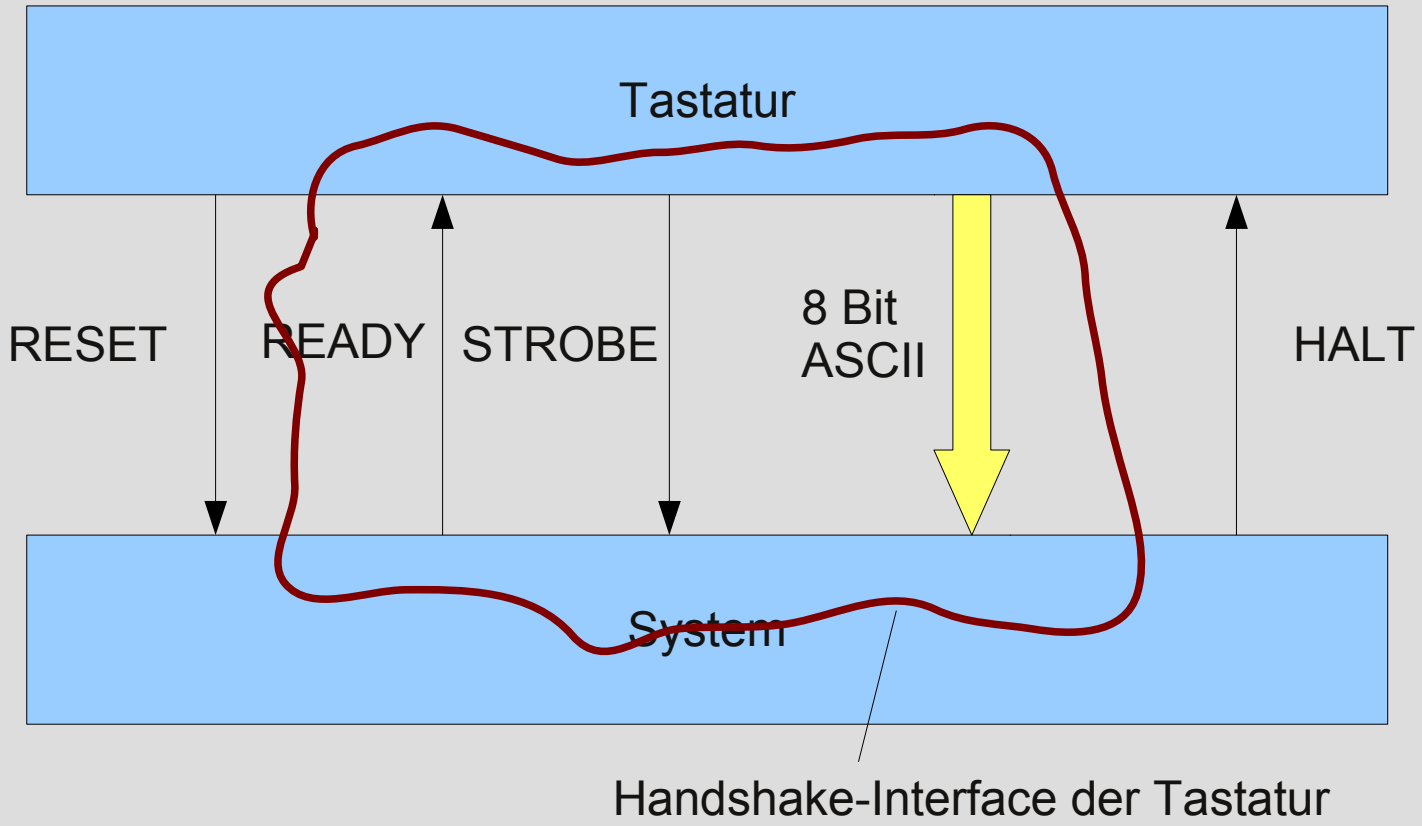
Schaltungstechnische Grundlagen (3)

- a) Schnittstellen



Schaltungstechnische Grundlagen (3)

- a) Schnittstellen



Schaltungstechnische Grundlagen (4)

- b) Ablaufgraph:
  - Siehe Vorlesung

Schaltungstechnische Grundlagen (5)

- b) Ablaufgraph des Handshake-Interfaces:
  - READY = 1: System ist bereit, Daten von der Tastatur zu übernehmen
  - READY = 0: System ist **nicht** bereit, Daten von der Tastatur zu übernehmen
  - STROBE = 1: Die von der Tastatur auf den Datenbus gelegten Daten sind gültig
  - STROBE = 0: Die von der Tastatur auf den Datenbus gelegten Daten sind **un**gültig

Schaltungstechnische Grundlagen (6)

- b) Ablaufgraph des Handshake-Interfaces:
  Signalspiel

  - Siehe Vorlesung

Kodierung der Daten

- Syntax der Eingabedaten:
  integerzahl_im_ascii_format
  ::=[vorzeichen]ascii_ziffer[ascii_ziffer[ascii_ziffer[ascii_ziffer[
  ascii_ziffer]]]]<cr<lf>

Kodierung der Daten

- Syntax der Eingabedaten:
  integerzahl_im_ascii_format
  ::=[vorzeichen]ascii_ziffer[ascii_ziffer[ascii_ziffer[ascii_ziffer[
  ascii_ziffer]]]]<cr><lf>

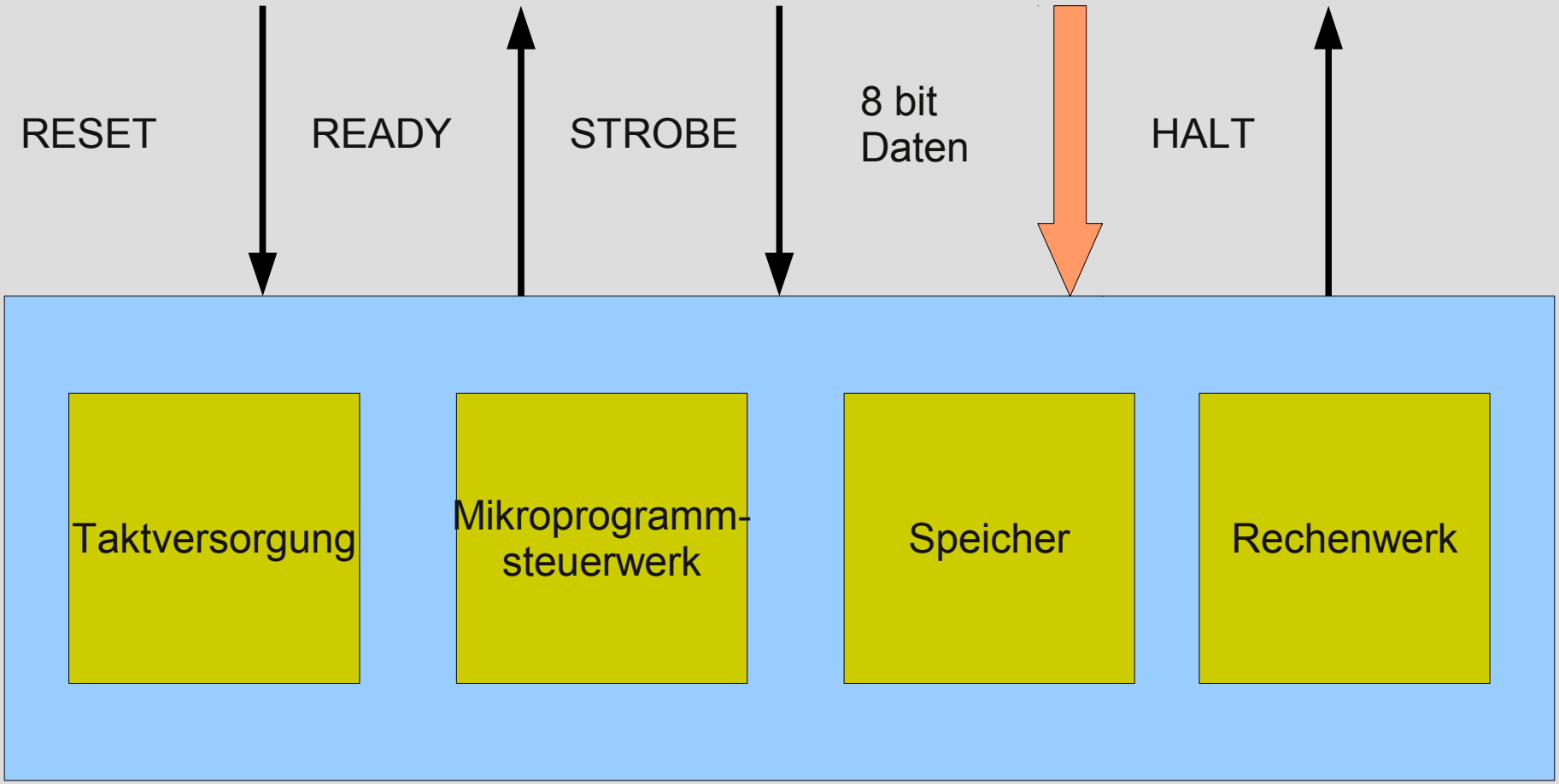
  vorzeichen ::= +|- , fehl vorzeichen, wird '+'
  angenommen
  ascii_ziffer ::= 0|1|2|3|4|5|6|7|8|9

Kodierung der Daten

- Kodierung:

+	2Bh	6	36h
-	2Dh	7	37h
0	30h	8	38h
1	31h	9	39h
2	32h	<cr>	0Dh
3	33h	<lf>	0Ah
4	34h		
5	35h		

Grobentwurf des Systems

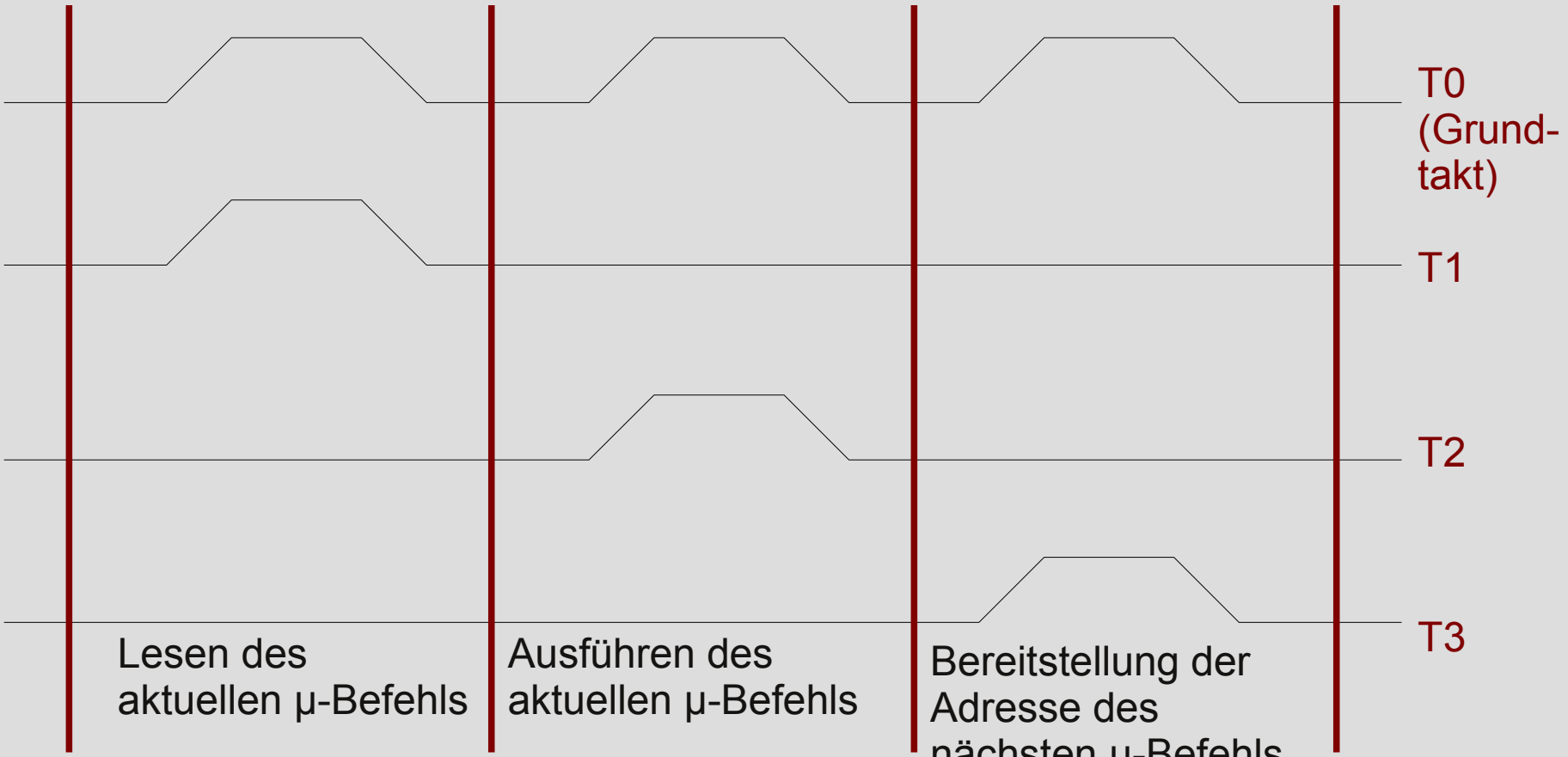


Feinentwurf des Systems

- Taktversorgung
  - Festlegung des Taktschemas ist immer der erste Schritt
  - Zur Realisierung der Taktversorgung sollen Master-Slave-FlipFlops verwendet werden.

Feinentwurf des Systems

- Taktversorgung



Zentrale Steuerschleife

Feinentwurf des Systems

- Taktversorgung
  - Weiterhin soll gelten:
    - (RESET=1) v (HALT=1) ⇒ T1=T2=T3=0
    - Der erste Takt T0 nach (RESET=0) ^ (HALT=0) führt zu T1

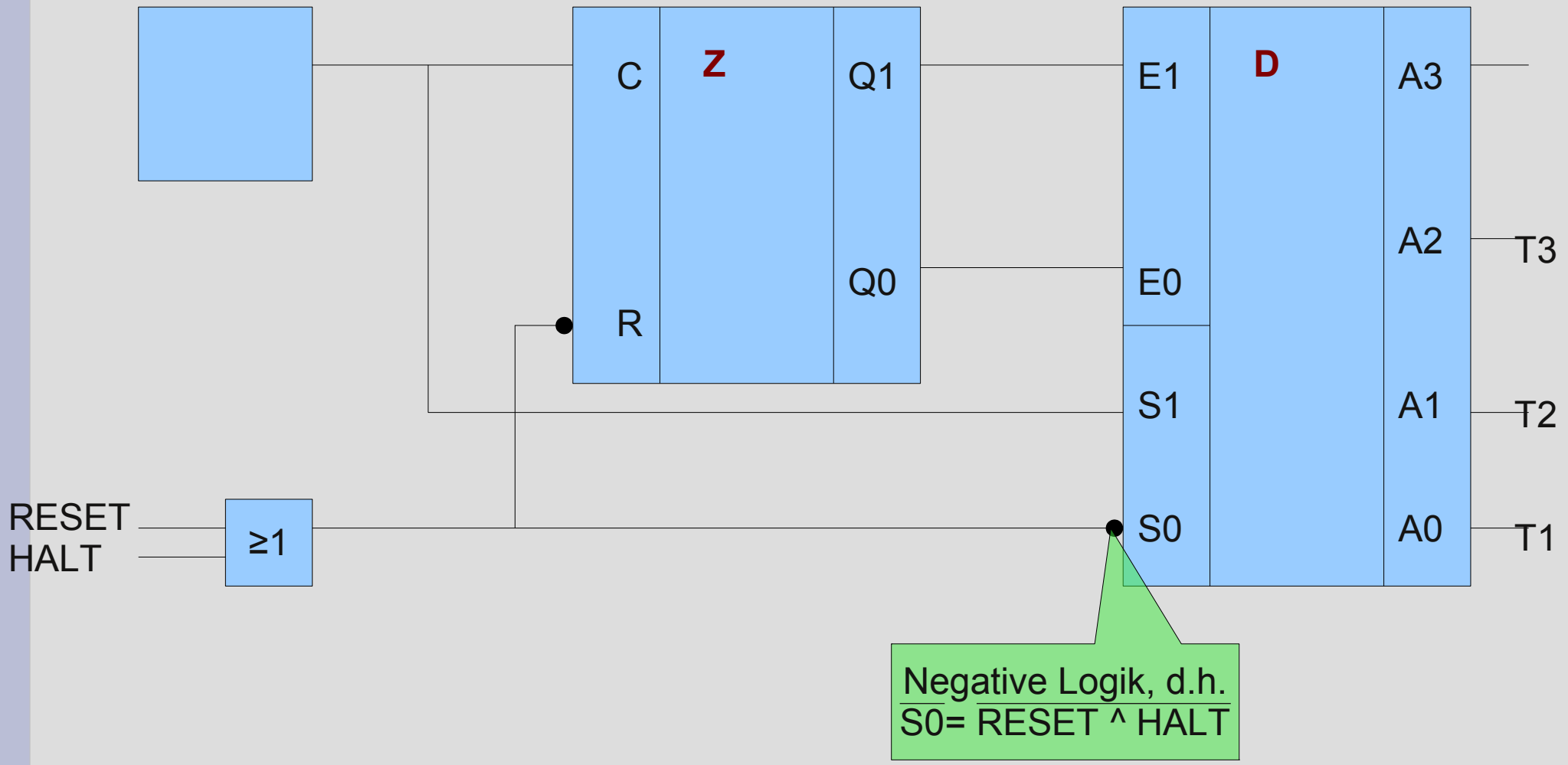
Feinentwurf des Systems

- Taktversorgung
  - Weiterhin soll gelten:
    - (RESET=1) v (HALT=1) ⇒ T1=T2=T3=0
    - Der erste Takt T0 nach (RESET=0) ^ (HALT=0) führt zu T1

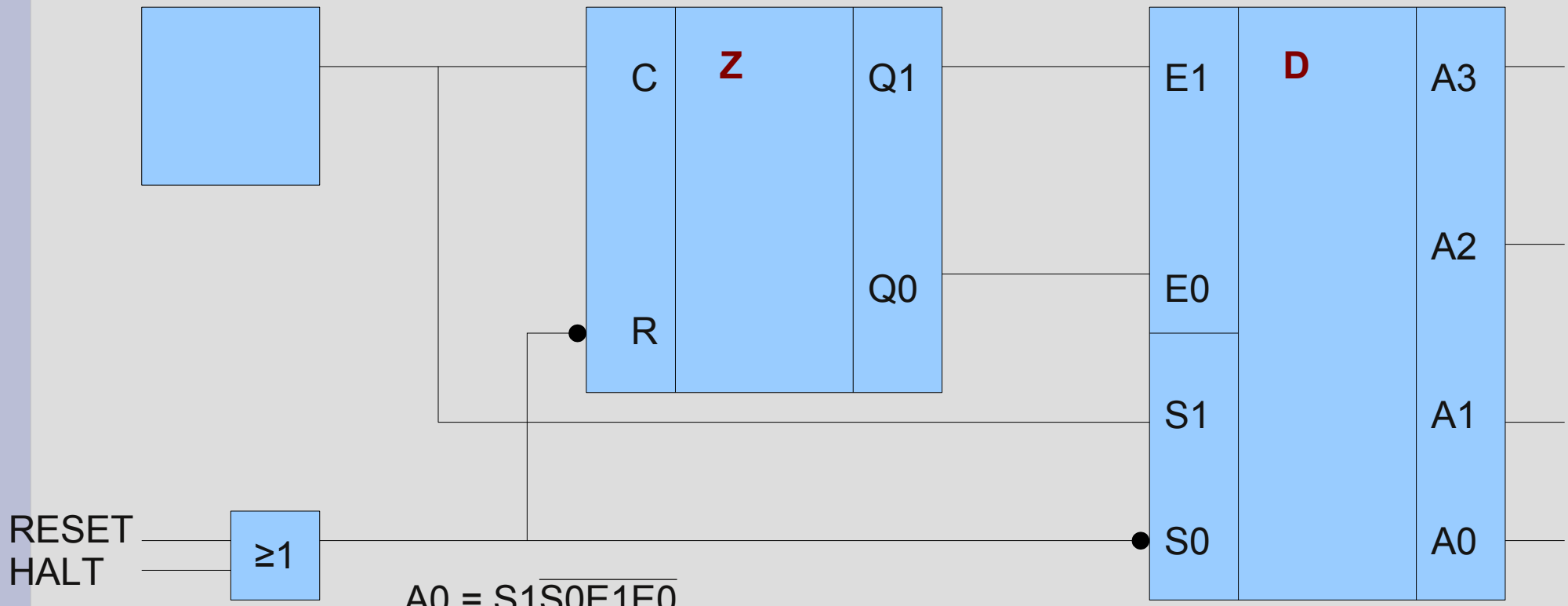
    ⇒ Zähler **Z** aus 2 FF, mit RESET v HALT initialisiert

    ⇒ Dekoder **D** zur Generierung der 3 Taktphasen aus den drei Zuständen, RESET v HALT gesperrt

Feinentwurf des Systems



Feinentwurf des Systems



$A0 = S1\overline{S0}E1\overline{E0}$
$A1 = S1\overline{S0}E1E0$
$A2 = S1\overline{S0}E1\overline{E0}$
$(A3 = S1\overline{S0}E1E0)$

Hinweis: Der Initialzustand muss der Zustand sein, der zuerst (bei T1) entschlüsselt werden soll.
Festlegung: Zustand 00

Feinentwurf des Systems

- Taktversorgung
  - Entwurf des Zählers

    siehe Vorlesung

Feinentwurf des Systems

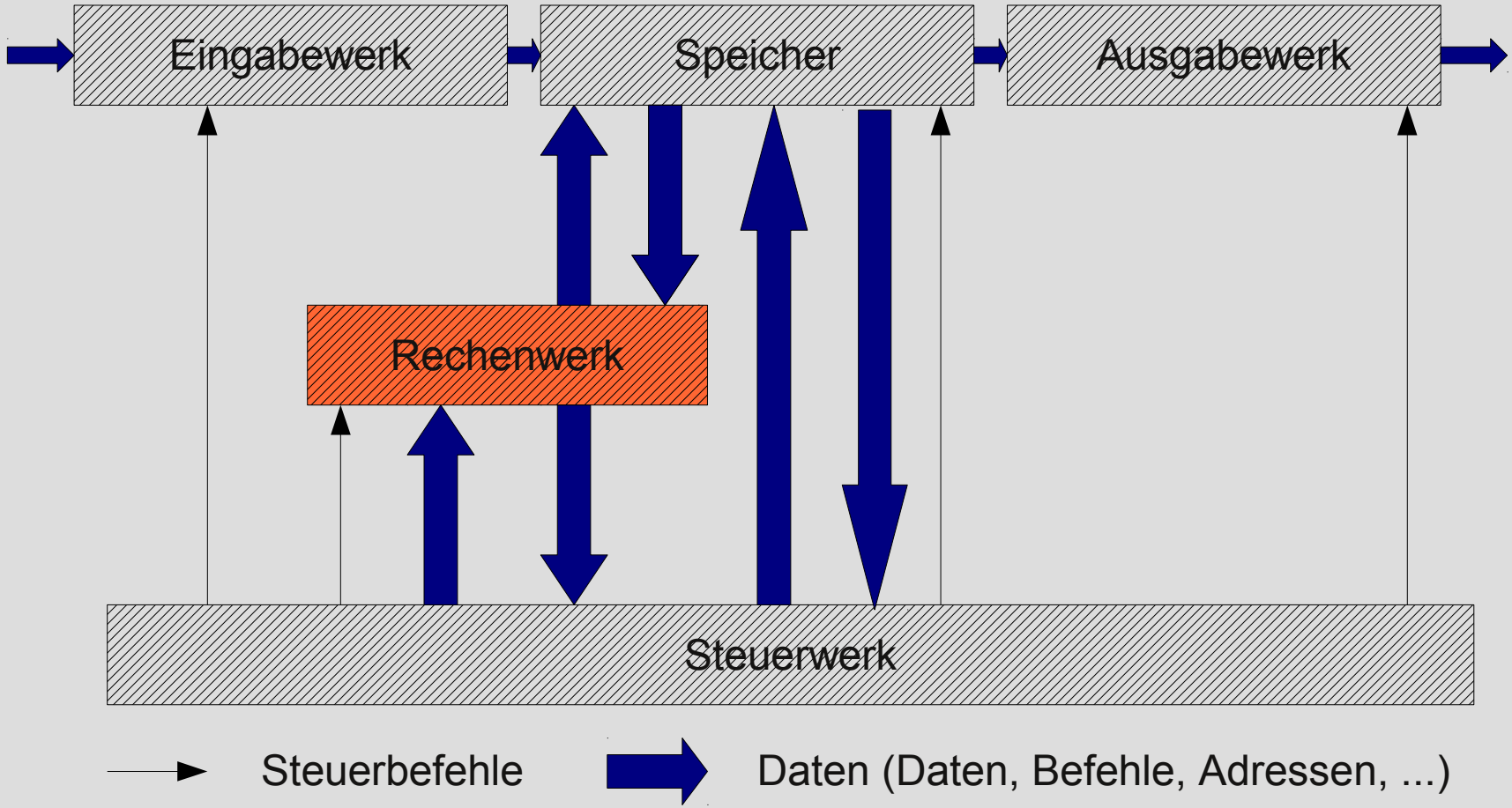
- (Mikroprogramm-) Steuerwerk (Teil 1)
  - Erzeugung der Signale für das Rechenwerk
  - Gewährleistung der Sequenz
  - Beeinflussung der Abarbeitungsreihenfolge
    - Adr := Adr +1 (autoinkrement)
    - Adr := Sprungziel (unbedingter Sprung)
    - If Bedingung then Adr := Sprungziel (bedingter Sprung)
  - Anhalten der Abarbeitung im letzten µ-Befehl (HALT-Bit)

Datenpfad

- Die Komponente des Prozessors, die arithmetische und logische Operationen ausführt. Der Datenpfad umfasst also die Komponenten zur Verarbeitung der Daten.

Datenpfad (2)

- v.-Neumann-Architektur:



Datenpfad (3)

- Beispielarchitektur:
  - Auszug aus einem Komplexbeispiel:
    - Hardware für das Einlesen von Zahlen von der Tastatur und deren Speicherung als Binärzahlen im 2-er Komplement
  - es soll ein 8-Bit-Rechner entworfen werden.
  - von der ALU sollen folgende Operationen bereitgestellt werden:
    - add, add + carry, sub, and, not, shl, mul (evtl. durch add, sub und zusätzlich shr zu emulieren).

Datenpfad (4)

- Darstellung des gesamten Programms (= Aufgabenstellung) in Pascal-ähnlicher Notation
- Pro Takt soll Schaltung arbeiten, d.h. Rechenwerk wird aufgerufen mit
  schaltung (RESET, STROBE, DATEN, READY, HALT)

Datenpfad (5)

- Daraus folgt dann

```
schaltung (1, x, x, ready, halt)
repeat
    repeat
        schaltung (0,0,x,ready,halt)
    until ready = 1                          Warten auf READY
    daten := ascii_zeichen
    repeat
        schaltung(0,1,daten,1,halt)  Warten auf READY
until daten = <lf>      d.h. Das für <lf> vereinbarte Steuerzeichen
repeat
        schaltung(0,0,x,ready,halt)  Warten auf HALT
```

- Ab jetzt steht die konvertierte Zahl im Speicher

Datenpfad (6)

- Algorithmus:
  - Vereinbarungen
    - (i) = (Adresse i) => Inhalt der Speicherzelle i
    - ((i)) = Inhalt der Speicherzelle, deren Adresse in Speicherzelle i steht
    - 8-Bit ALU!

Datenpfad (7)

```
if reset = 1
        then {Rücksetzen der gesamten
                Schaltung, darunter HALT = 0}
        else BEGIN   {Daten von Tastatur einlesen}
                    (1) := (0);
                    REPEAT
                        ready := 1;
                        while strobe = 0 do;
                        ((1)) := daten;
                        (1) := (1)+1;
                        ready := 0;
                UNTIL daten=LF; {vereinbartes SZ}
```

Datenpfad (8)

```
{Einer}
                (1) := (1)-3;   {im Datenbereich auf letzte Ziffer
                                  stellen}
                ((2)) := (1) and 0FH; {niederwertige 4 Bit
                                        maskieren}
                (1) := (1) -1;
{Zehner}
if ((1) ≥ 0) and (((1)) ≠ 2BH) and (((1)) ≠ 2DH)  {+,-
                                                    ausschliessen}
  then begin
        ((2)) := ((2)) + (((1)) and 0FH) * 0AH;
        (1) := (1) -1;
```

Datenpfad (9)

```
{Hunderter}
        if
            PROD := (((1)) and 0FH) * 64H;
            ((2)) := ((2)) + letzte8(PROD);
            ((2)+1) := erste8(PROD)+CY;
            (1) := (1) -1;
```

Datenpfad (10)

```
{Tausender}
        if
            then begin
                    PROD := (((1)) and 0FH) 68H;
                    ((2)) := ((2)) + letzte8(PROD);
                    ((2)+1) := ((2)+1) + erste8(PROD) + CY;
                    PROD := (((1)) AND 0FH * 07H;
                    PROD := SHL7(PROD);
                    ((2)):= ((2))+letzte8(PROD);
                    ((2)+1):=((2)+1)+erste8(PROD)+CY;
                    (1):= (1)-1;
                    {1000 := 3E8H = 380H+68H= 7*128+104}
```

Datenpfad (11)

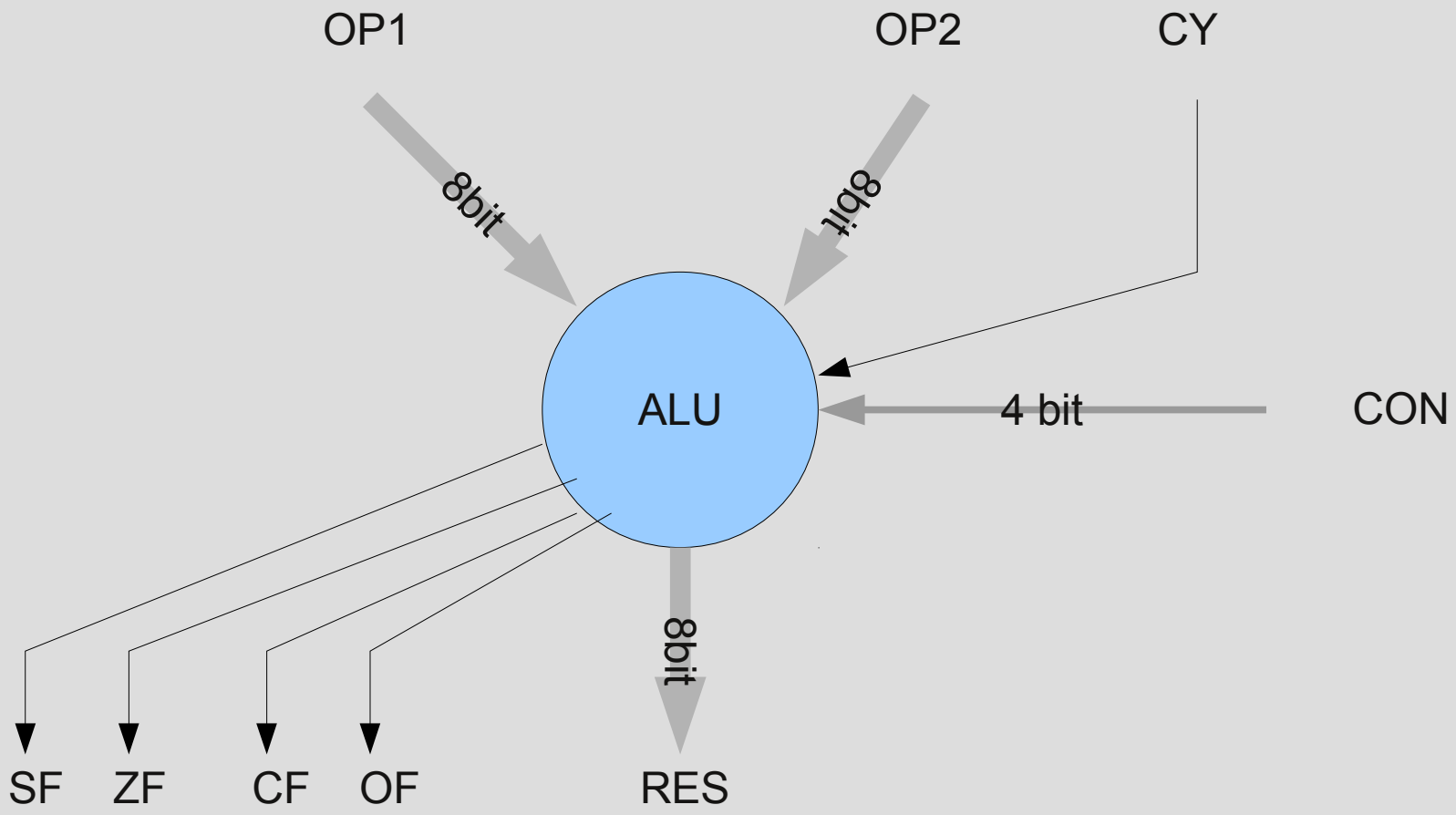
```
{Zehntausender}
        if
            then begin
                    PROD:=(((1)) and 0FH)*10H;
                    ((2)) := ((2)) + letzte8(PROD);
                    ((2)+1) := ((2)+1) + erste8(PROD) + CY;
                    PROD := (((1)) and 0FH)*27H;
                    PROD := SHL8(PROD);
                    ((2)+1):= ((2)+1) + erste8(PROD);
{10000 = 2710H = 2700H+10H = 39 * 256+16=
   27H+100H+10H}
            end;
        end; end; end;
```

Datenpfad (12)

```
if(1)  = 2DH {negativ}
     then begin
               ((2)) := not ((2));
               ((2)+1) := not (((2)+1));
               ((2)) := ((2))+1;
              ((2)+1) := ((2)+1)+CY;
           end;
   HALT := 1;
end;
```

Datenpfad (13)

- ALU-Struktur:



Datenpfad (14)

- Kondierung der Befehle (CON):

CON *3 2 1 0*	RES
0 0 0 0	OP1 + OP2
0 0 0 1	OP1 + OP2 + CY
0 0 1 0	OP1 - OP2
0 0 1 1	OP1 - OP2 - CY
0 1 0 0	OP1 & OP2
0 1 0 1	OP1 | OP2
0 1 1 0	OP1
0 1 1 1	OP2
1 0 0 0	!OP1
1 0 0 1	!OP2
1 0 1 0	OP1 + 1
1 0 1 1	OP2 + 1

Datenpfad (15)

- Kodierung der Befehle (CON):

CON *3 2 1 0*	RES
0 0 0 0	OP1 + OP2
0 0 0 1	OP1 + OP2 + CY
0 0 1 0	OP1 - OP2
0 0 1 1	OP1 - OP2 - CY
0 1 0 0	OP1 & OP2
0 1 0 1	OP1 | OP2
0 1 1 0	OP1
0 1 1 1	OP2
1 0 0 0	!OP1
1 0 0 1	!OP2
1 0 1 0	OP1 + 1
1 0 1 1	OP2 + 1

Datenpfad (16)

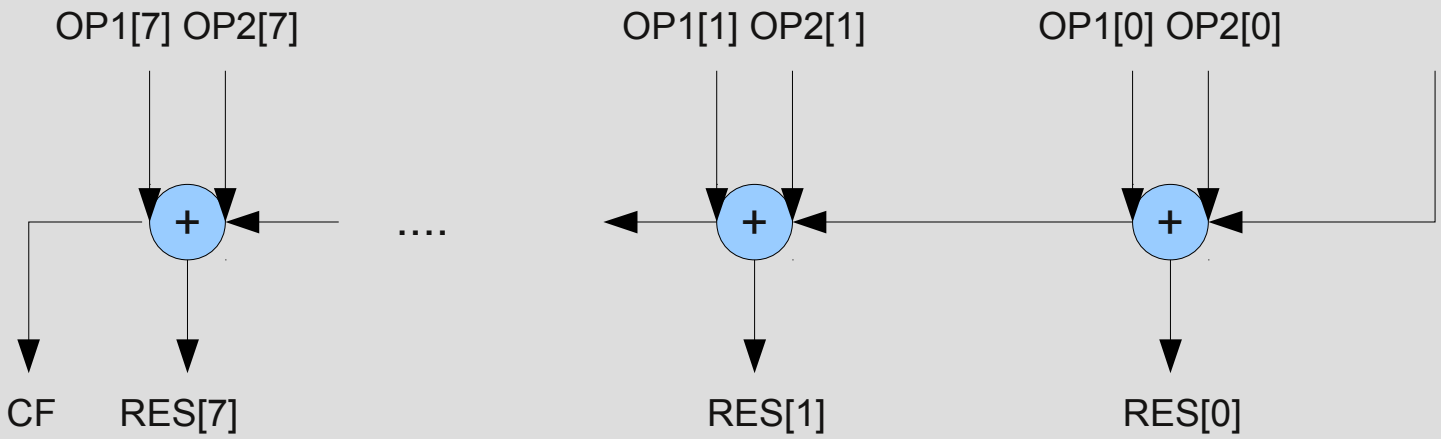
- Flags:
  - Sign-Flag: SF = RES[7];
  - Zero-Flag: ZF = $\overline{RES[7]}$ & $\overline{RES[6]}$ & .. & $\overline{RES[0]}$;
  - Overflow-Flag

CON	OP1[7]	OP2[7]	RES[7]	
+	0	0	1	
+	1	1	0	OF = 1;
-	0	1	1	
-	1	0	0	

sonst OF = 0;

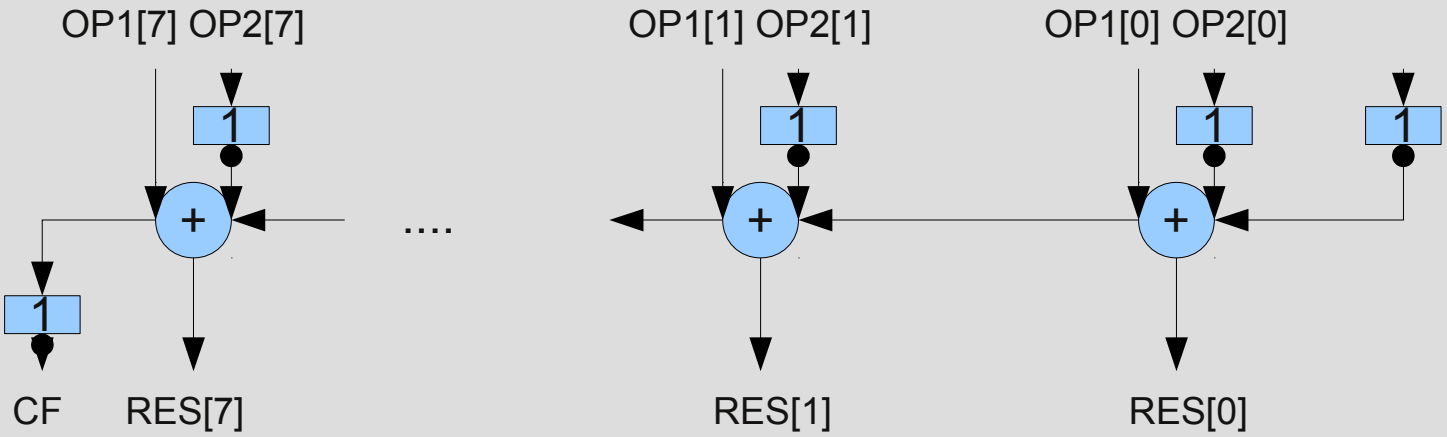
Datenpfad (17)

- Flags:
  - Carry-Flag (Übertrag-Flag)
    - Addition



Datenpfad (18)

- Flags:
  - Carry-Flag (Übertrag-Flag)
    - Subtraktion

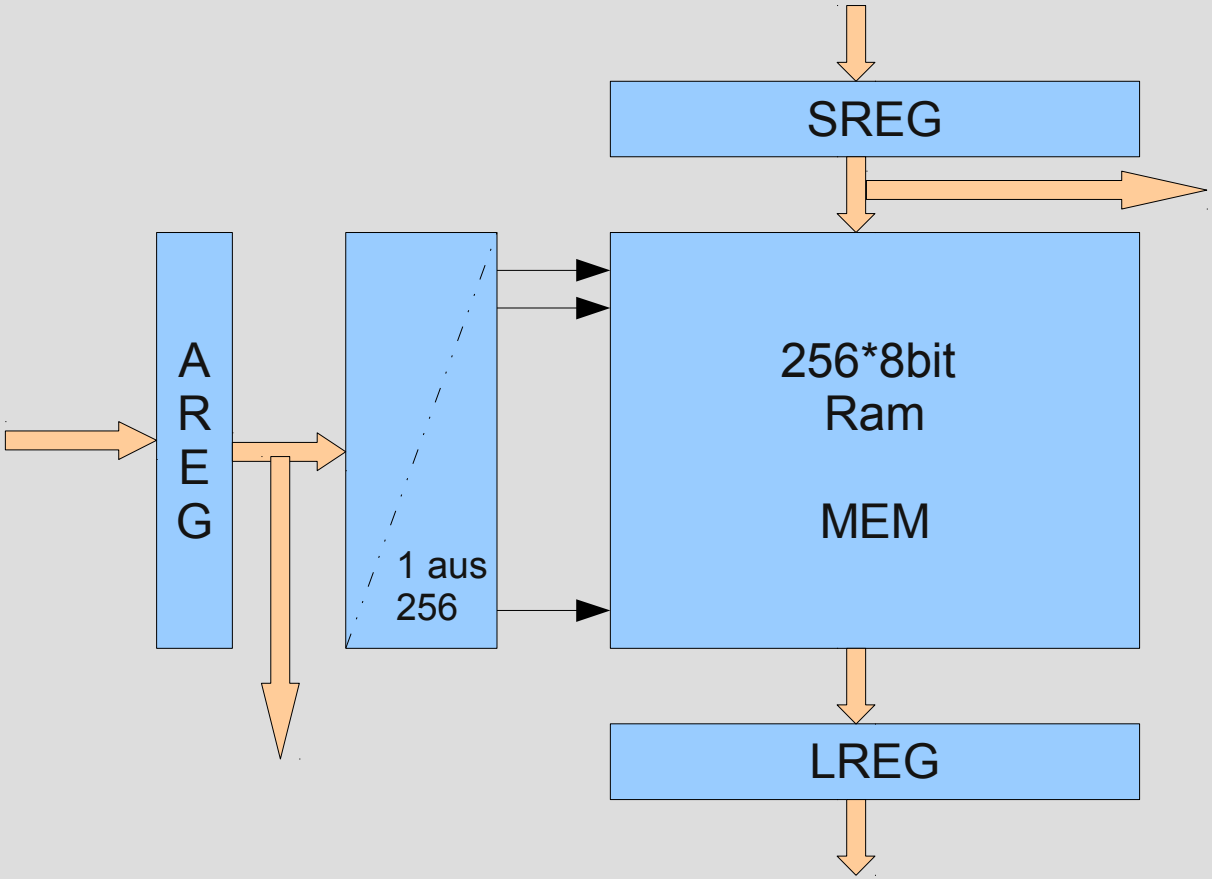


Datenpfad (19)

- Registerstruktur:
  - abzuleiten aus
    - auszuführende Operationen
    - Format der zu behandelnden Operanden:
      - 8 Bit -Rechner ⇒ 8 bit Operanden
      - aber 16 bit Ergebniss (z.B. Multiplikation)
      - man könnte damit auch insgesamt auf eine 16 bit Architektur gehen (aus Aufwandsgründen wird darauf verzichtet)
    - Operationen mit 16-bit Ergebnis:
      - Multiplikation
      - 16 bit-Addition:
        - ((Z)) := ((Z)) + lower8 (PROD)
        - ((Z)+1) := ((Z)+1) + higher8 (PROD) + CY
    - OP1 und OP2 16 bit Register (eigentlich 2 * 8 bit)

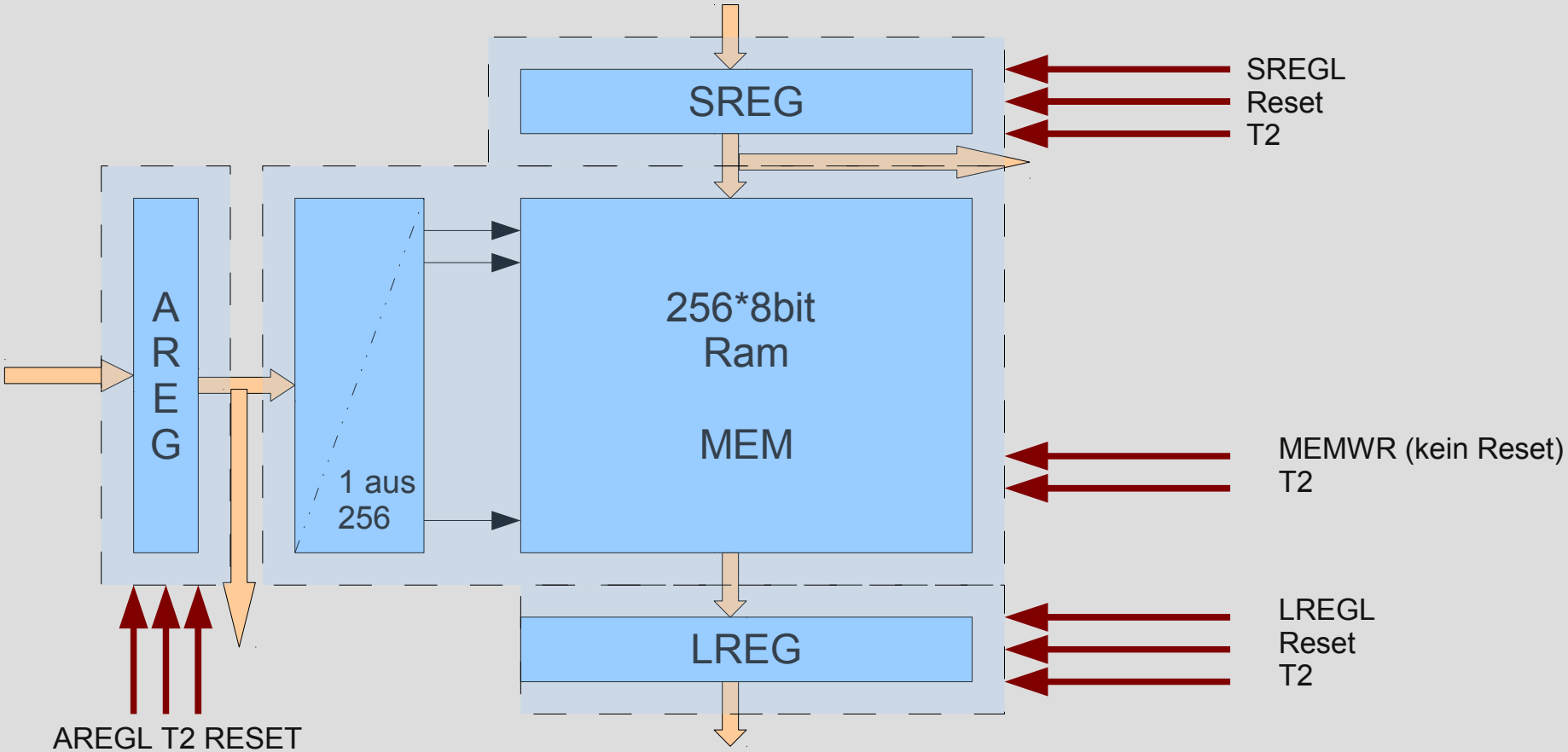
Datenpfad (20)

- Entwurf des Speichers



Datenpfad (21)

- Entwurf des Speichers mit notwendigen Steuersignalen



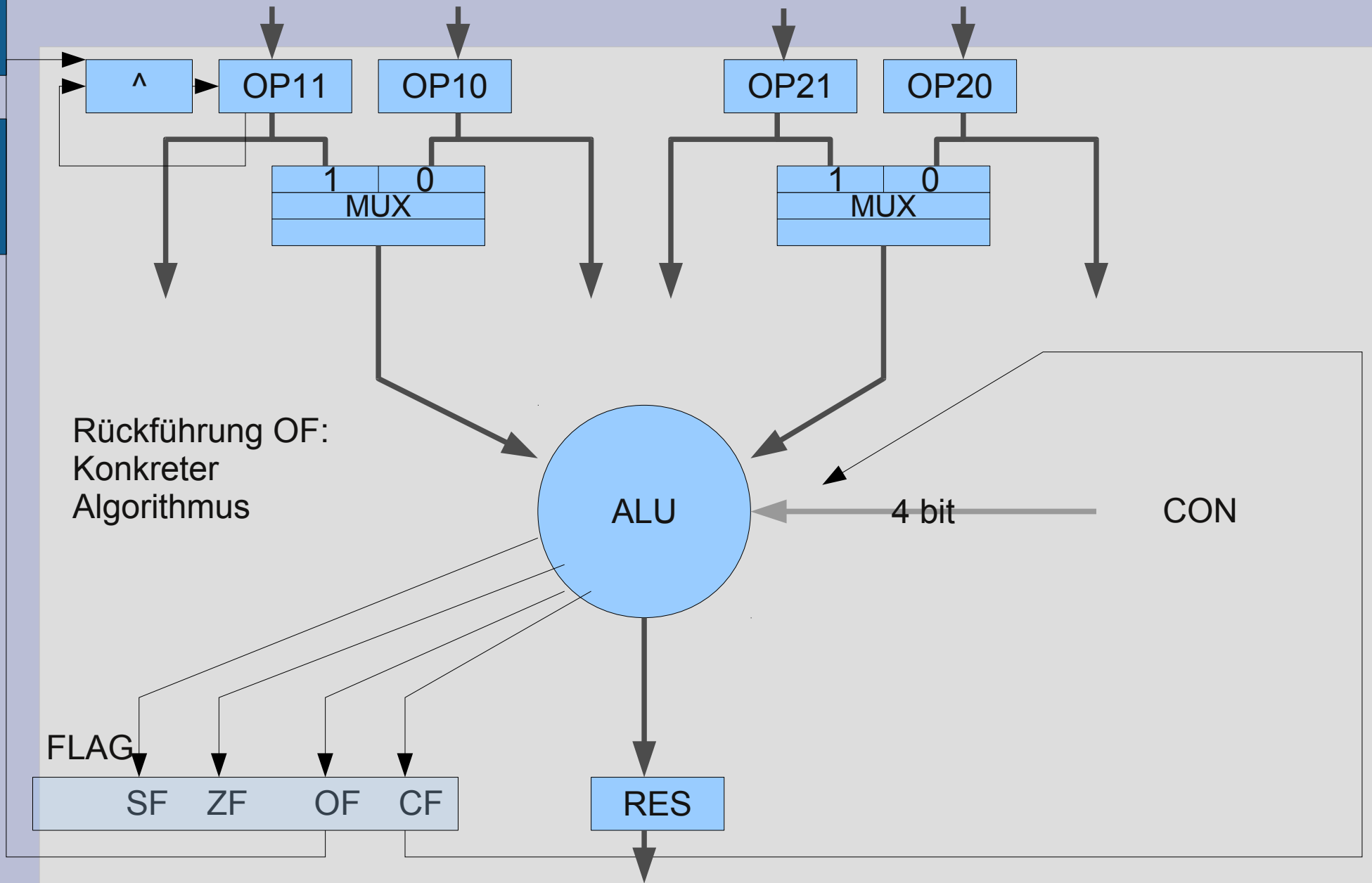
Datenpfad (22)

- AREG, SREG und LREG sowie MEM sind T2 getaktet (Schreiben und Lesen erfordern 2 Mikrobefehle)
  - Schreiben:
    - Laden SREG, AREG (AREGL =1, SREGL=1, MEMWR =0)
    - Schreiben in MEM (AREGL = X, SREGL=X, MEMWR =1)

Datenpfad (23)

- AREG, SREG und LREG sowie MEM sind T2 getaktet (Schreiben und Lesen erfordern 2 Mikrobefehle)
  - Lesen:
    - Laden AREG (AREGL = 1; MEMWR = 0)
    - Laden von LREG (AREGL=X, LREGL=1)

Datenpfad (24) Gesamtentwurf



Steuerwerk

- Aufgaben des Steuerwerkes:
  - Laden der Programmbefehle aus dem Speicher in der richtigen Reihenfolge
  - Dekodieren der Befehle
  - Interpretation der Befehle
  - Versorgung der an der Ausführung der Befehle beteiligten Funktionseinheiten mit den notwendigen Steuersignalen

Steuerwerk (2)

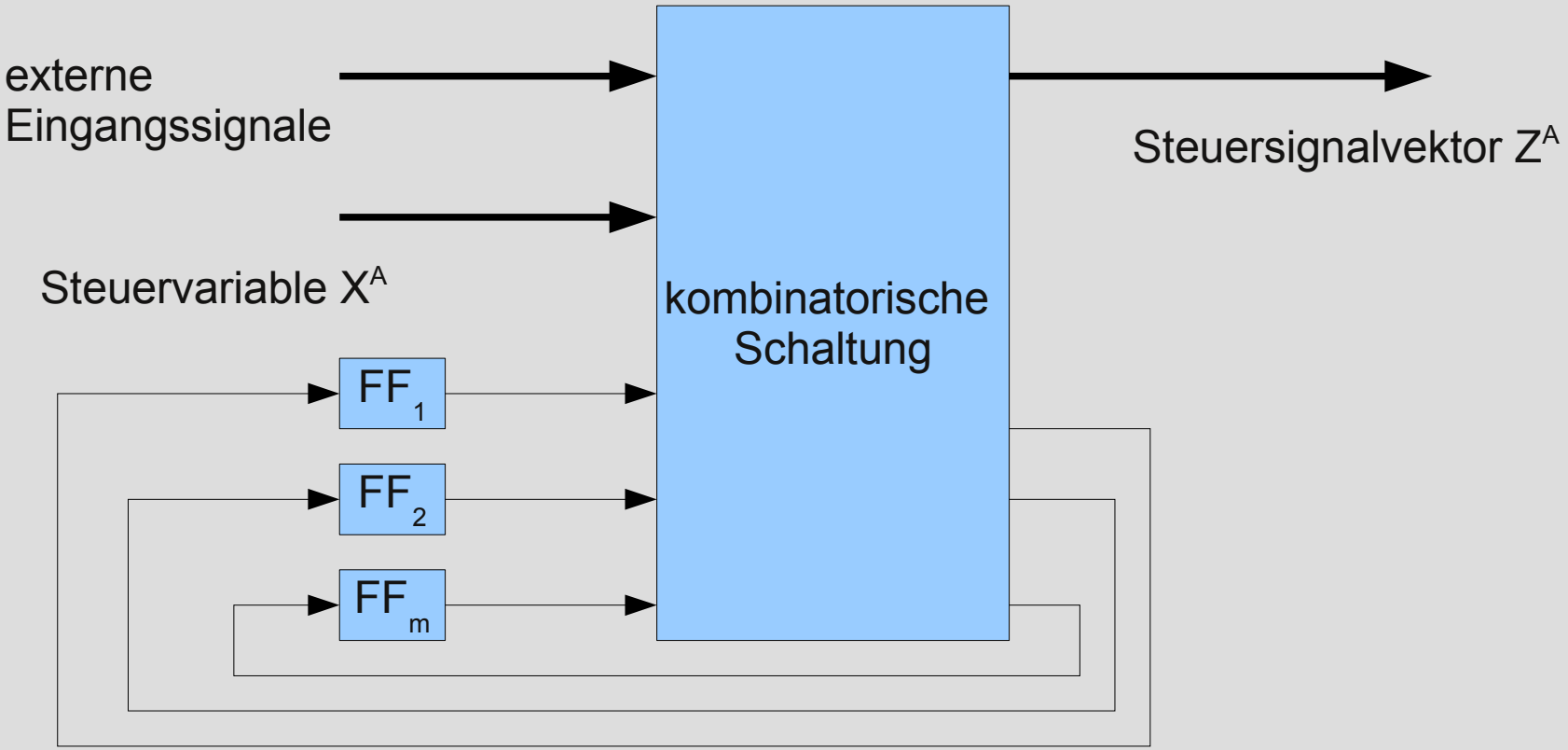
- allgemein:
  - $A = (X^A, Y^A, Z^A, f^A, g^A)$

  mit

  X – Steuervariablenvektor
  Y – Zustandsmenge
  Z – Steuersignalvektor
  f – Überführungsfunktion $y_{t+1} = f(x_t, y_t)$
  g – Ausgabefunktion $z_t = g(x_t, y_t)$
- Methoden des Steuerwerkentwurfes:
  - verdrahtes Steuerwerk
    - Zustandstabellenmethode
    - Folgezählermethode

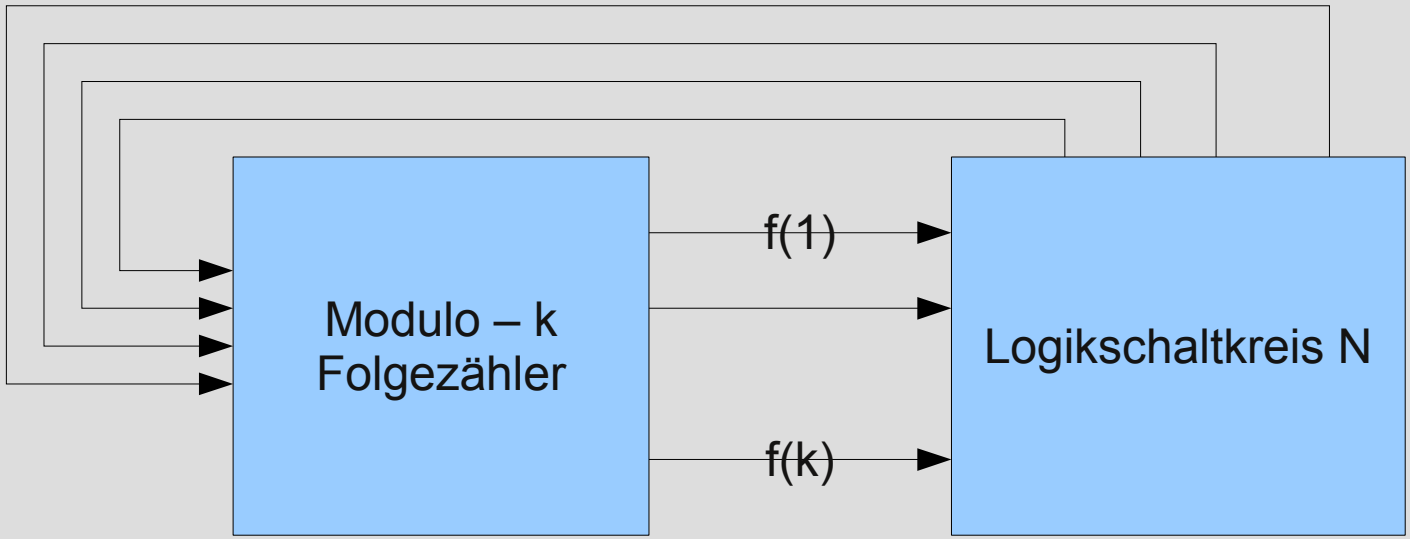
Steuerwerk (3)

- Methoden des Steuerwerkentwurfes:
  - verdrahtes Steuerwerk
    - Zustandstabellenmethode (z.B. Z80)



Steuerwerk (4)

- Methoden des Steuerwerkentwurfes:
  - verdrahtes Steuerwerk
    - Zustandstabellenmethode
    - Folgezählermethode

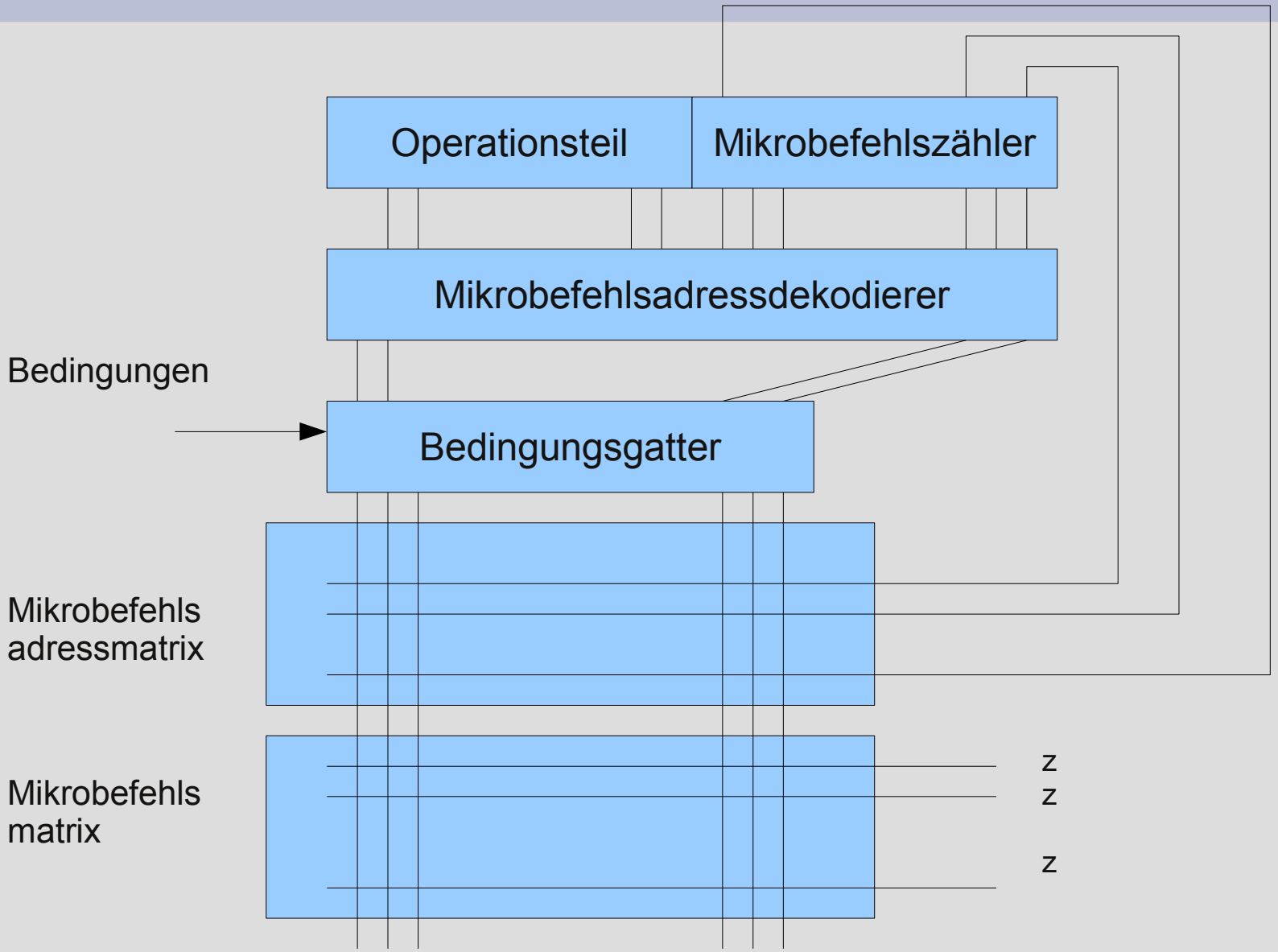


Aufteilung des Grundtaktes in k Phasen f(i) mit i= 1,...,k

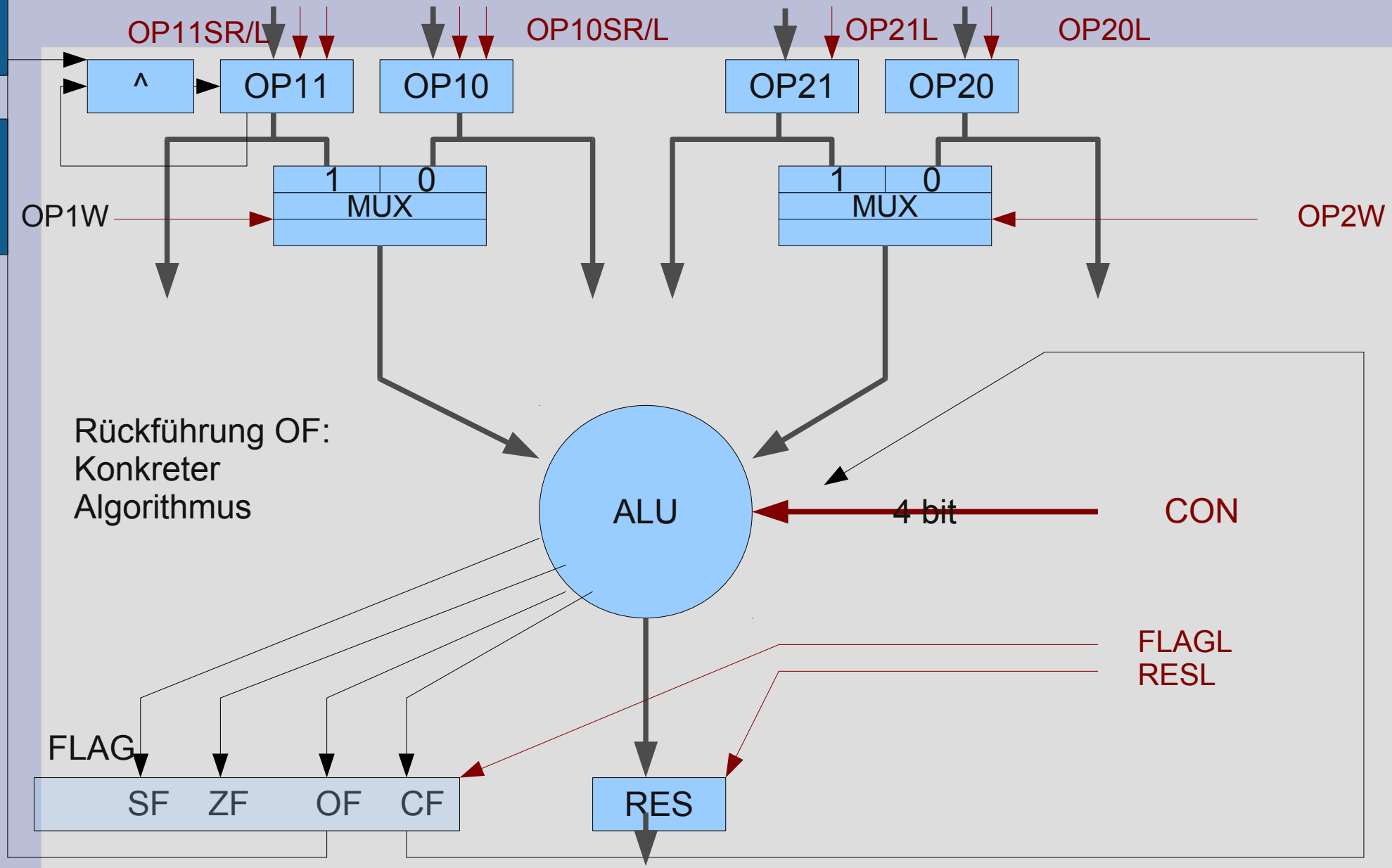
Steuerwerk (5)

- Methoden des Steuerwerkentwurfes:
  - verdrahtes Steuerwerk
    - Zustandstabellenmethode
    - Folgezählermethode
  - Mikroprogrammsteuerwerk
    - mithilfe der dekodierten Informationen des Operationskodes wird eine Folge von Signalen zur Ausführung eines Befehls erzeugt.
      - fest verdrahtet
      - programmierbar
      - vertikal
      - horizontal

Steuerwerk (6)



Steuerwerk (7) Steuersignale



Steuerwerk (8)

- Mikroprogrammsteuerwerk für Beispielprozessor
  - Erzeugung der Signale für andere Komponenten
  - Gewährleitung der Sequenz
    - Beeinflussung der Abarbeitungsreihenfolge
      - Adr := adr + 1
      - Adr := sprungziel {unbedingter Sprung}
      - If bed then adr := sprungziel {bedingter Sprung}
    - Anhalten der Abarbeitung im letzten Mikrobefehl

      → HALT-Bit

Steuerwerk (9)