Prozessorarchitektur

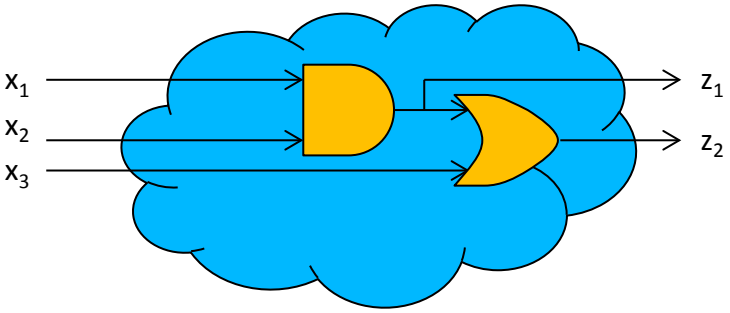
Kapitel 1 - Wiederholung

M. Schölzel

Wiederholung

Kombinatorische Logik:

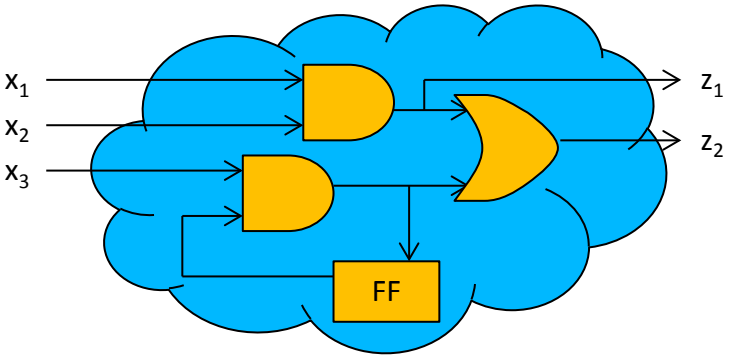
- Ausgaben hängen funktional von den Eingaben ab.



$$z_1 = f_1(x_1,\ldots,x_n)$$
$$\vdots$$
$$z_m = f_m(x_1,\ldots,x_n)$$

Sequentielle Logik

- Keine funktionale Abhängigkeit von den Eingaben wegen internem Speicher

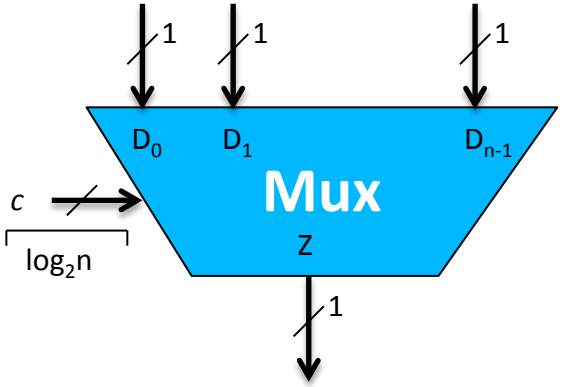


$$z_1 = f_1(FF_1,\ldots,FF_k,x_1,\ldots,x_n)$$
$$\vdots$$
$$z_m = f_m(FF_1,\ldots,FF_k,x_1,\ldots,x_n)$$

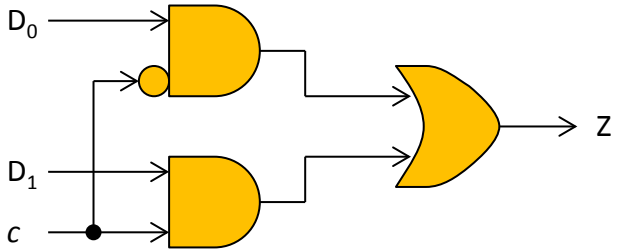
Wiederholung (kombinatorische Logik)

Multiplexer (Beispiel für kombinatorische Logik)

- D_i … Dateneingänge (1 Bit)
- Z … Datenausgang (1 Bit)
- c … Steuereingang, kodiert binär eine Zahl von 0 bis n-1
- Funktion des Multiplexers: $Z(D_0,\ldots,D_{n-1},c) = D_c$



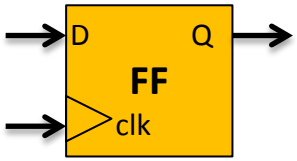
2:1-Multiplexer aus Grundgattern:

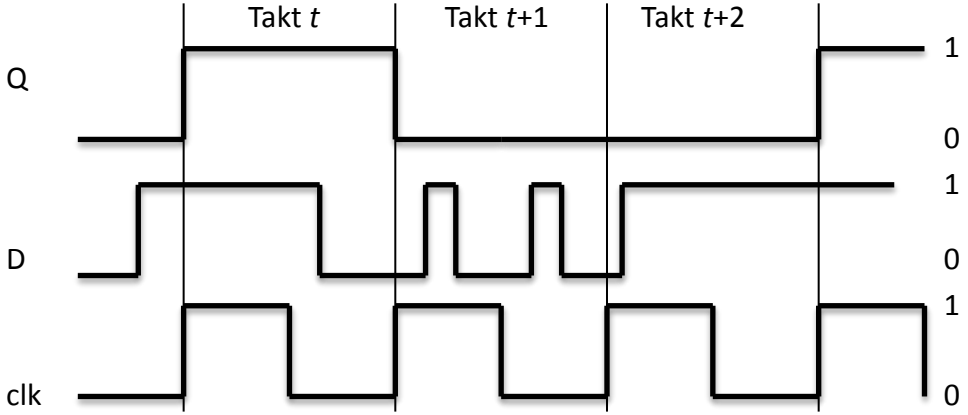


Wiederholung (sequentielle Logik)

Verhalten flankengesteuerte Flip-Flops (D-Flip-Flop)

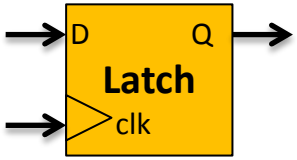
- D … Dateneingang
- Q … Datenausgang
- clk … Takteingang

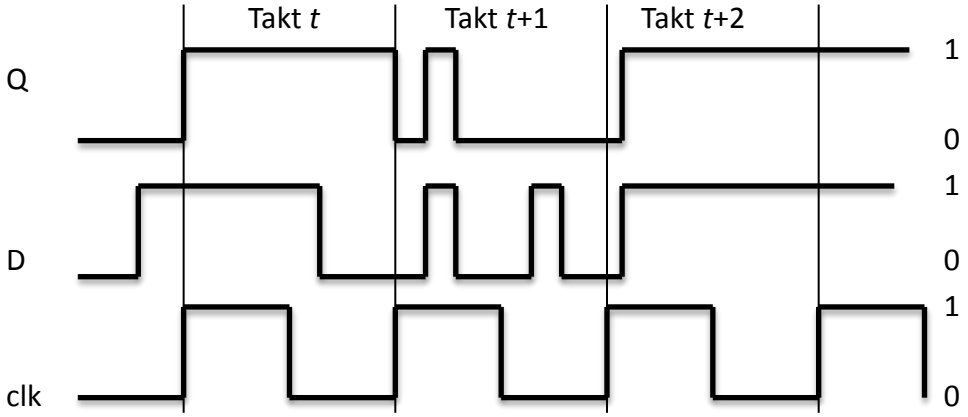




Verhalten zustandsgesteuerte Flip-Flops (D-Latch)

- D … Dateneingang
- Q … Datenausgang
- clk … Takteingang



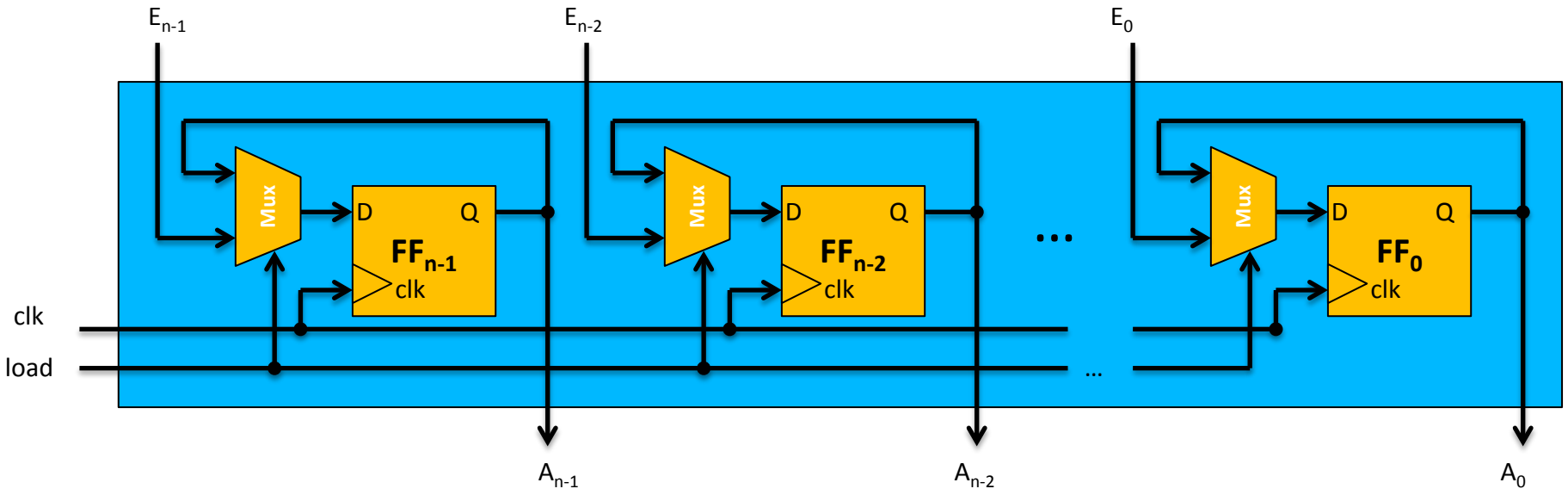


Wiederholung (sequentielle Logik)

Register

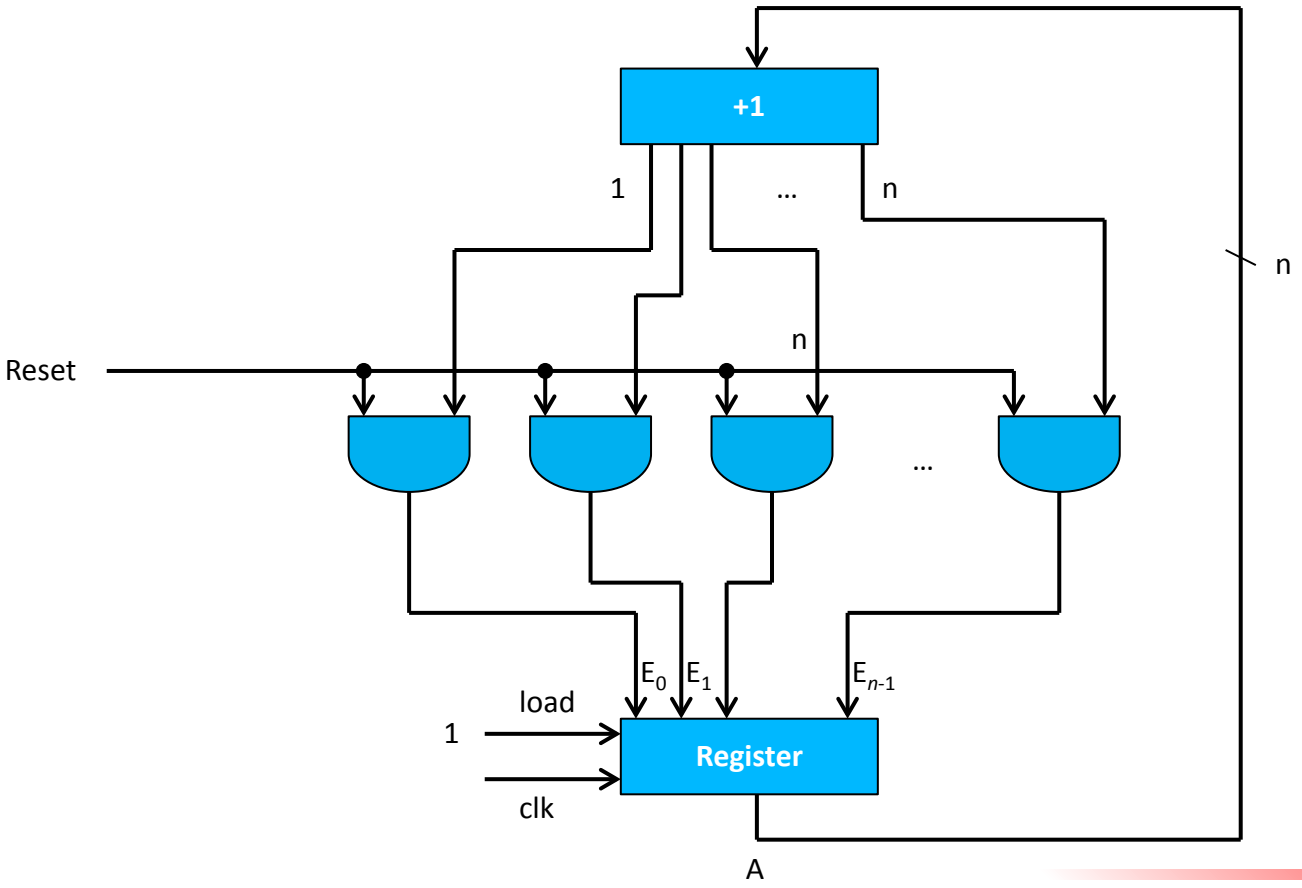
- Zum schnellen Speichern von *n* Binärwerten (z.B. in einem Prozessor)
- $E_{n-1},\ldots,E_0$ … Dateneingänge
- $A_{n-1},\ldots,A_0$ … Datenausgänge
- load … Ladesignal
- Verhalten:

$$A(clk) = \begin{cases} A(clk-1), & \text{falls } load = 0 \\ E(clk-1), & \text{sonst} \end{cases}$$



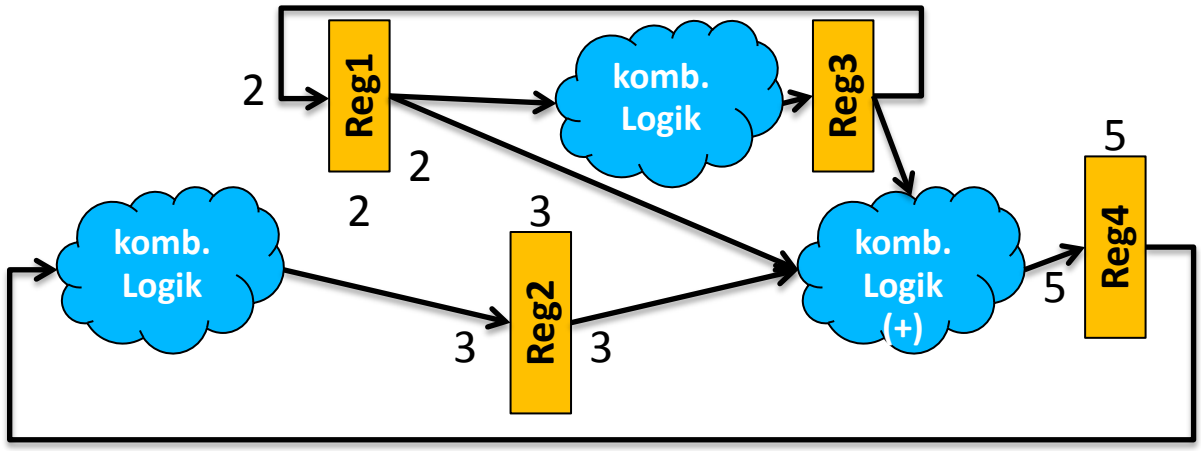
Wiederholung (sequentielle Logik)

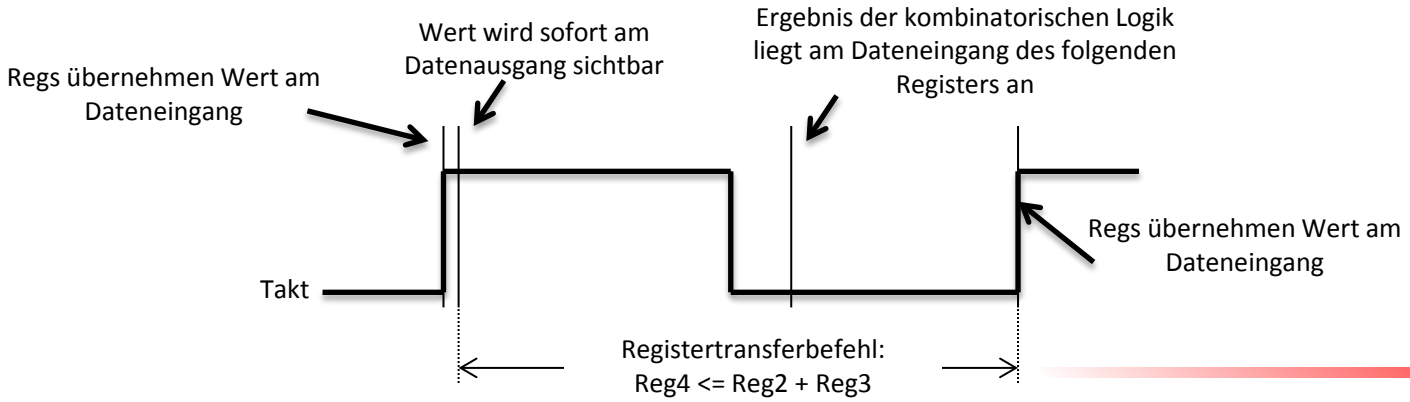
- *n*-Bit Zähler mit synchronem Reset (low-aktiv):



Registertransferbefehle

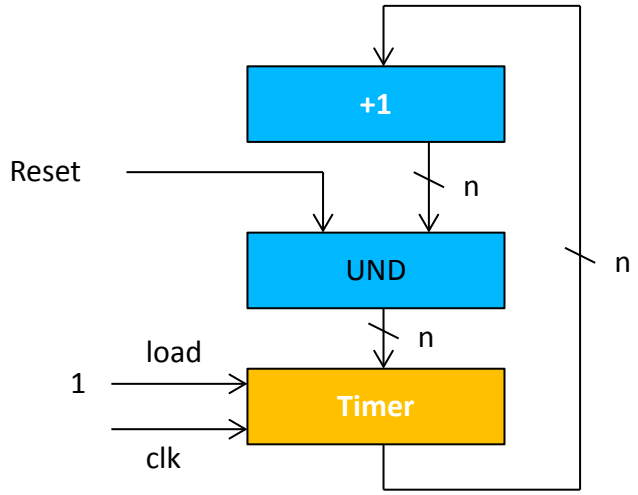
Beschreiben die Änderung von Registerwerten innerhalb eines Taktes in einer Schaltung, die aus Registern und kombinatorischer Logik aufgebaut ist.





Beispiele: Registertransferbefehle

Registertransferbefehle (Beispiel)



- ✓ Timer <= Timer + 1
- ✓ Timer <= 0
- ✘ Timer <= Timer - 1
- ✘ Timer <= Timer + 2
- ✘ if reset Timer <= 0 else Timer <= Timer + 1
- ✓ if reset Timer <= Timer + 1 else Timer <= 0

Grundstruktur eines von-Neumann-Computers

Computer besteht aus

- Prozessor + Speicher + Ein-/Ausgabe

Speicher besteht aus

- Worten fester Länge und enthält Daten und Instruktionen im gleichen Speicher (von-Neumann-Architektur)

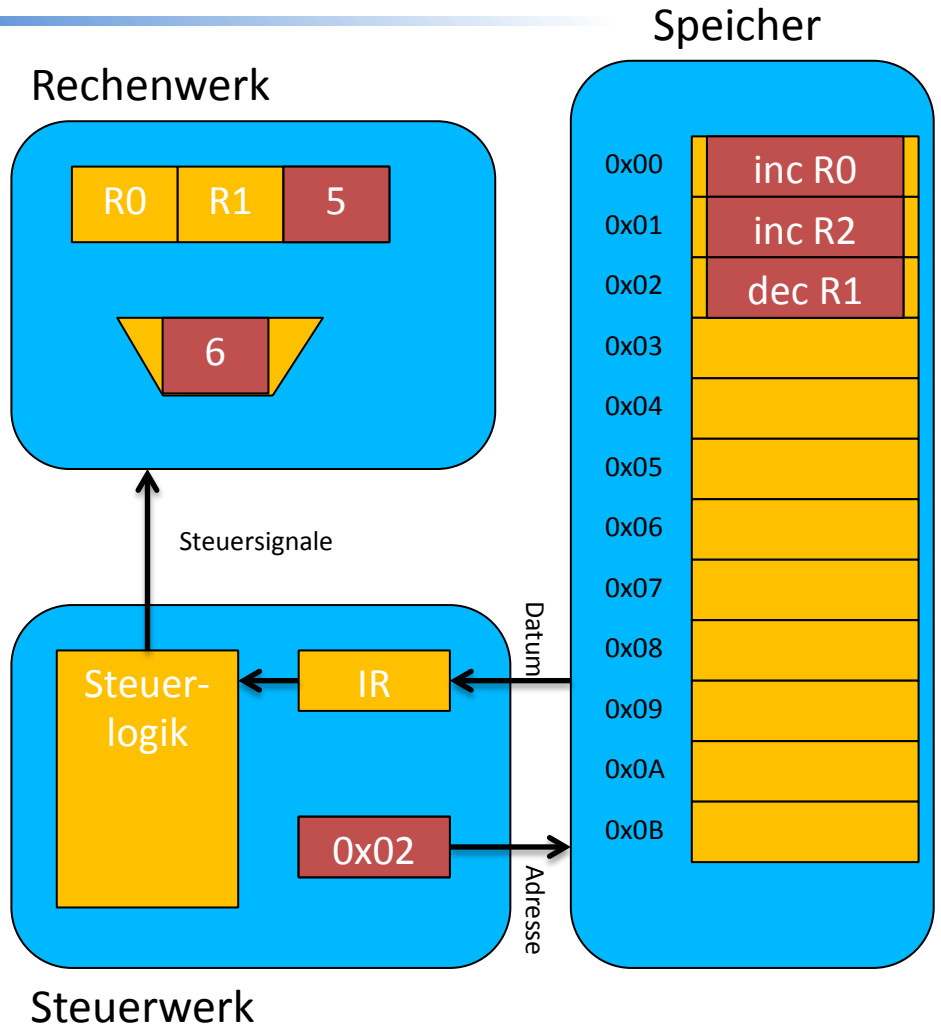
Prozessor besteht aus Rechenwerk und Steuerwerk

- Rechen- und Steuerwerk = Central Processing Unit (CPU)
- Im Program Counter (PC, Befehlszähler) steht die Speicheradresse der nächsten auszuführenden Instruktion
- Weitere Register im Rechenwerk, da Operationen mit Registern schneller ausführbar sind als mit Operanden, die sich im Speicher befinden

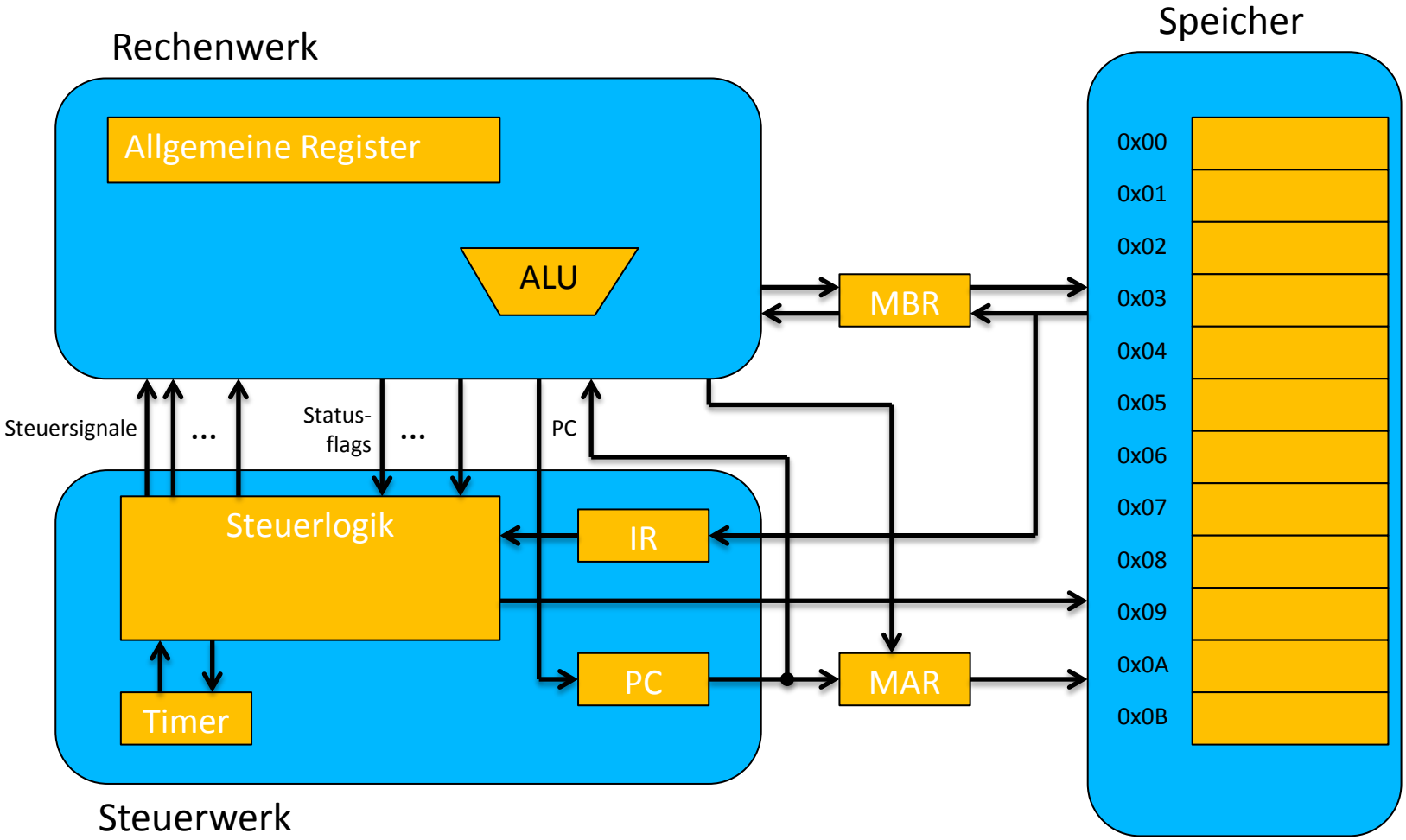
Prinzip eines programmierbaren Prozessors

- PC enthält Speicheradresse des aktuellen Befehls
- Befehl von dieser Adresse aus dem Speicher in das Instruktionsregister (IR) holen
- PC auf Adresse des nächsten Befehls setzen
- Aktuellen Befehl im IR im Datenpfad ausführen
- Nächsten Befehl holen…
- Es ergibt sich folgende Verarbeitungsschleife:
  - Aktuellen Befehl in das IR holen und PC aktualisieren (FE)
  - Befehl im IR verarbeiten:
    - Operandenwerte laden (DE)
    - Operandenwerte verarbeiten (EX)
    - Ergebnis zurückschreiben (WB)

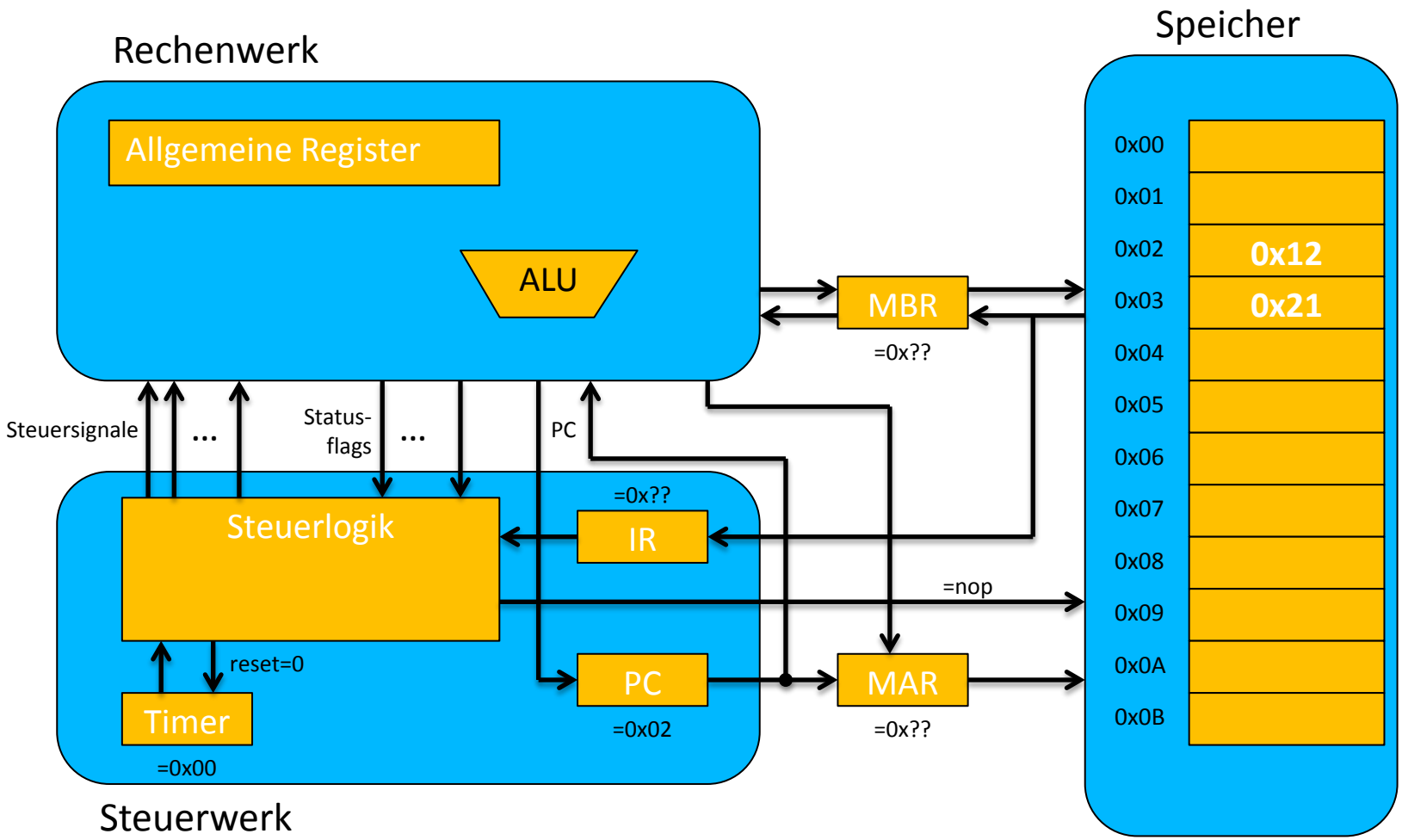
Und jetzt die Details…



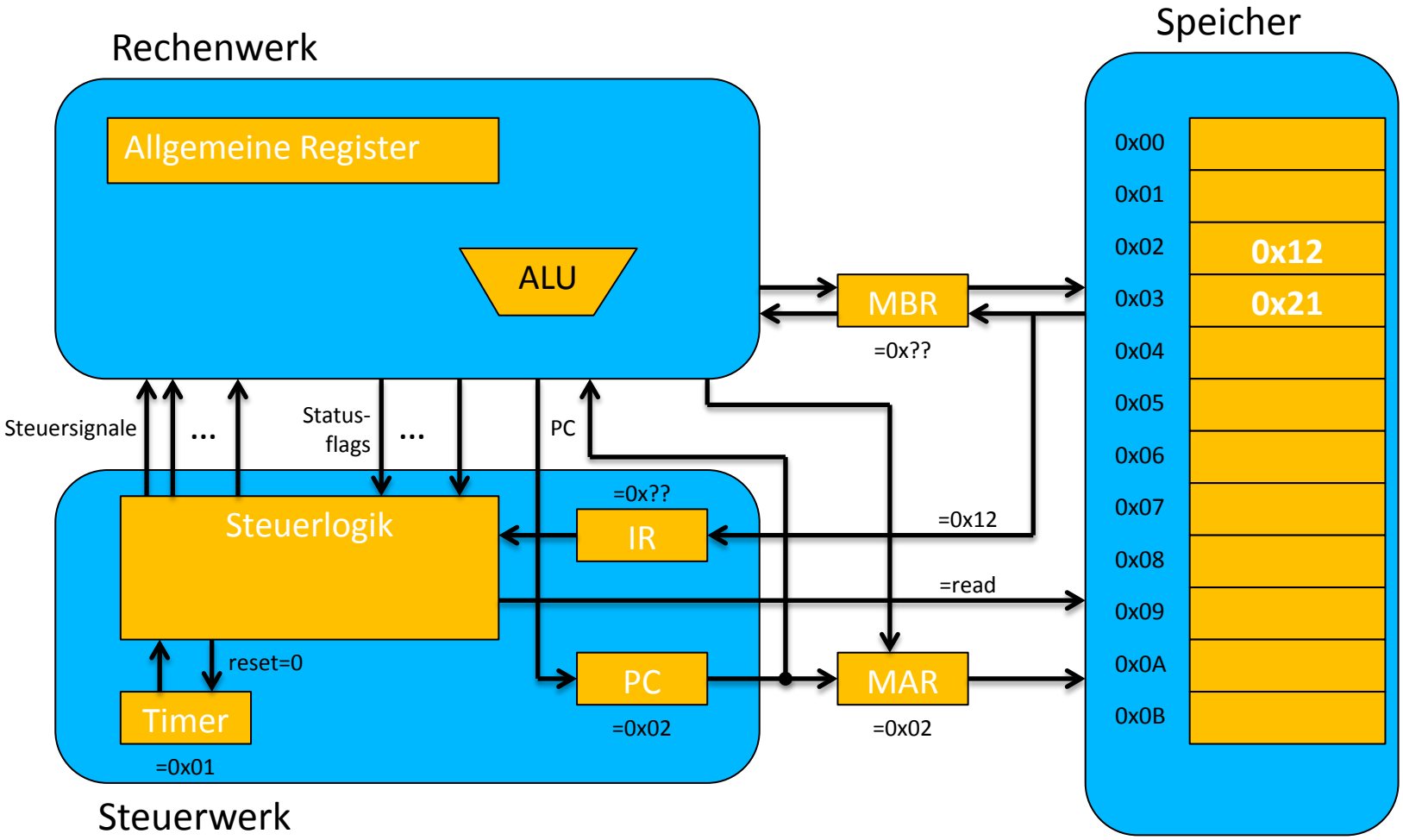
Blockschaltbild eines einfachen Prozessors



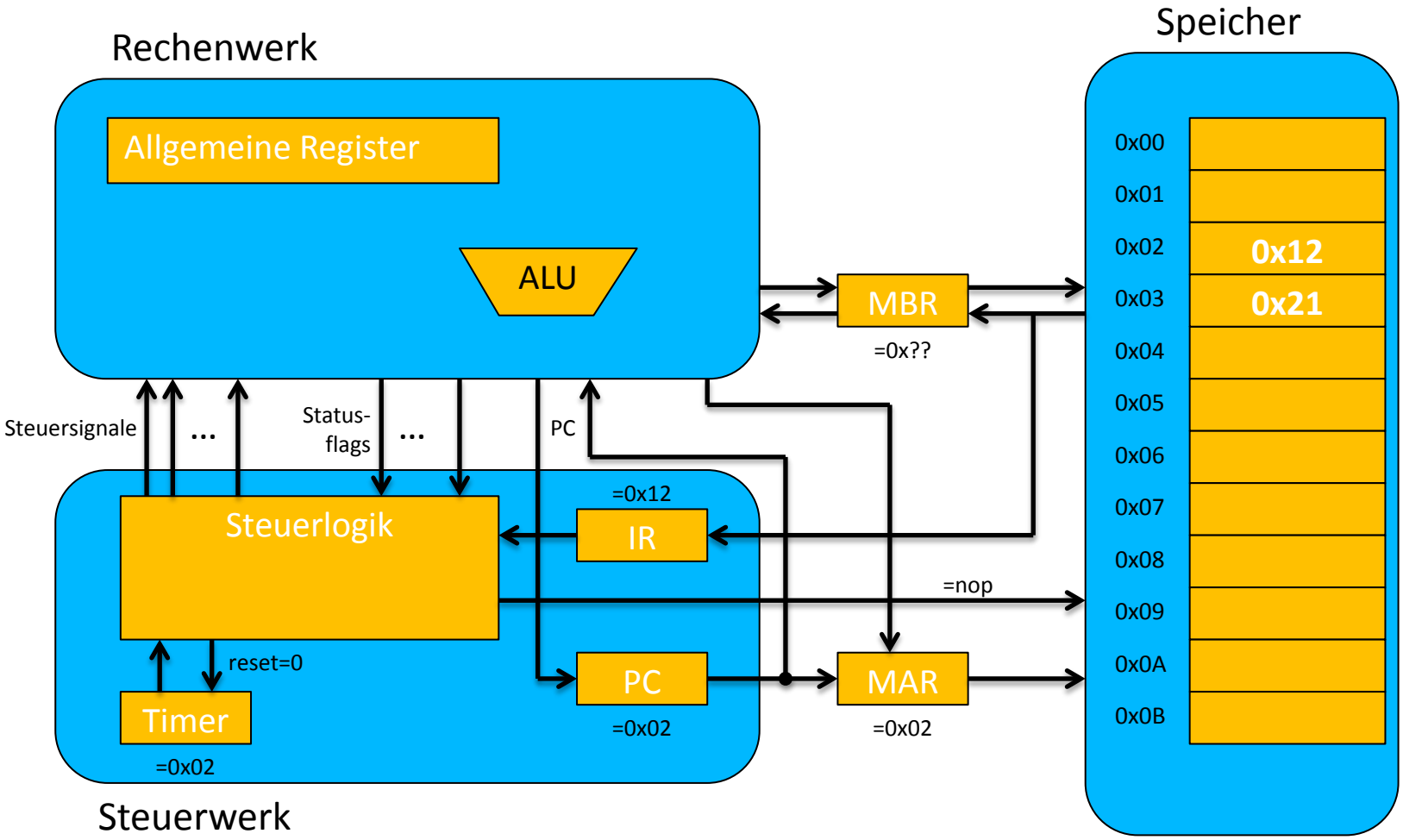
Takt 0: Befehl holen (MAR <= PC)



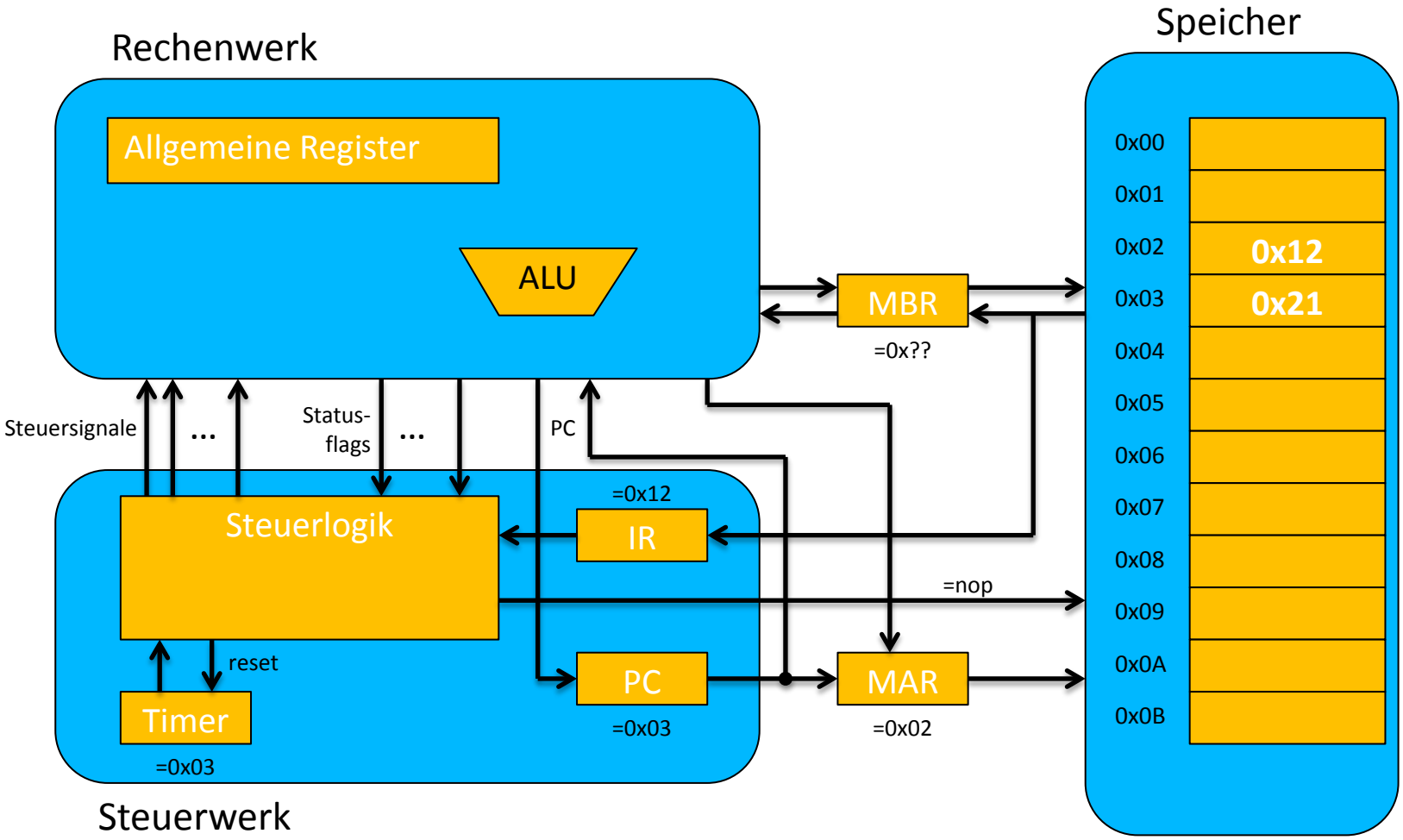
Takt 1: Befehl holen (IR <= MEM[MAR])



Takt 2: Befehl holen (PC <= PC + 1)



Situation zu Beginn von Takt 3



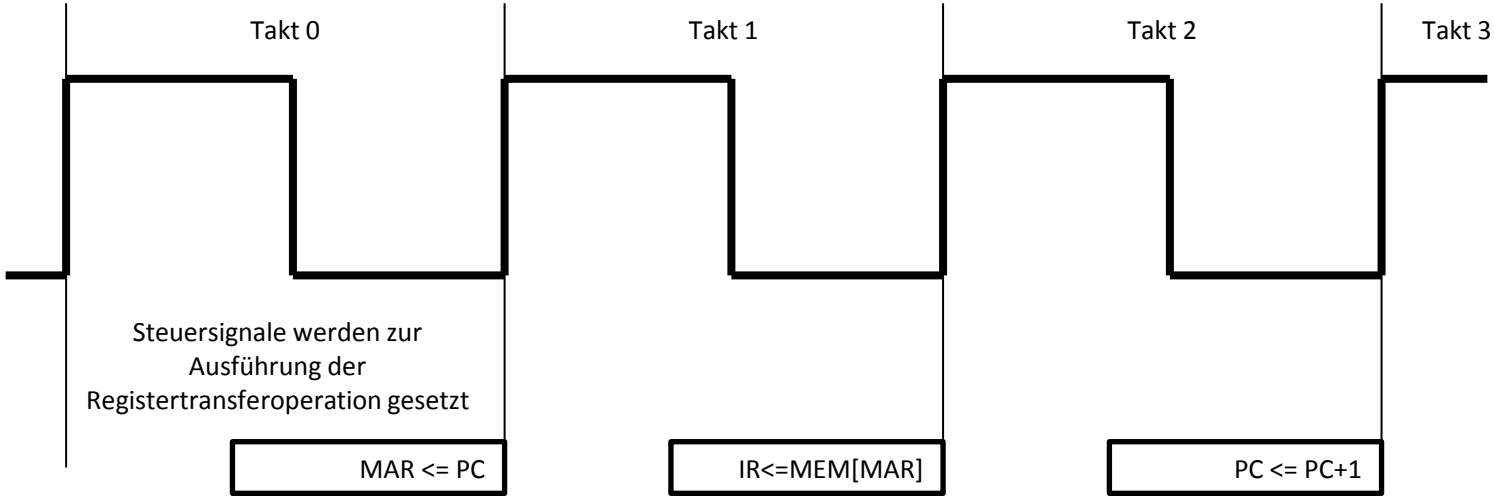
Zusammenfassung Befehlsholphase

Ausgeführte Registertransferoperationen:

- Timer=0: MAR <= PC
- Timer=1: IR <= MEM[MAR]
- Timer=2: PC <= PC + 1

Danach befindet sich Befehlscode im IR

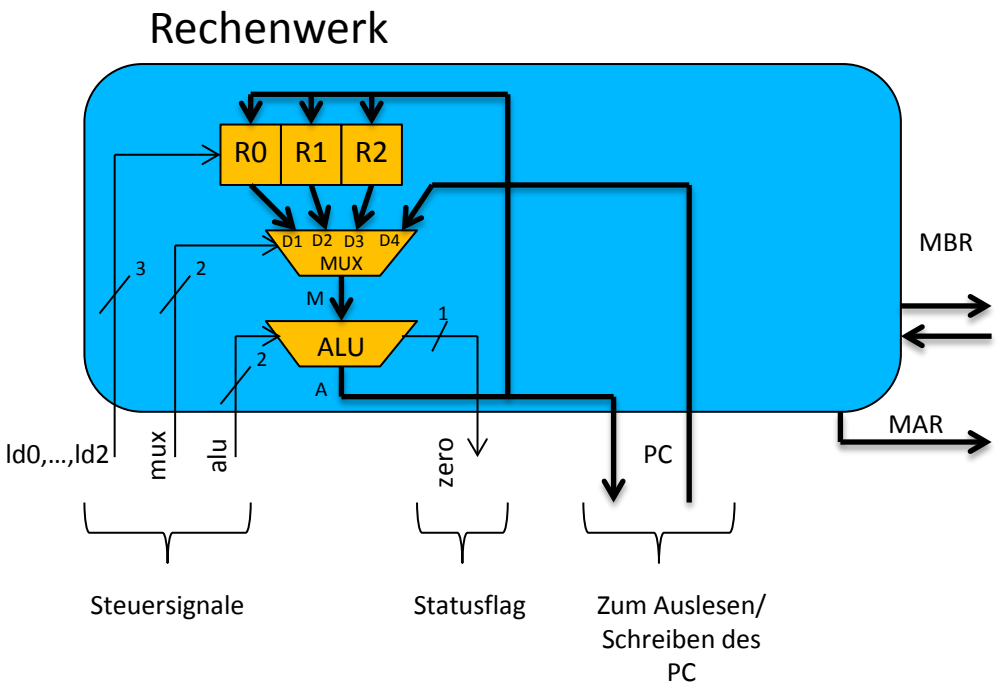
Ab Takt 3 kann die Kontrolllogik abhängig vom Timer und dem Befehlscode den Datenpfad steuern



Im Folgenden Verfeinerung des Rechenwerkes

Details Rechenwerk

Folgendes einfaches Rechenwerk wird verwendet:



Steuersignale legen Verhalten der Komponenten fest

MUX:

Steuersignal *mux*	Ausgang *M*
0	D1
1	D2
2	D3
3	D4

ALU:

Signal *alu*	Ausgang	
	A	*zero*
0 (test)	M	1, falls M=0 0, sonst
1 (+1)	M+1	0
2 (-1)	M-1	0
3 (transfer)	M	0

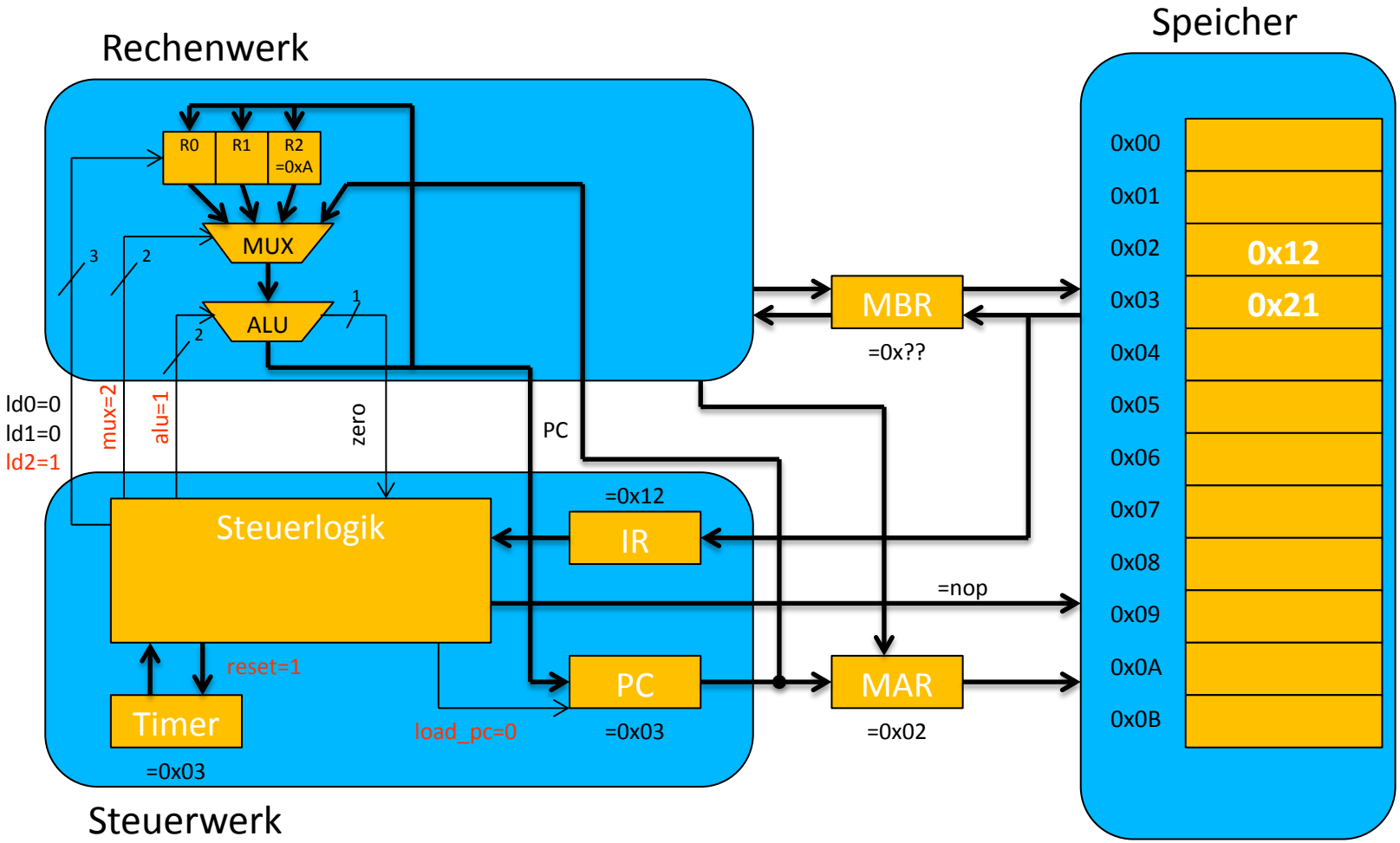
R*i*, mit $0 \leq i \leq 2$:

- R*i*(*t*+1) = R*i*(*t*), falls ld*i* = 0
- R*i*(*t*+1) = A, falls ld*i* = 1

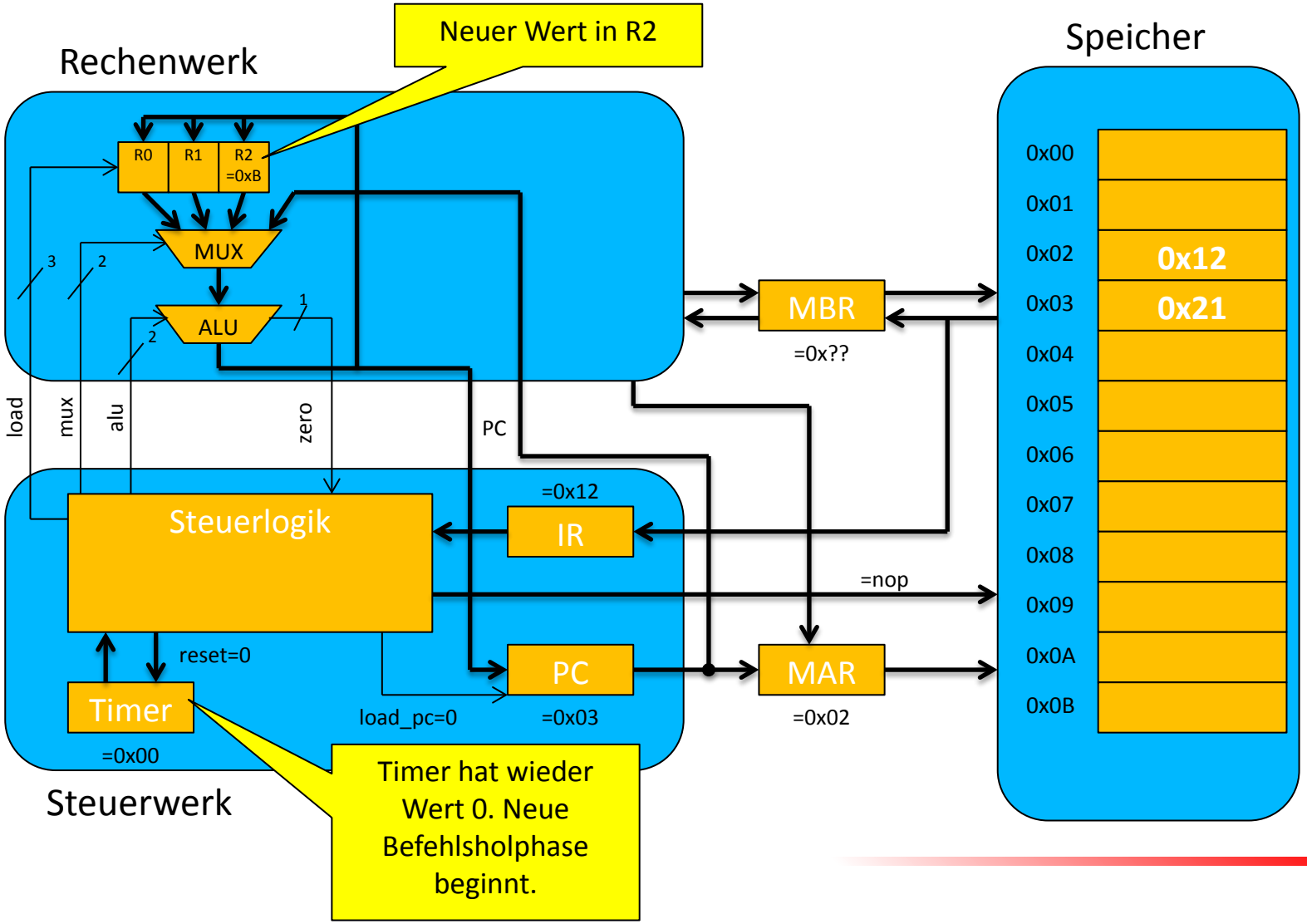
Kodierung für Inkrement/Dekrement

Befehlscode im IR	Bedeutung	Assemblersyntax
0x10	R0 <= R0+1	inc R0
0x11	R1 <= R1+1	inc R1
0x12	R2 <= R2+1	inc R2
0x20	R0 <= R0-1	dec R0
0x21	R1 <= R1-1	dec R1
0x22	R2 <= R2-1	dec R2

Takt 3: Befehl im IR verarbeiten (R2 <= R2 + 1)



Situation zu Beginn von Takt 4 = Takt 0



Zusammenfassung

- Sie sollten mit folgenden Begriffen und Konzepten vertraut sein:
  - Kombinatorische / Sequentielle Logik
  - Verhalten Latch / Flip-Flop
  - Register-, Zählerimplementierung (Feedback)
  - Registertransferbefehle
  - Befehlsverarbeitung von-Neumann-Computer
    - Steuerwerk
    - Rechenwerk