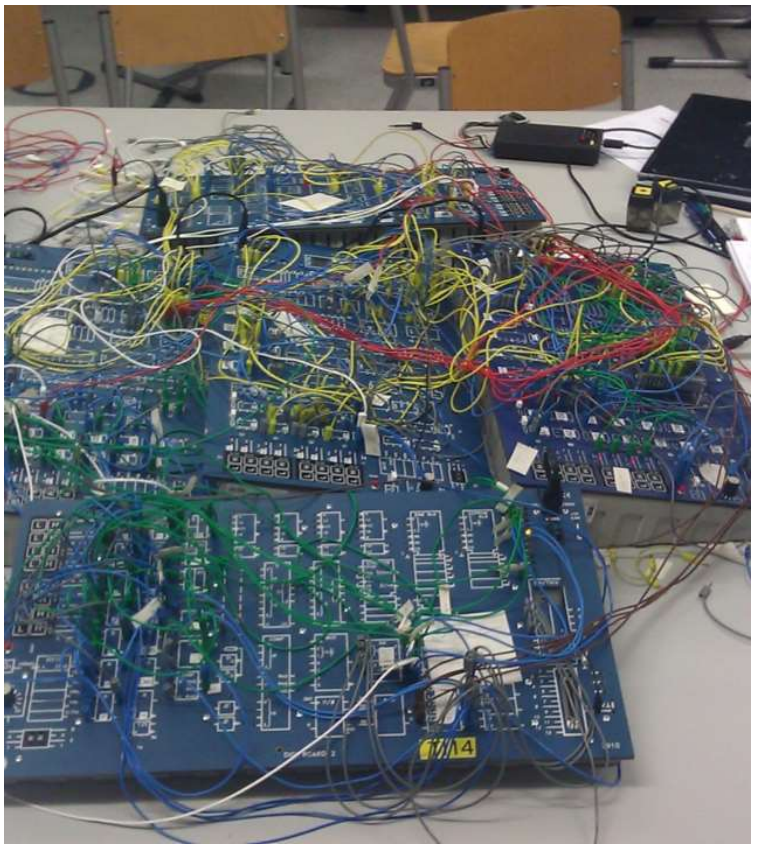
Hochschule
München
University of
Applied Sciences

Prof. Dr. Orehek
Prof. Dr. Wallentowitz
Prof. Dr. Zugenmaier
Prof. Dr. Henrici

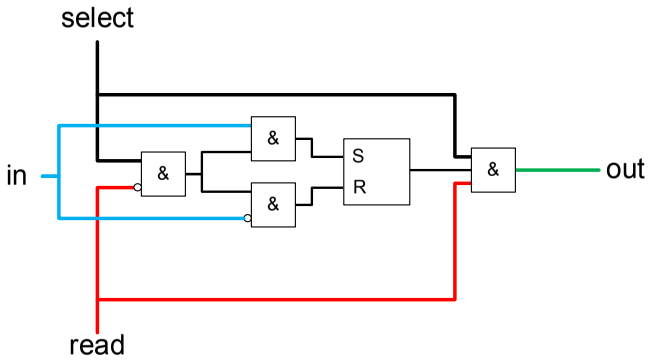
Technische Informatik I

Abbildungen aus:
Grundlagen der Technischen Informatik
Dirk W. Hoffmann, Hanser Verlag

HM

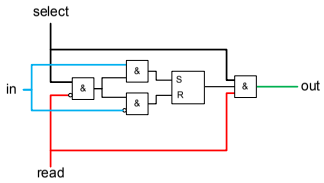


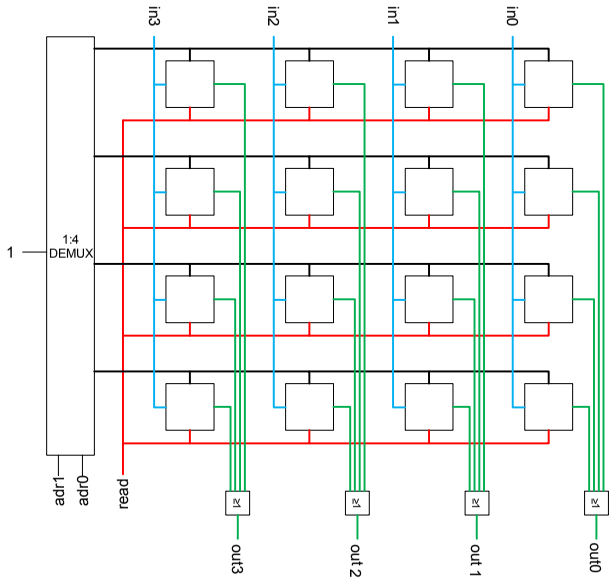
Speicherzelle



HM

Speicher

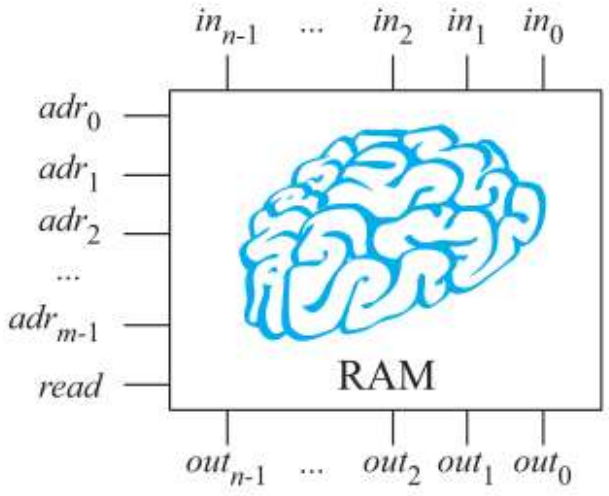

select
in
&
&
&
S
R
&
out
read


in3
in2
in1
in0
1
1:4
DEMUX
adr1
adr0
read
out3
out 2
out 1
out0

HM

Hauptspeicher

- Allgemeines Schema



Mikroprozessor: Von-Neumann-Architektur

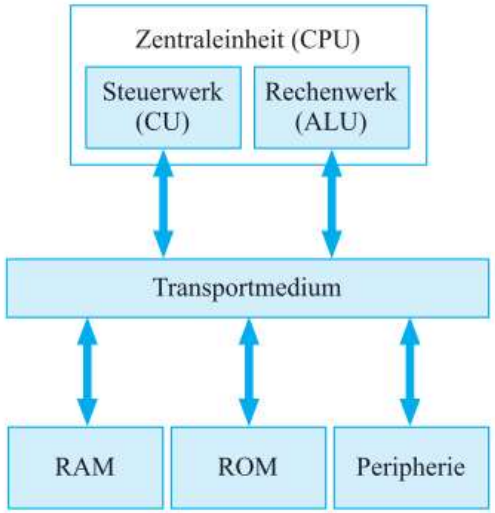
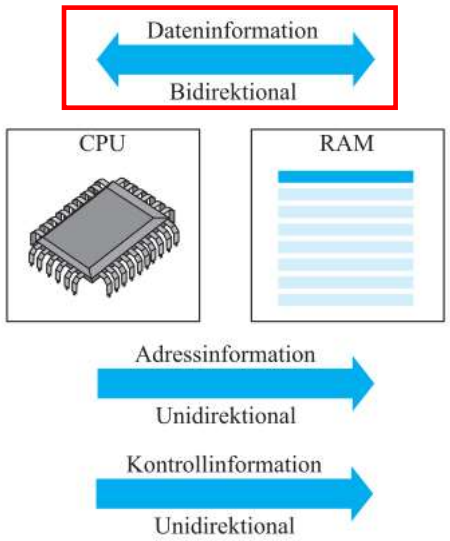
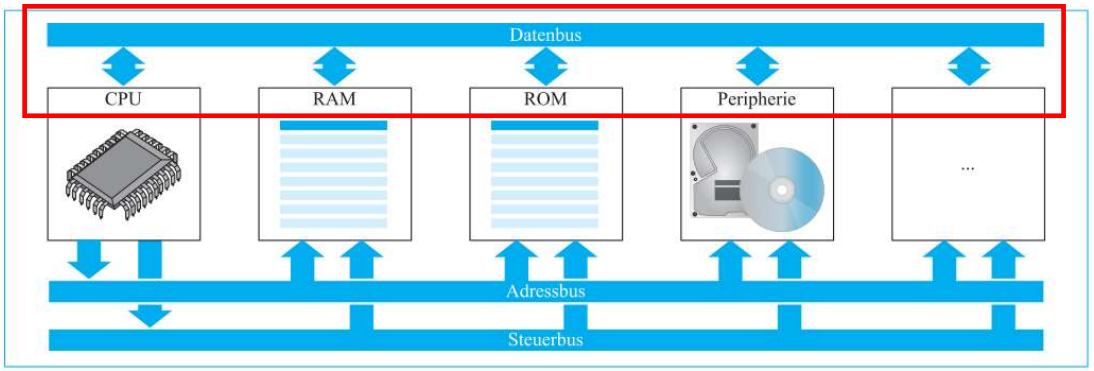


Abbildung 11.1: Aufbau eines Rechners nach dem Von-Neumann-Prinzip

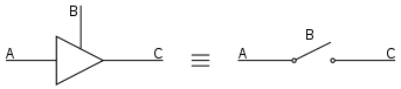
CPU <-> Speicher



Typischer Aufbau eines prozessorgesteuerten Mikrorechners



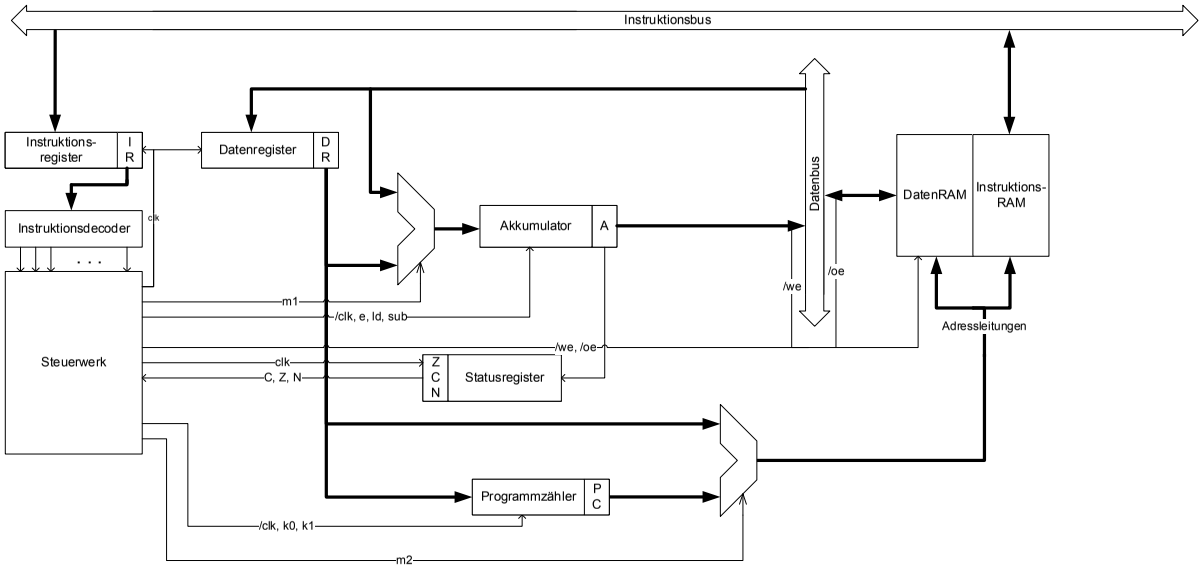
- Tri-State [0,1,Z] mit Z als hochohmiger Zustand



Befehlssatz

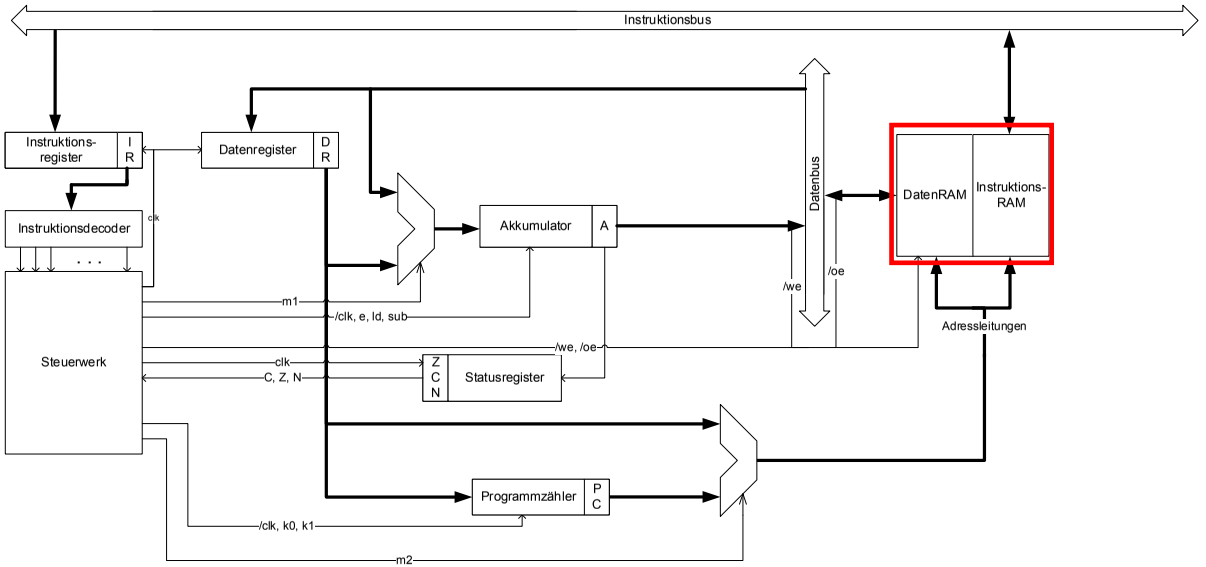
Nr	Befehl	Codierung				Beschreibung
0	`NOP`	0	0	0	0	Wartezyklus (*No Operation*)
						Lade- und Speicherbefehle
1	`LDA #n`	0	0	0	1	Lädt den Akkumulator mit dem Wert *n*
2	`LDA (n)`	0	0	1	0	Lädt den Akkumulator mit dem Inhalt der Speicherstelle *n*
3	`STA n`	0	0	1	1	Überträgt den Akkumulatorinhalt in die Speicherstelle *n*
						Arithmetikbefehle
4	`ADD #n`	0	1	0	0	Erhöht den Akkumulatorinhalt um den Wert *n*
5	`ADD (n)`	0	1	0	1	Erhöht den Akkumulatorinhalt um den Inhalt der Speicherstelle *n*
6	`SUB #n`	0	1	1	0	Erniedrigt den Akkumulatorinhalt um den Wert *n*
7	`SUB (n)`	0	1	1	1	Erniedrigt den Akkumulatorinhalt um den Inhalt der Speicherstelle *n*
						Sprungbefehle
8	`JMP n`	1	0	0	0	Lädt den Instruktionszähler mit dem Wert *n*
9	`BRZ #n`	1	0	0	1	Addiert *n* auf den Instruktionszähler, falls das Zero-Bit gesetzt ist
10	`BRC #n`	1	0	1	0	Addiert *n* auf den Instruktionszähler, falls das Carry-Bit gesetzt ist
11	`BRN #n`	1	0	1	1	Addiert *n* auf den Instruktionszähler, falls das Negations-Bit gesetzt ist

Übersicht



HM

Blockschaltbild



HM

Speicherzellenbelegung

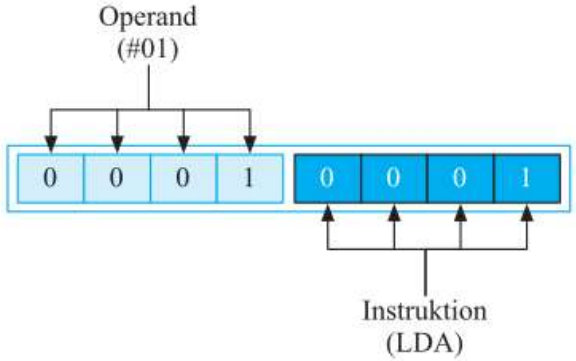
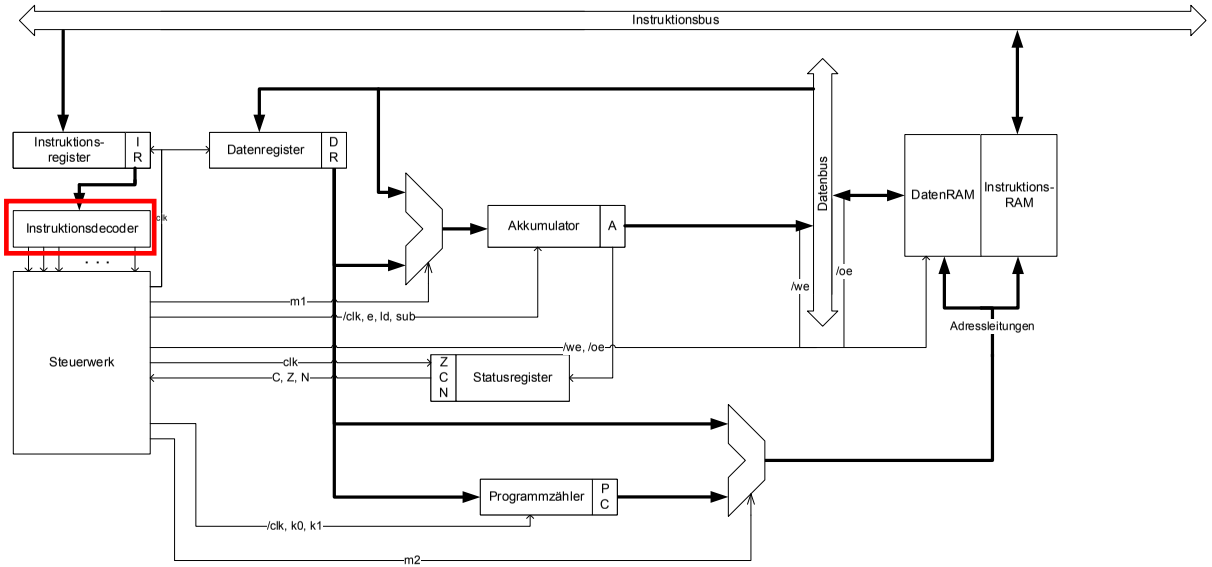


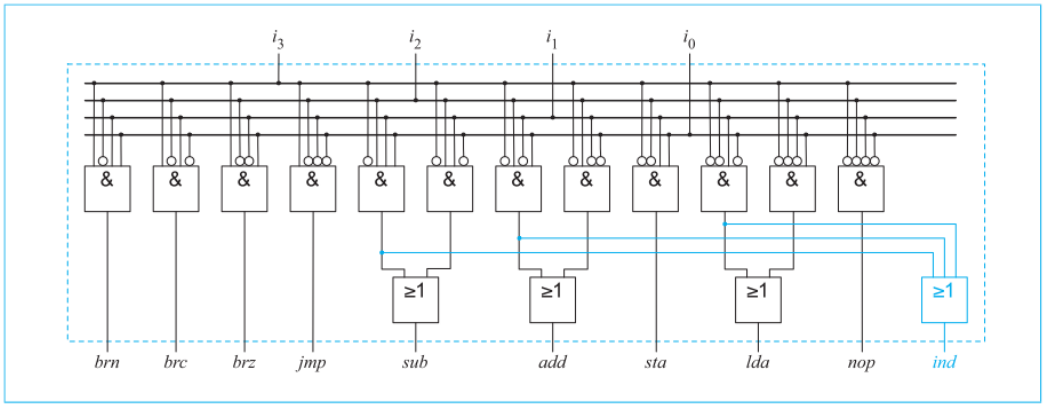
Abbildung 11.8: Das Instruktionsformat unseres Modellprozessors. Die oberen 4 Bit codieren den Operanden, die unteren 4 Bit den auszuführenden Befehl.

Blockschaltbild

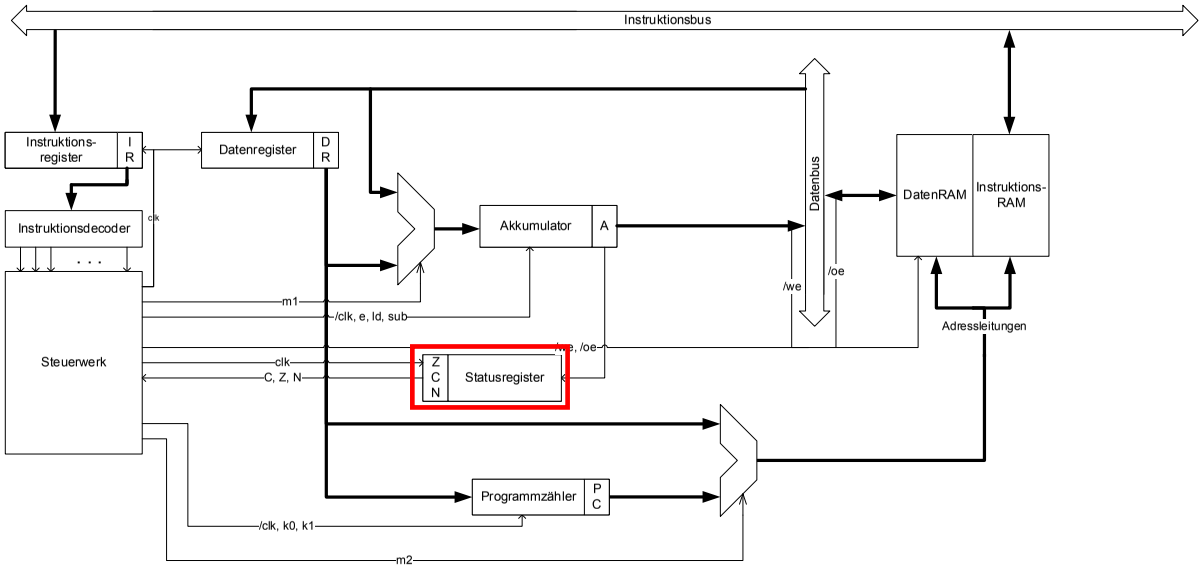


HM

Instruktionsdecoder



Blockschaltbild



HM

Statusregister

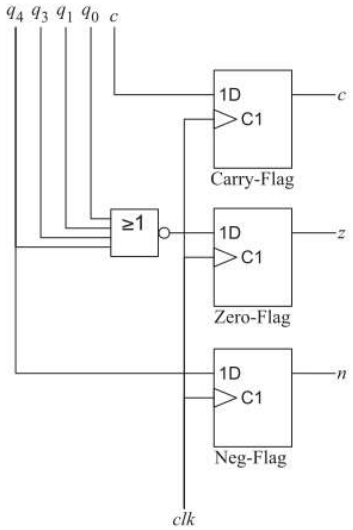
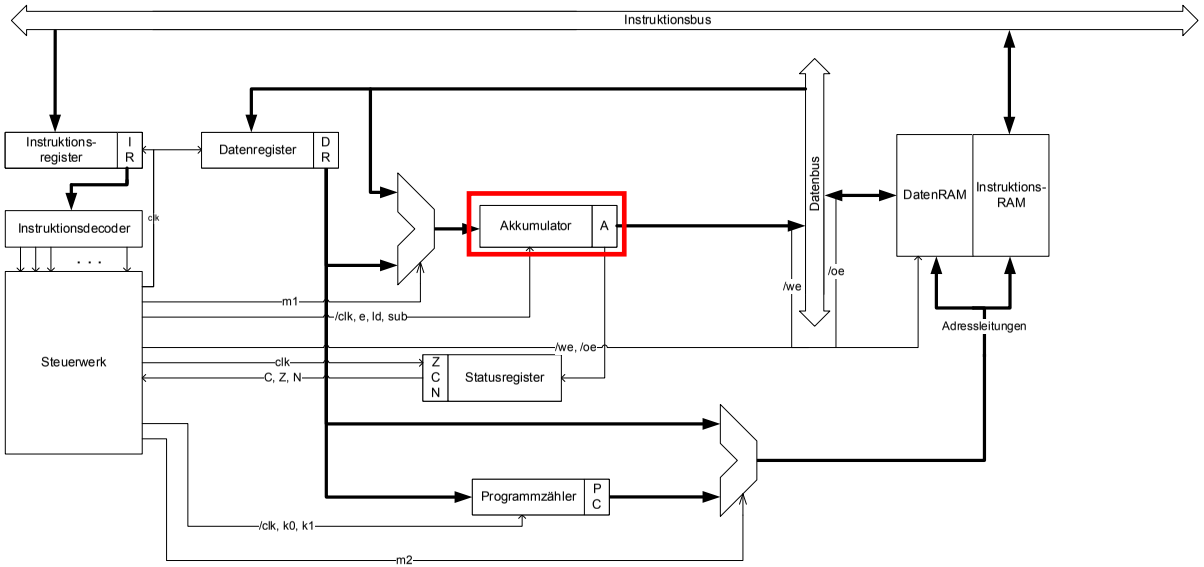


Abbildung 11.15: Das Statusregister unseres Modellrechners

Blockschaltbild



HM

Akkumulatorregister

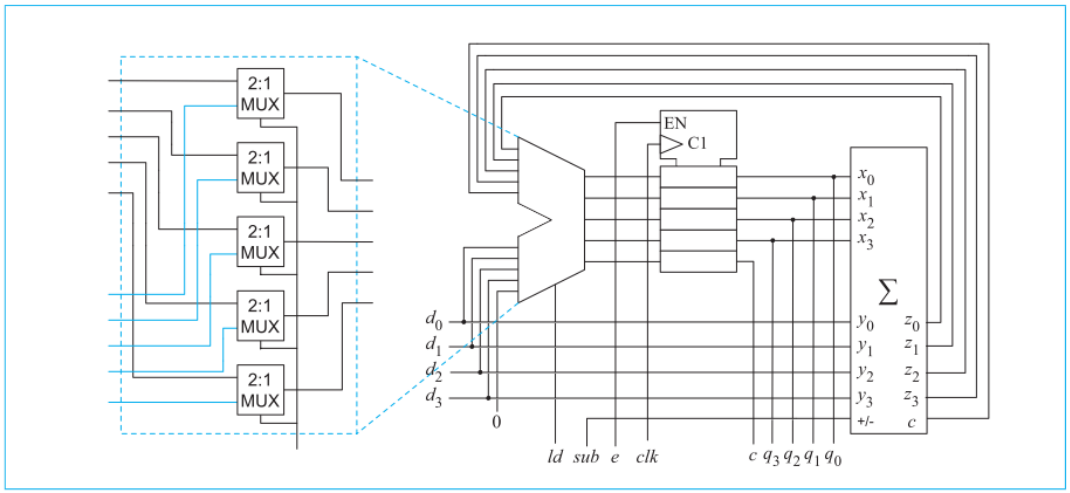
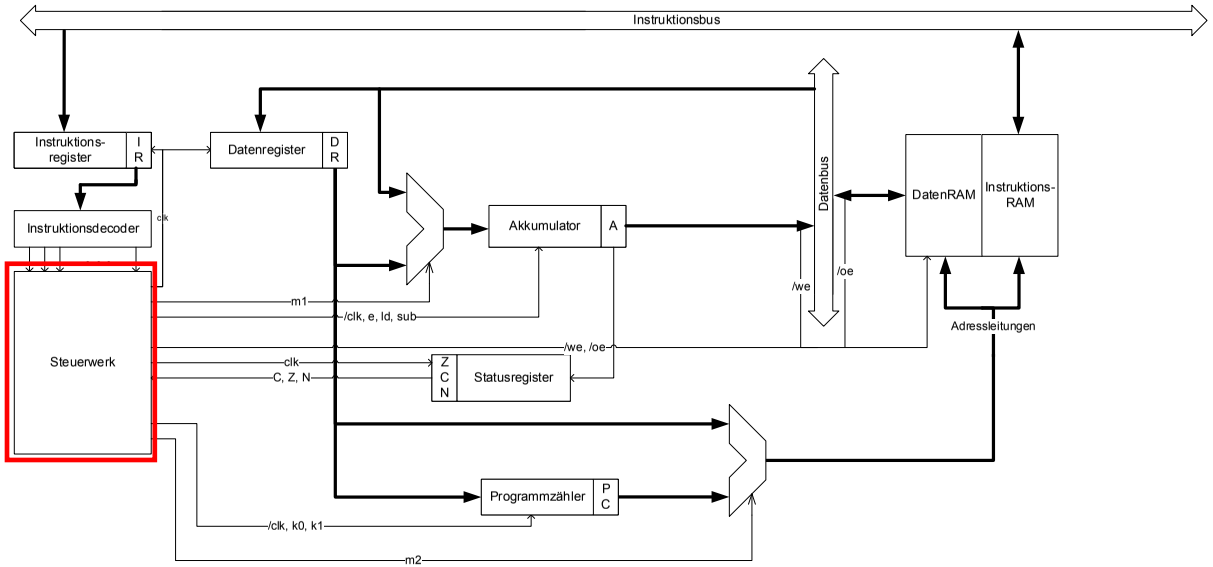


Abbildung 11.16: Das Akkumulatorregister unseres Modellrechners

Blockschaltbild



HM

Aufgabe des Steuerwerks

	Statusvariablen			Akkumulator			PC		RAM	Multiplexer	
Befehl	z	c	n	e	ld	sub	s_1	s_0	we	m_1	m_2
				Fetch: $clk = 0$							
–	–	–	–	0	–	–	1	0	0	–	1
				Decode + Execute + Write: $clk = 1$							
NOP	–	–	–	0	–	–	1	0	0	–	–
LDA	–	–	–	1	1	–	1	0	0	$\neg ind$	0
STA	–	–	–	0	–	–	1	0	1	–	0
ADD	–	–	–	1	0	0	1	0	0	$\neg ind$	0
SUB	–	–	–	1	0	1	1	0	0	$\neg ind$	0
JMP	–	–	–	0	–	–	0	1	0	–	–
BRZ	0	–	–	0	–	–	1	0	0	–	–
BRZ	1	–	–	0	–	–	0	0	0	–	–
BRC	–	0	–	0	–	–	1	0	0	–	–
BRC	–	1	–	0	–	–	0	0	0	–	–
BRN	–	–	0	0	–	–	1	0	0	–	–
BRN	–	–	1	0	–	–	0	0	0	–	–

Tabelle 11.3: Beschaltung von Akkumulator, Instruktionszähler, RAM und Multiplexer durch das Steuerwerk

Steuerwerk

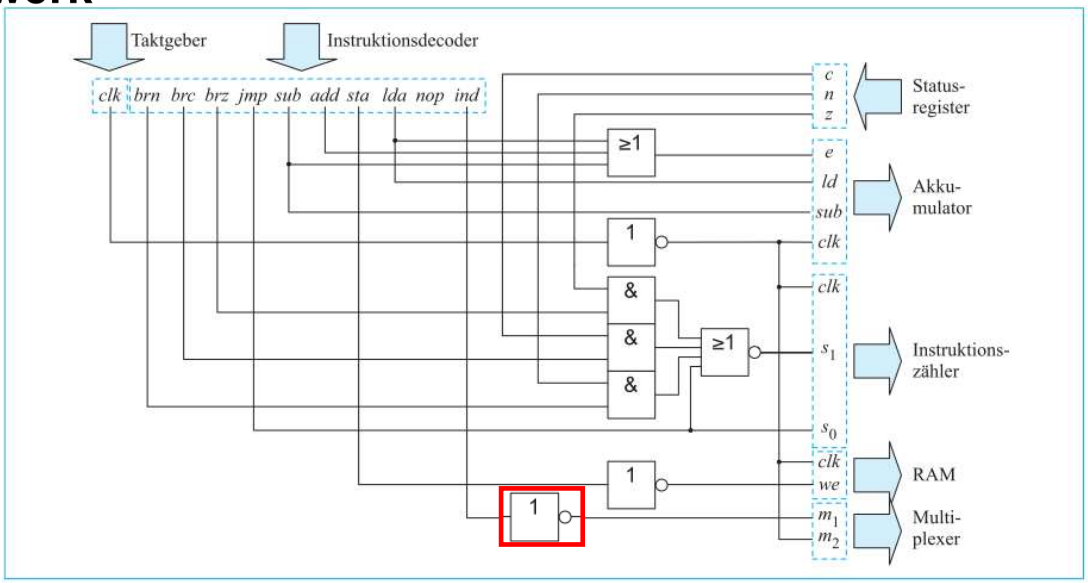
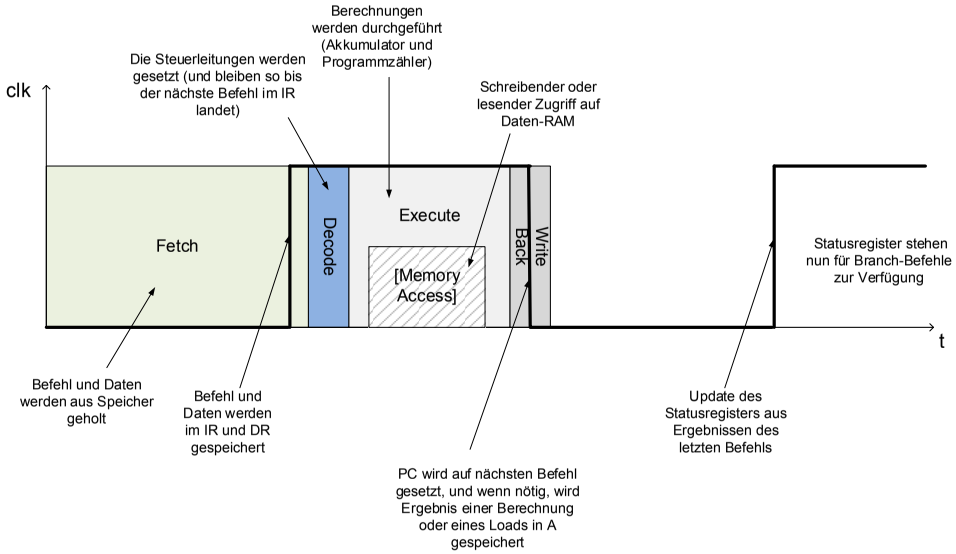
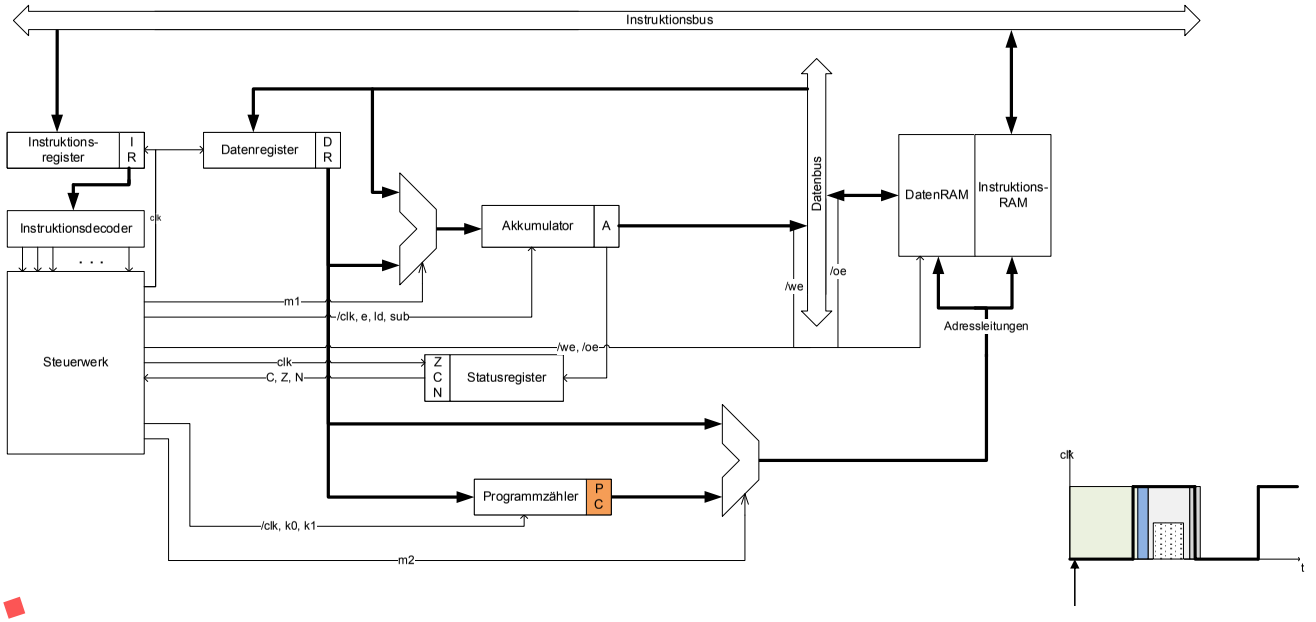


Abbildung 11.17: Vollständig implementiertes Steuerwerk unserer Modell-CPU

5 Phasen

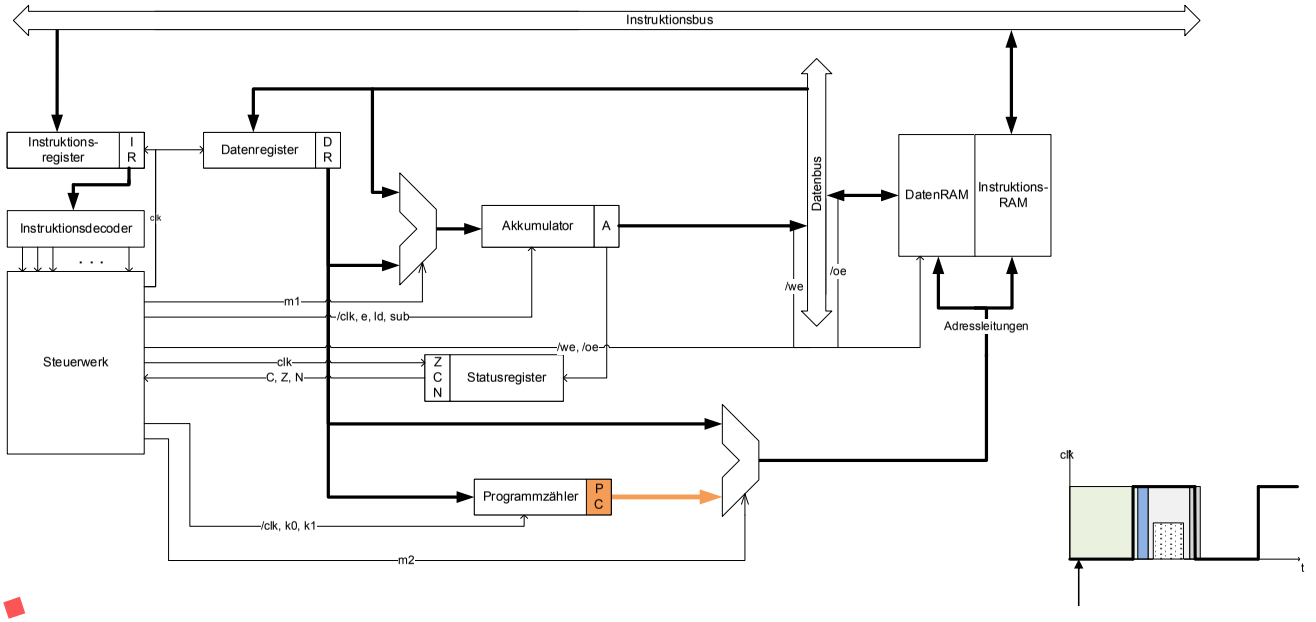

clk
Die Steuerleitungen werden gesetzt (und bleiben so bis der nächste Befehl im IR landet)
Berechnungen werden durchgeführt (Akkumulator und Programmzähler)
Schreibender oder lesender Zugriff auf Daten-RAM
Fetch
Decode
Execute
[Memory Access]
Back
Write
Statusregister stehen nun für Branch-Befehle zur Verfügung
t
Befehl und Daten werden aus Speicher geholt
Befehl und Daten werden im IR und DR gespeichert
PC wird auf nächsten Befehl gesetzt, und wenn nötig, wird Ergebnis einer Berechnung oder eines Loads in A gespeichert
Update des Statusregisters aus Ergebnissen des letzten Befehls

Fetch-Phase



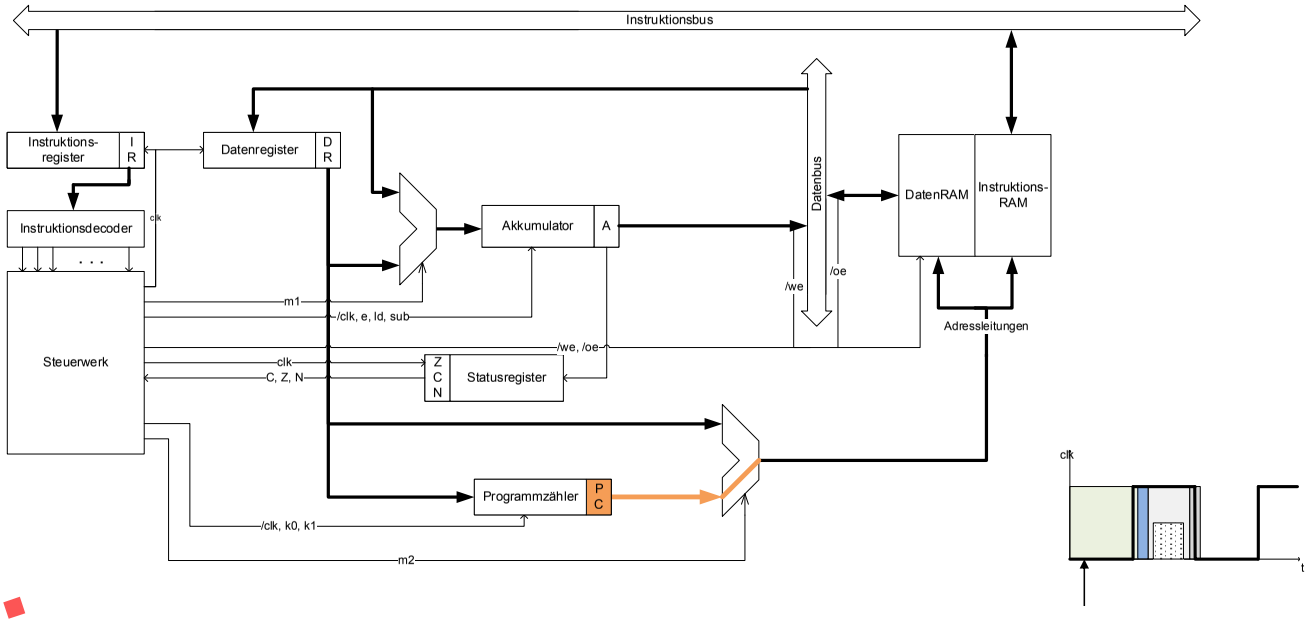
HM

Fetch-Phase



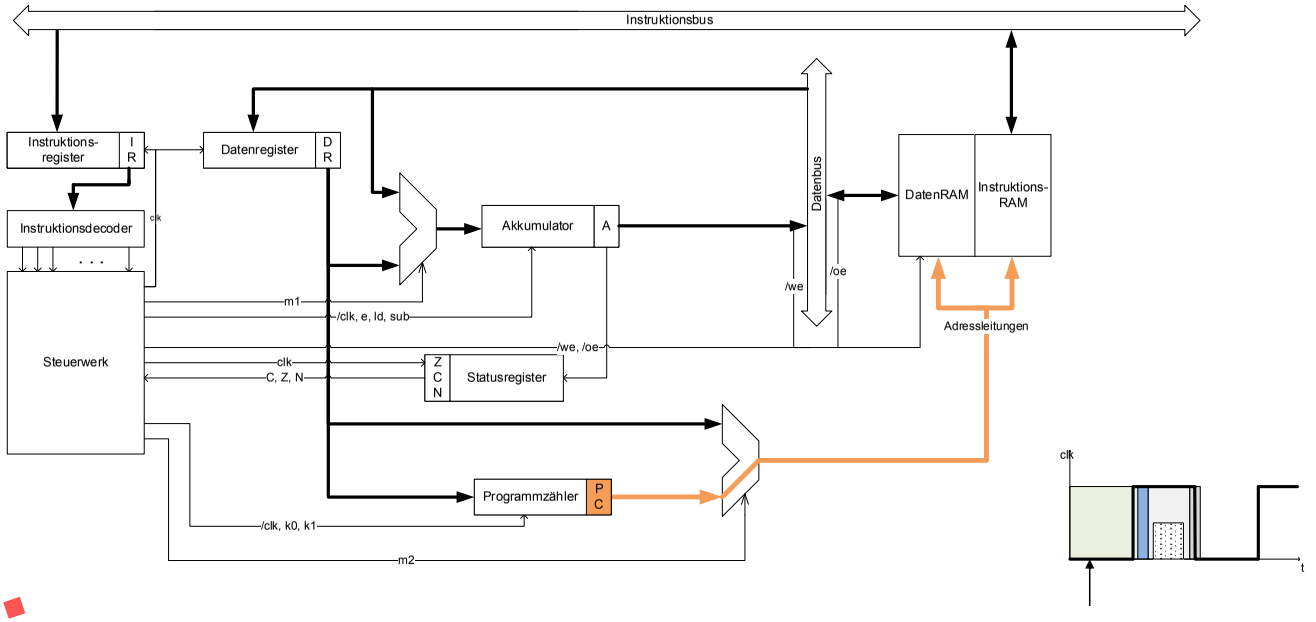
HM

Fetch-Phase



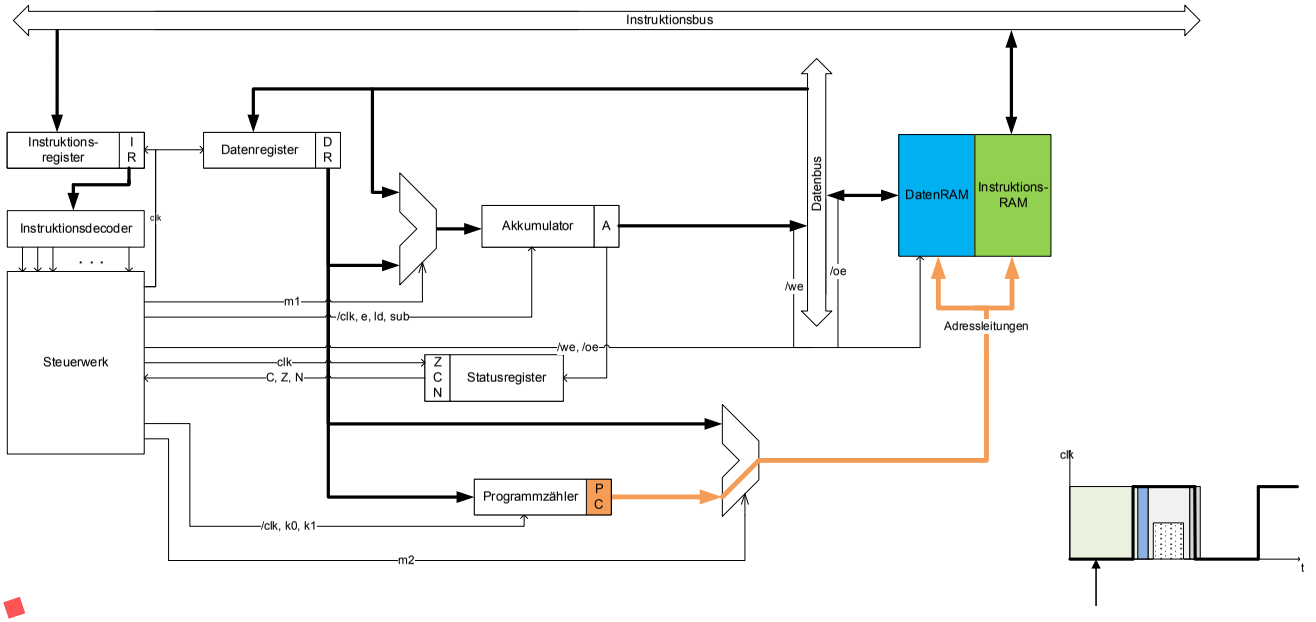
HM

Fetch-Phase



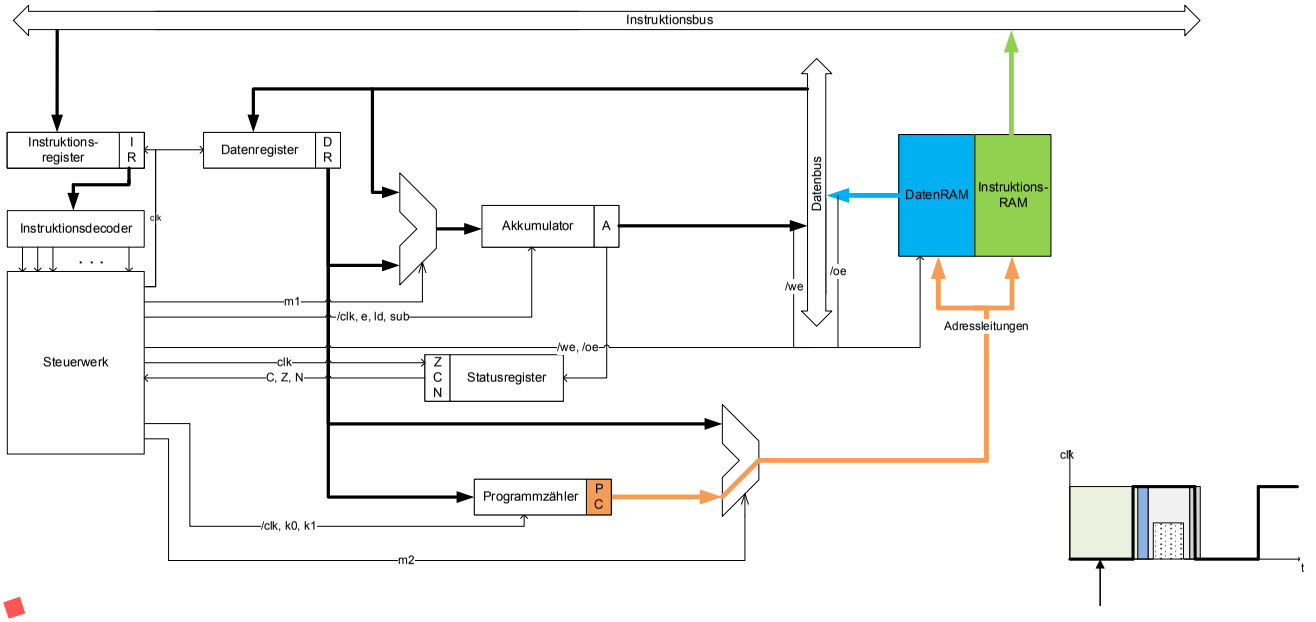
HM

Fetch-Phase



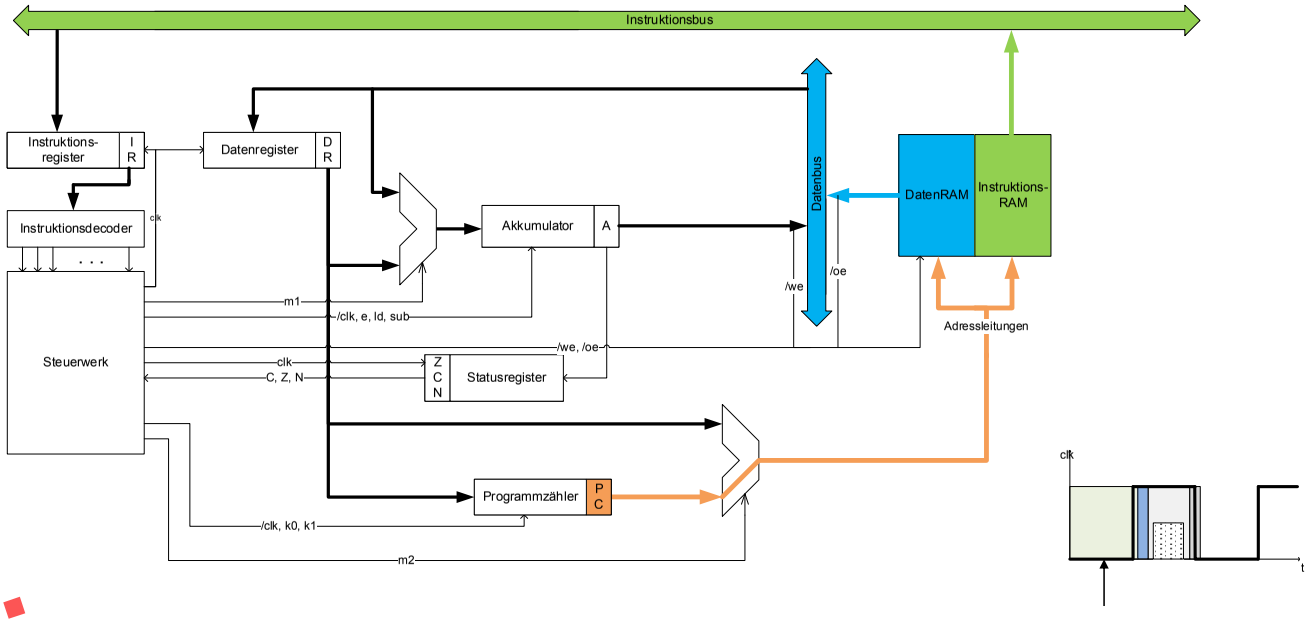
HM

Fetch-Phase



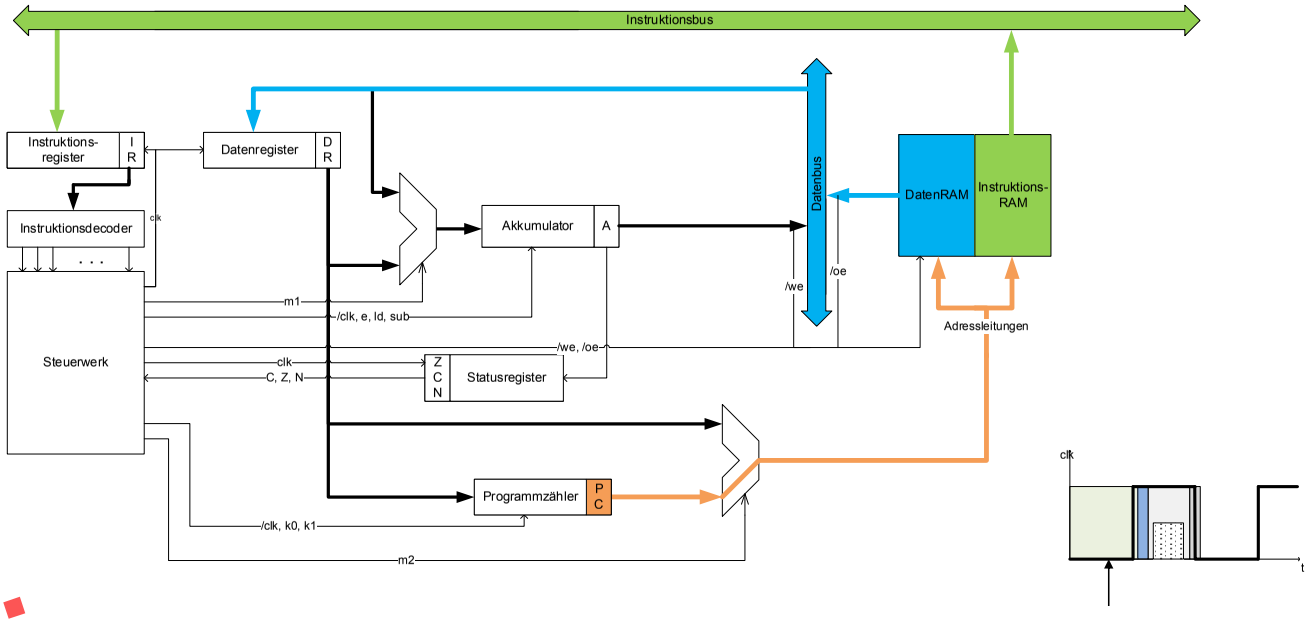
HM

Fetch-Phase



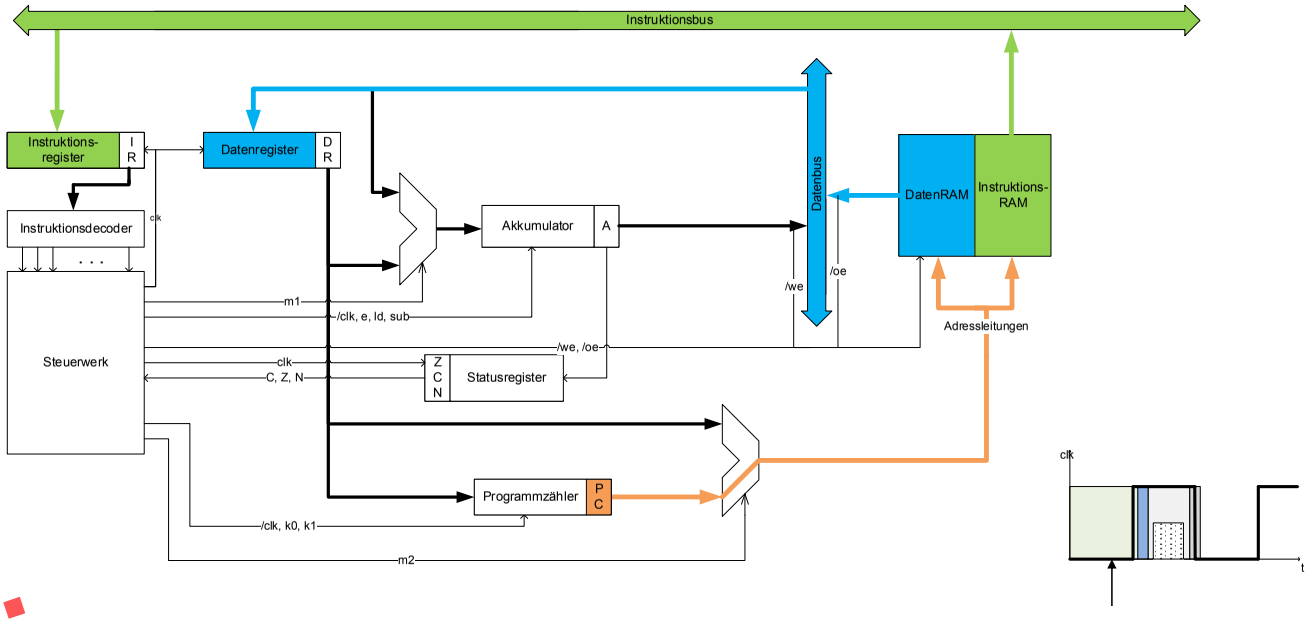
HM

Fetch-Phase



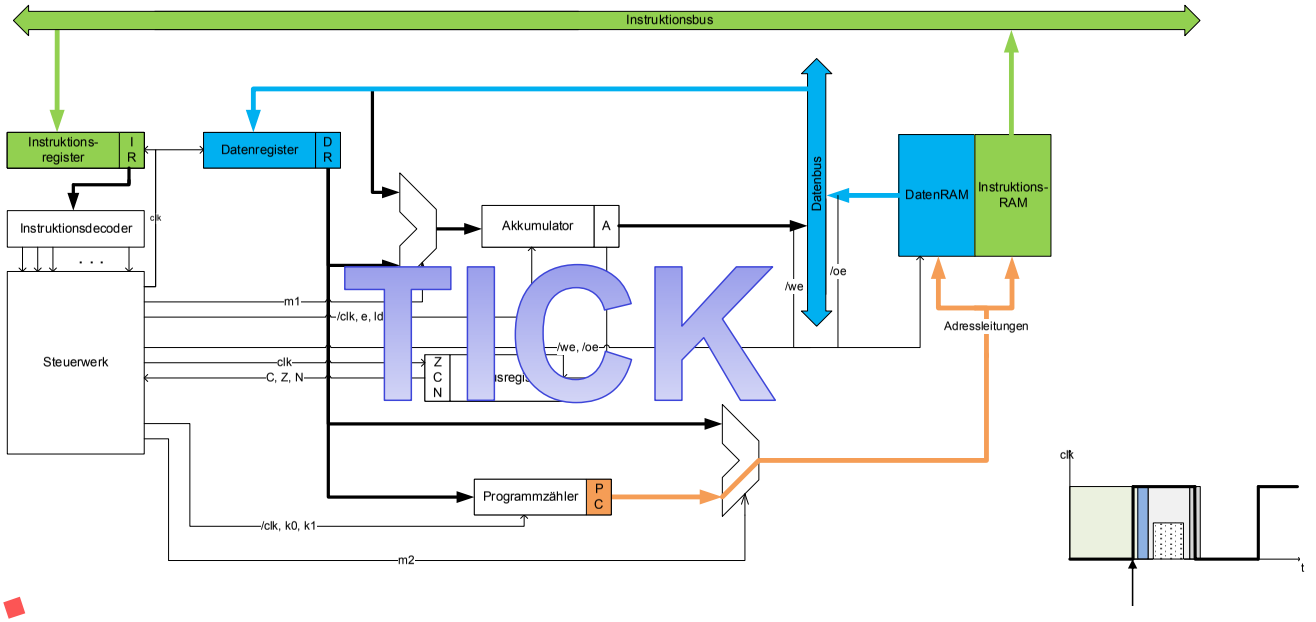
HM

Fetch-Phase



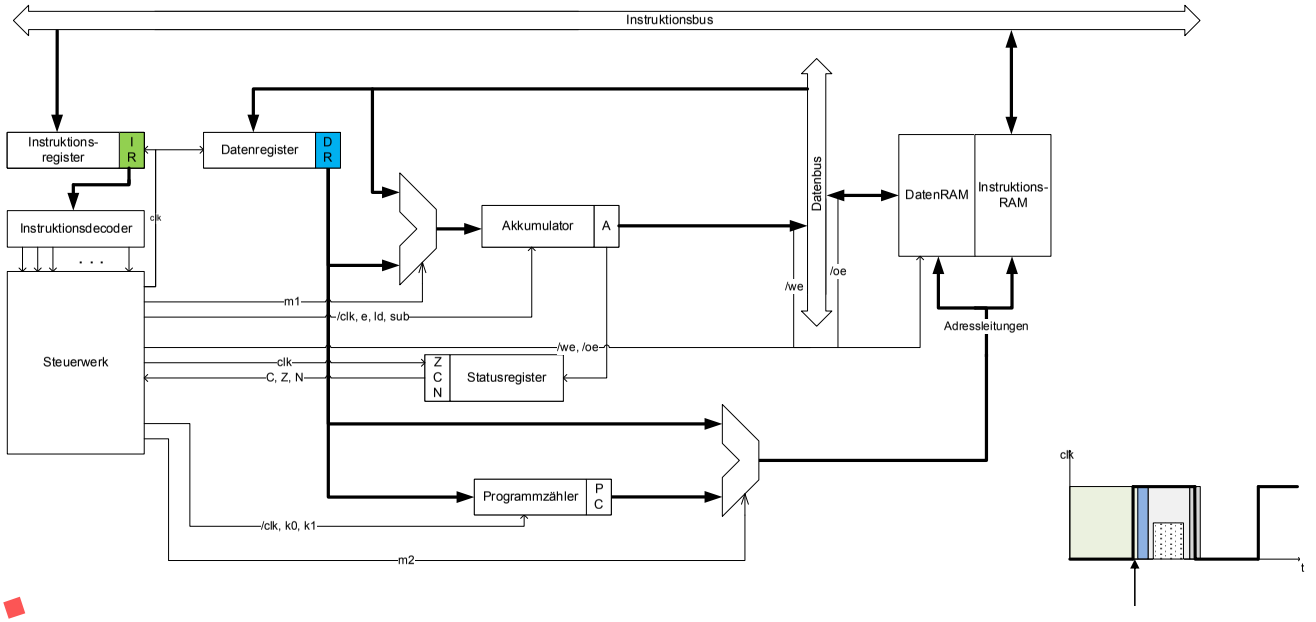
HM

Fetch-Phase



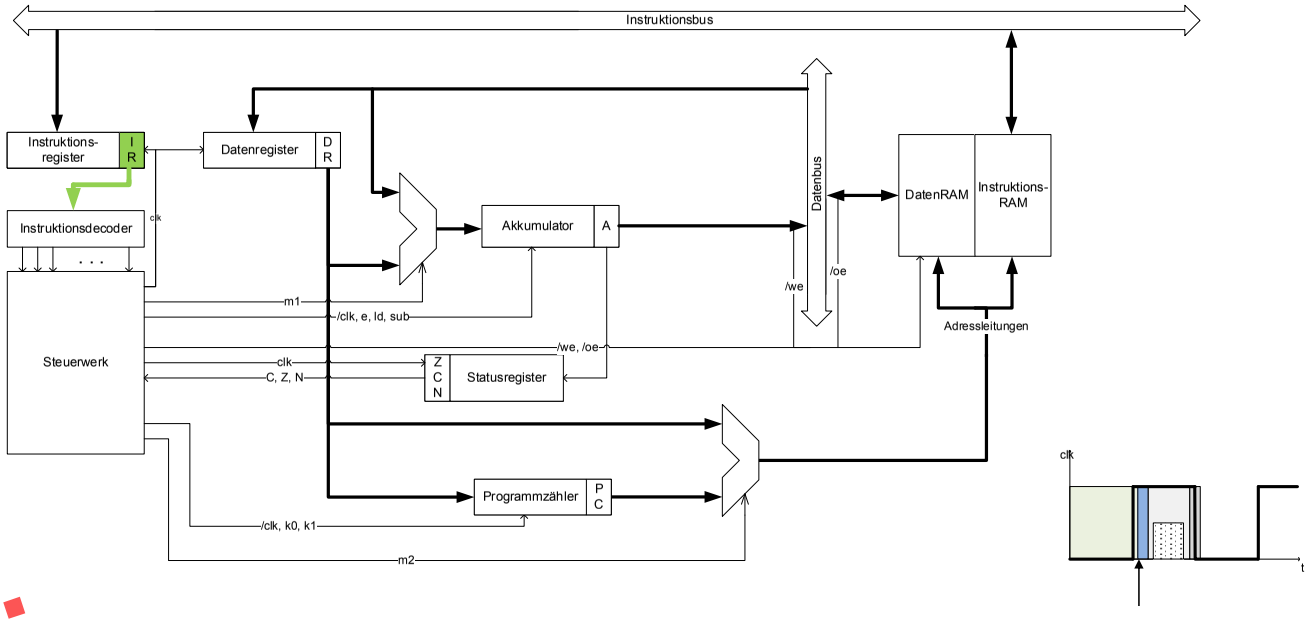
HM

Fetch-Phase



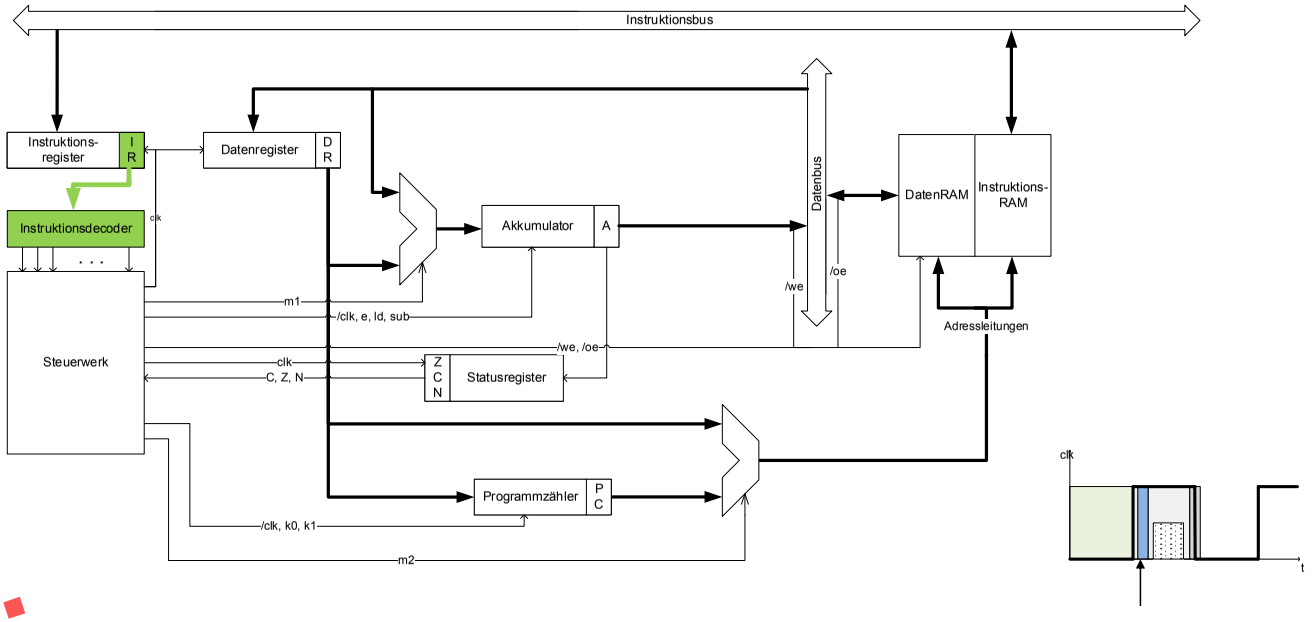
HM

Decode-Phase



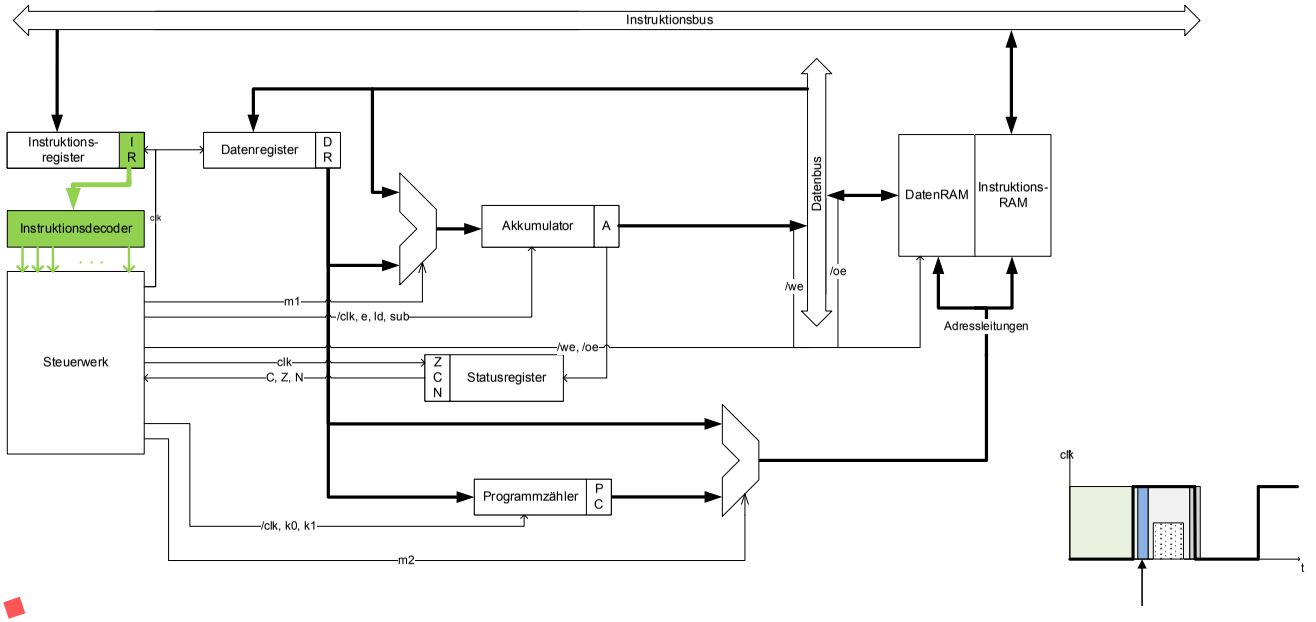
HM

Decode-Phase



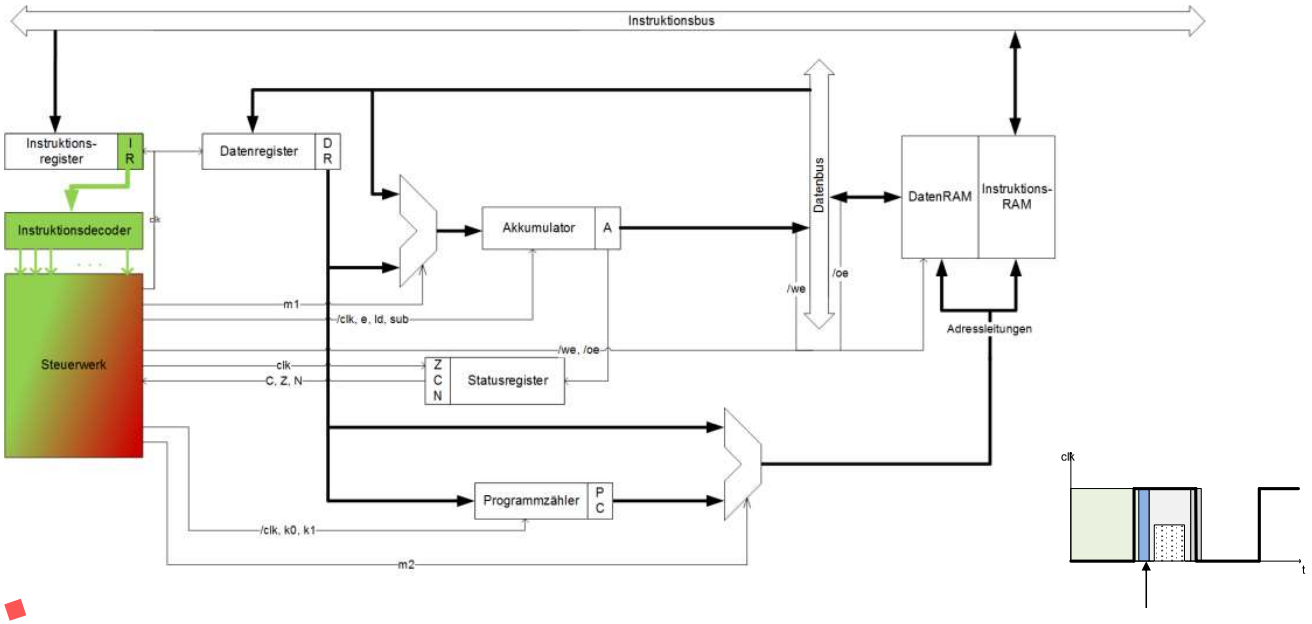
HM

Decode-Phase



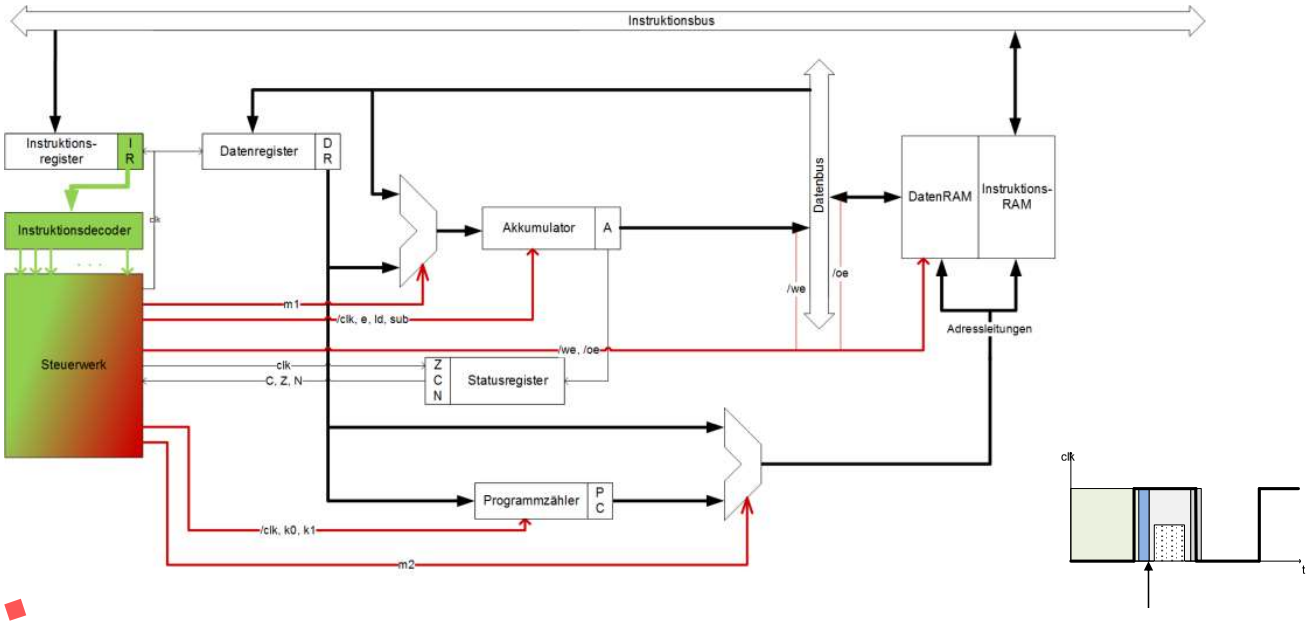
HM

Decode-Phase



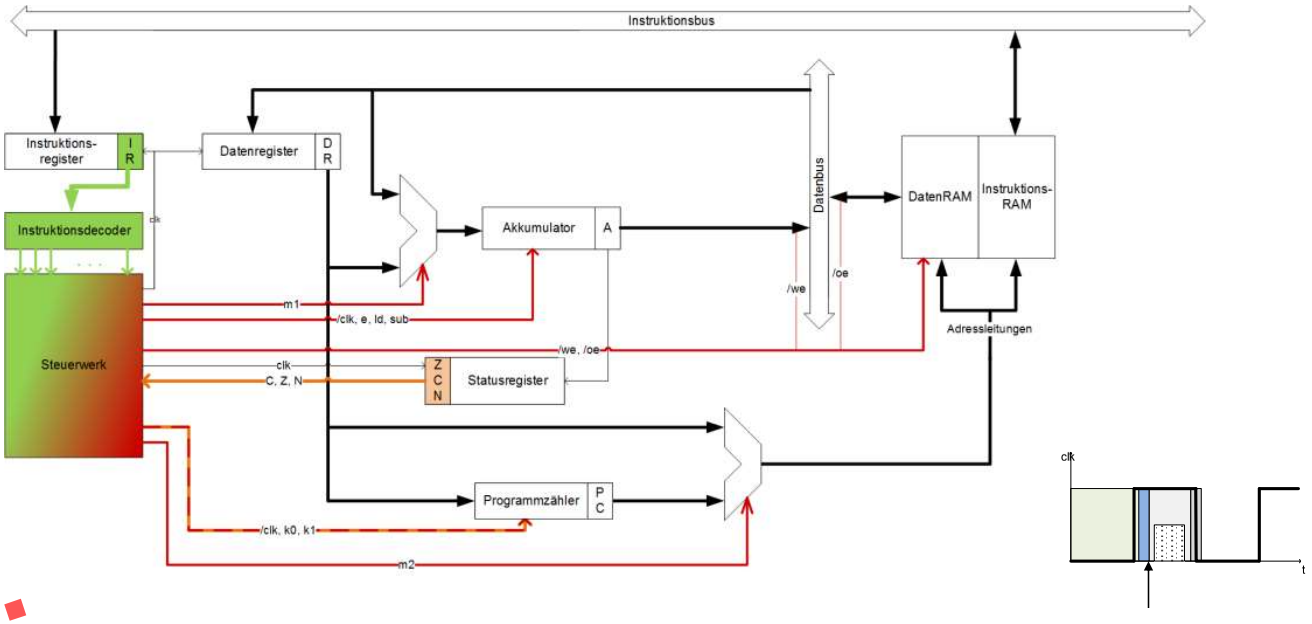
HM

Decode-Phase



HM

Decode-Phase für bedingte Sprünge `BRx #n`

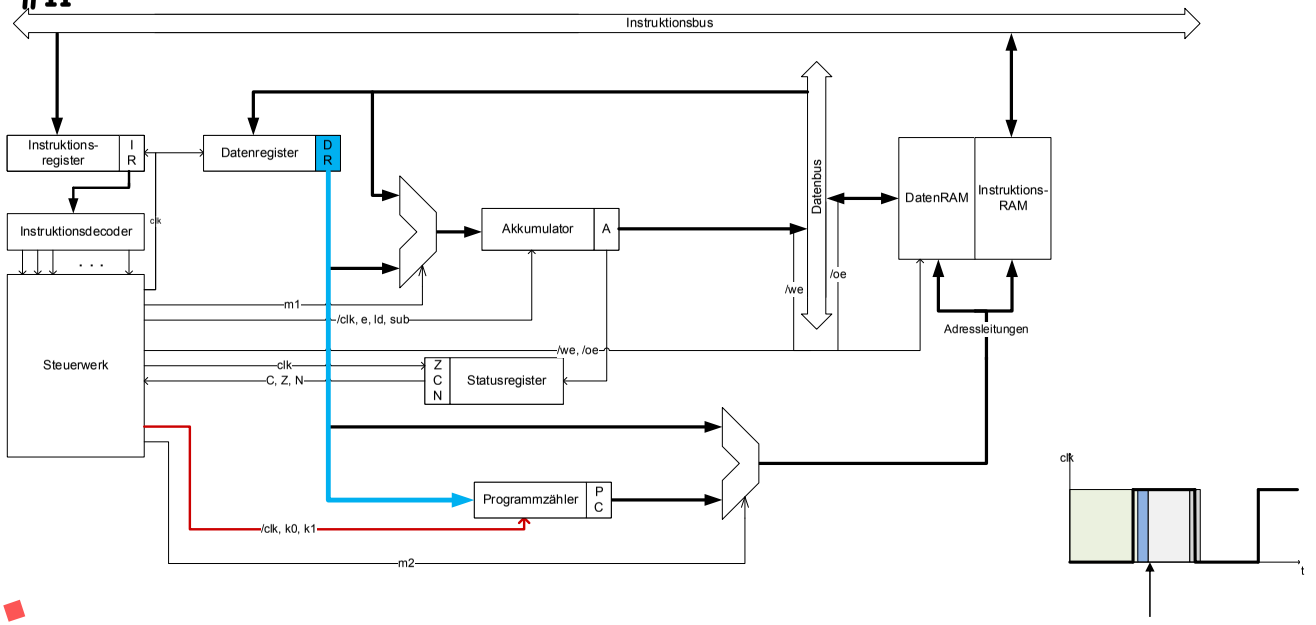


HM

Execute Phase für Sprünge

BRx #n

JMP #n

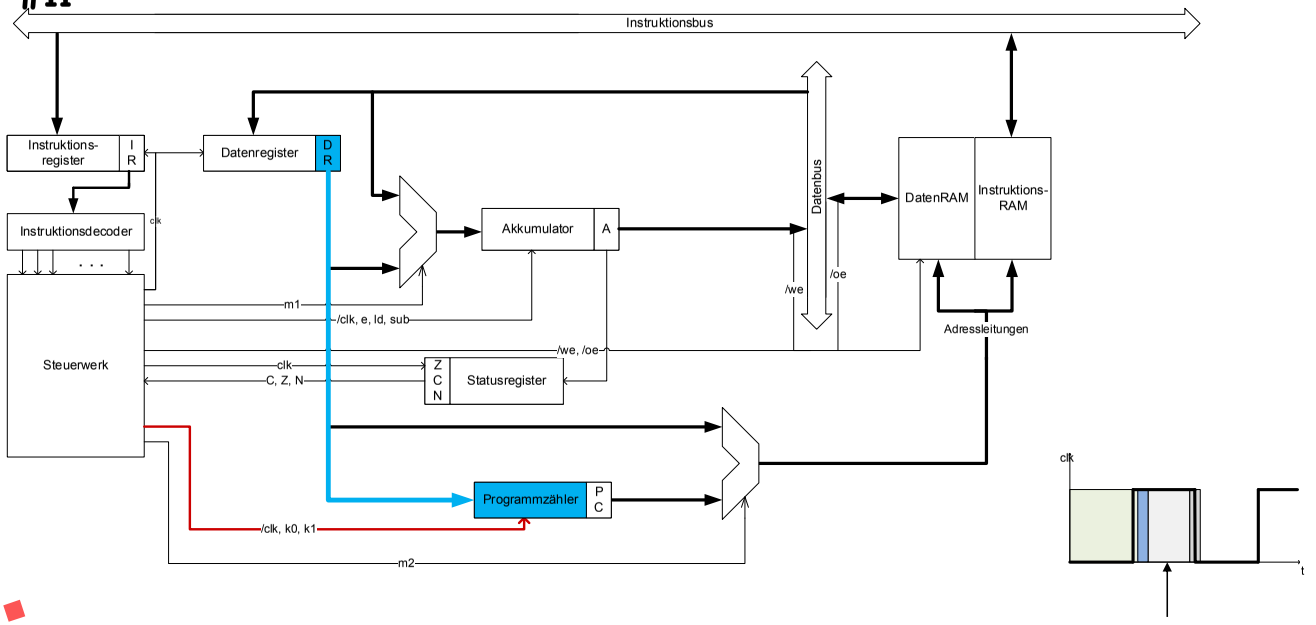


HM

Execute Phase für Sprünge

BRx #n

JMP #n

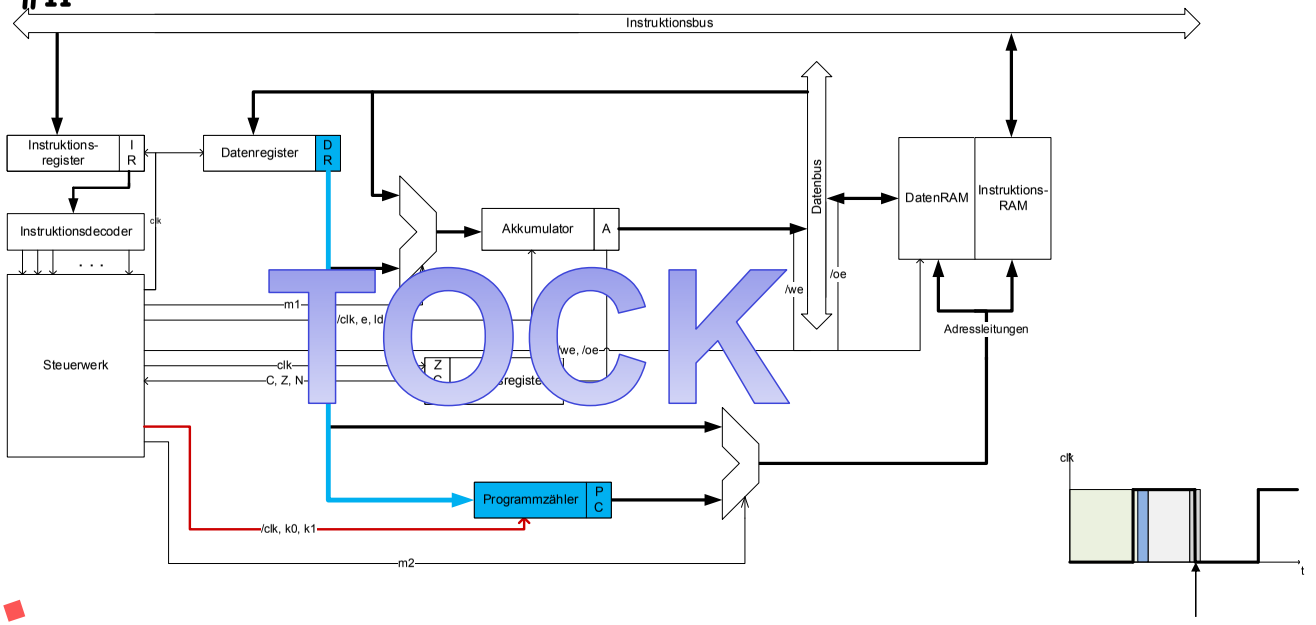


HM

Execute Phase für Sprünge

BRx #n

JMP #n

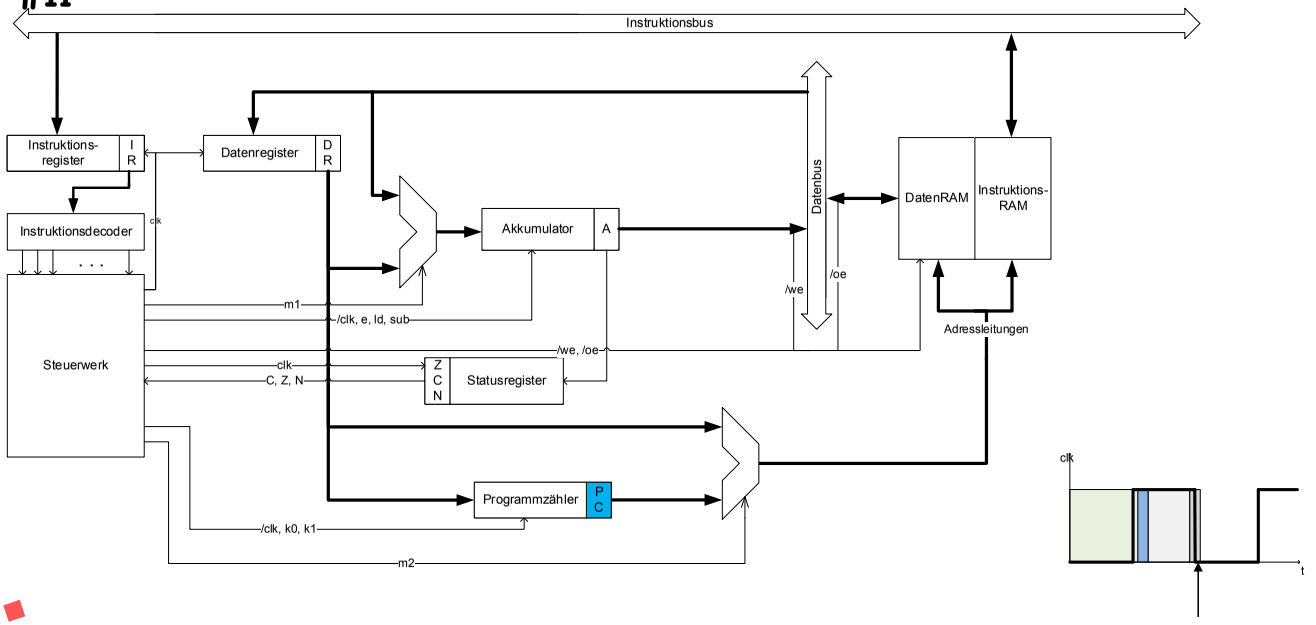


HM

Execute Phase für Sprünge

BRx #n

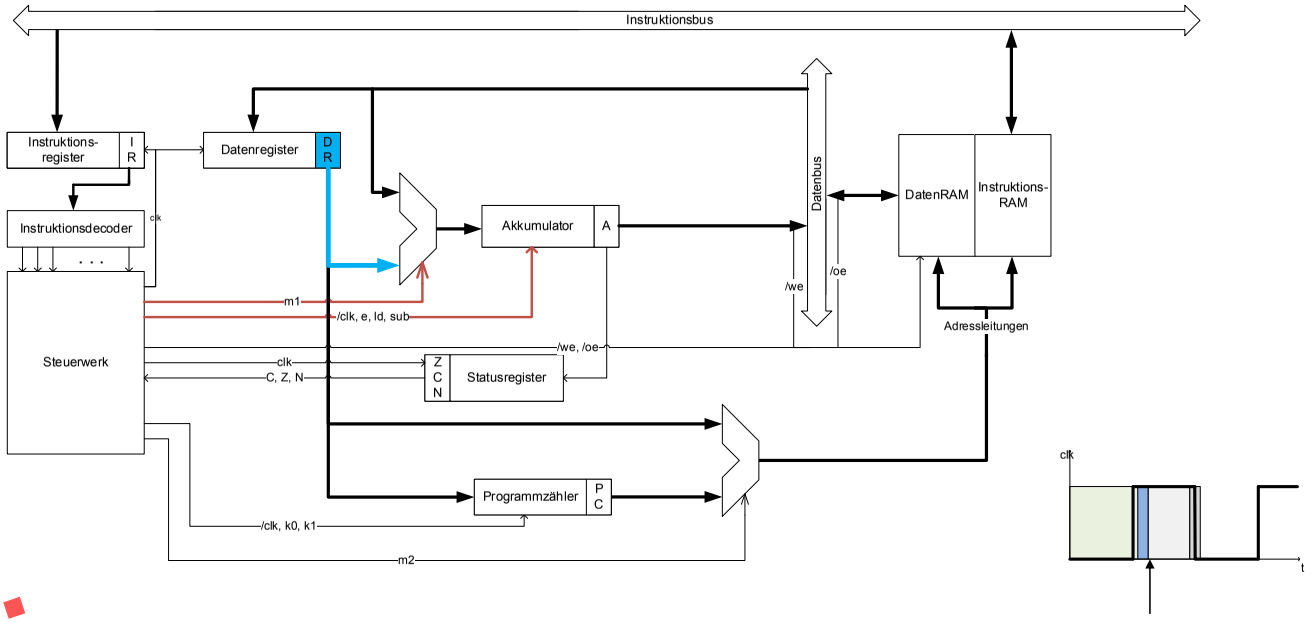
JMP #n



HM

Execute Phase für Immediate Adressierung

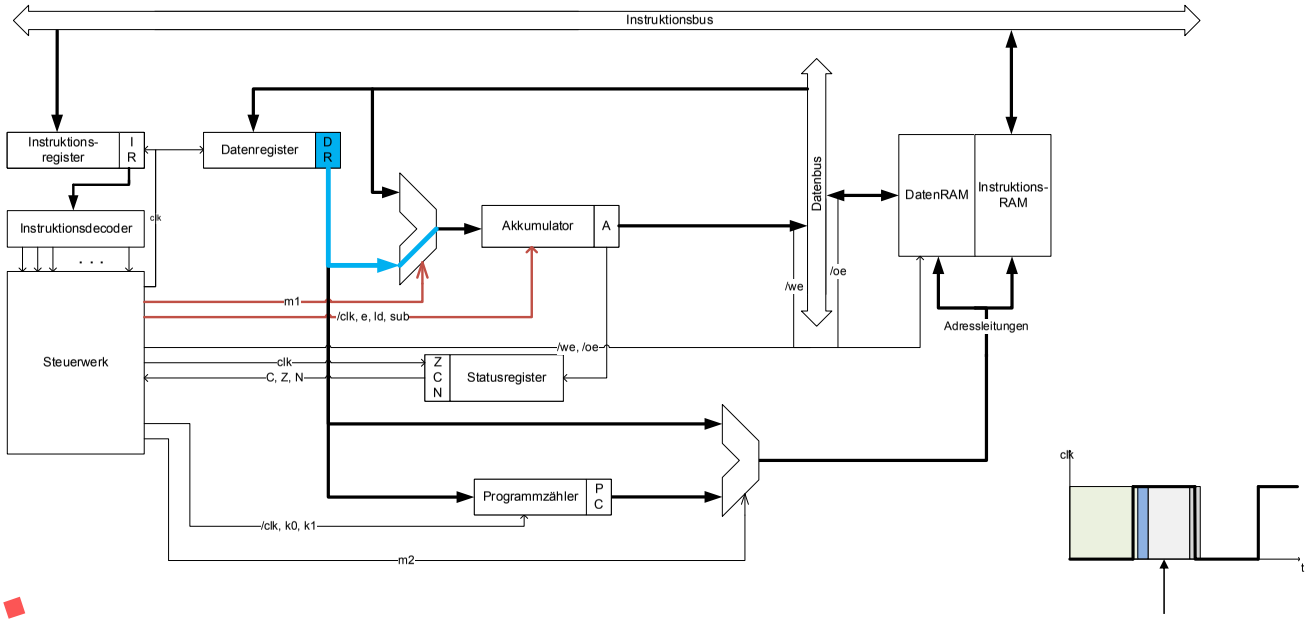
`LDA #n, ADD #n, SUB #n`



HM

Execute Phase für Immediate Adressierung

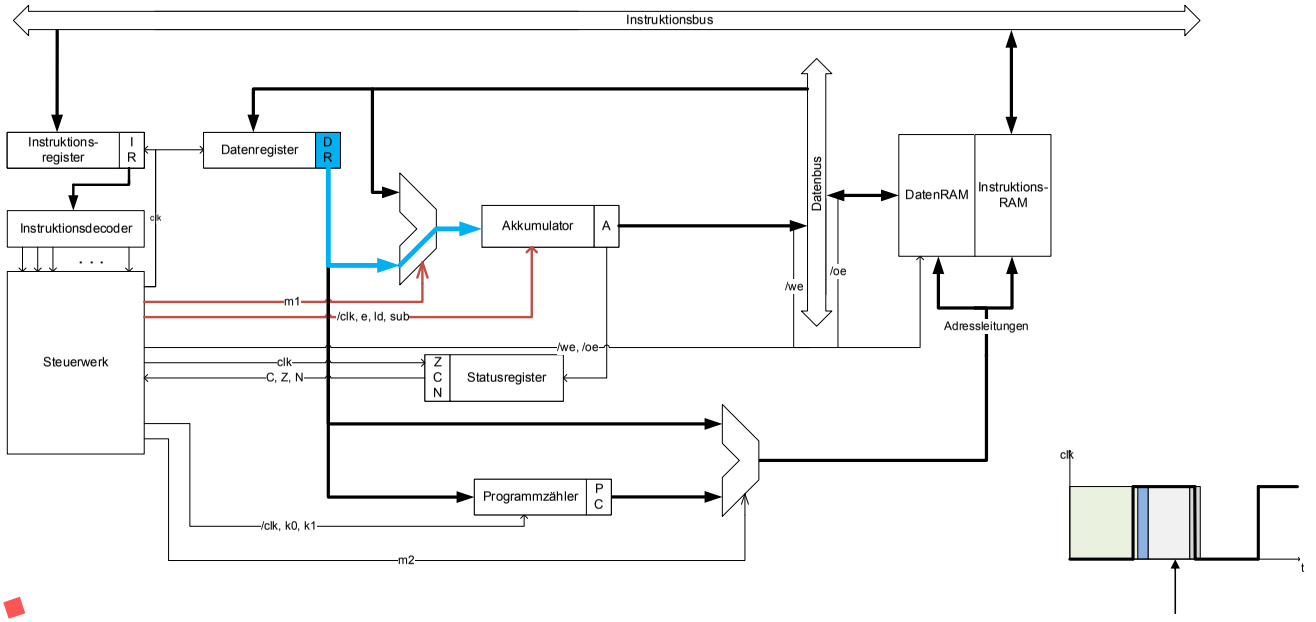
`LDA #n, ADD #n, SUB #n`



HM

Execute Phase für Immediate Adressierung

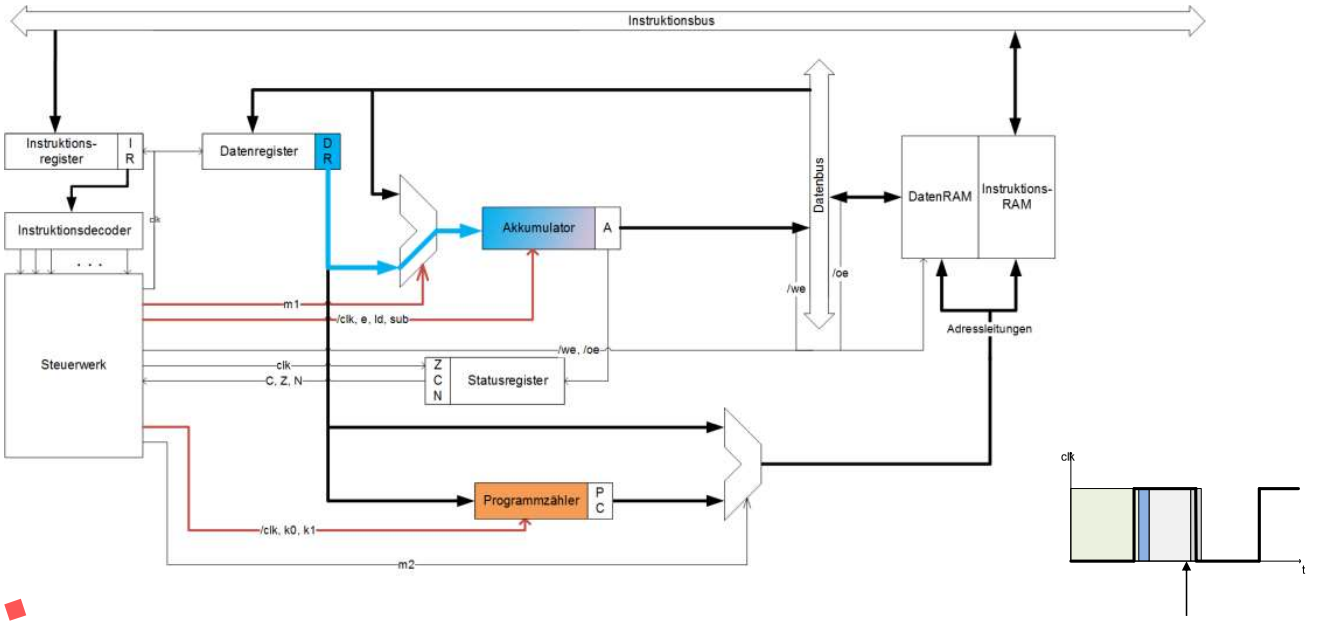
`LDA #n, ADD #n, SUB #n`



HM

Execute Phase für Immediate Adressierung

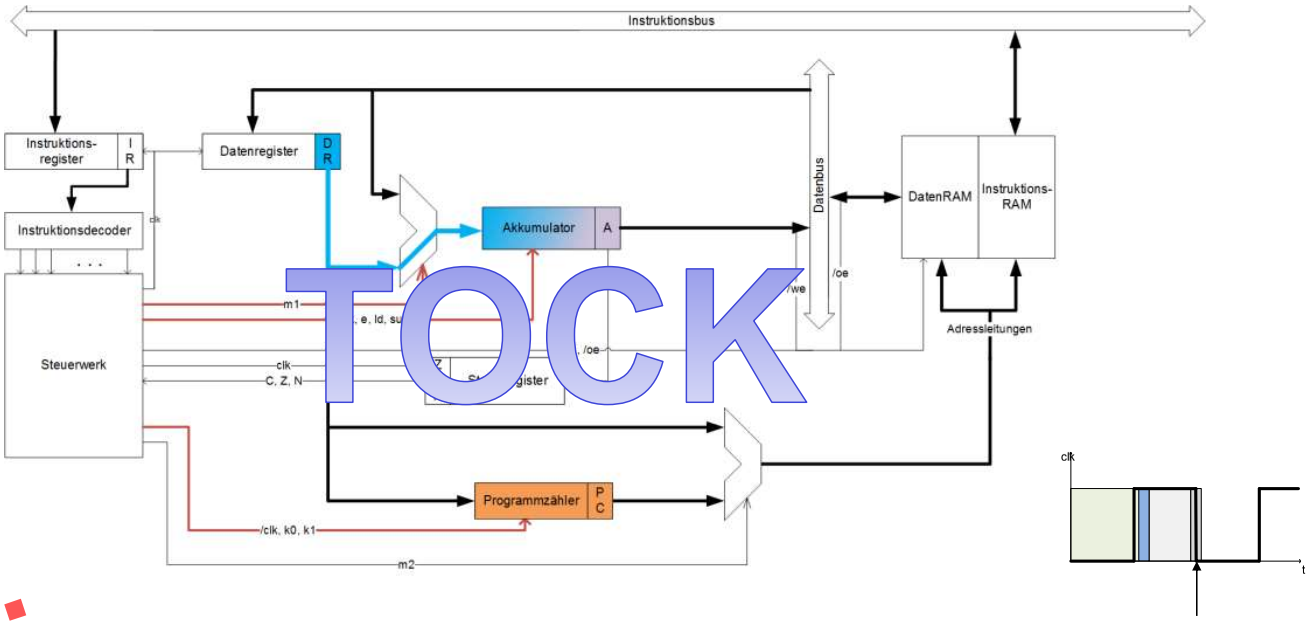
`LDA #n, ADD #n, SUB #n`



HM

Execute Phase für Immediate Adressierung

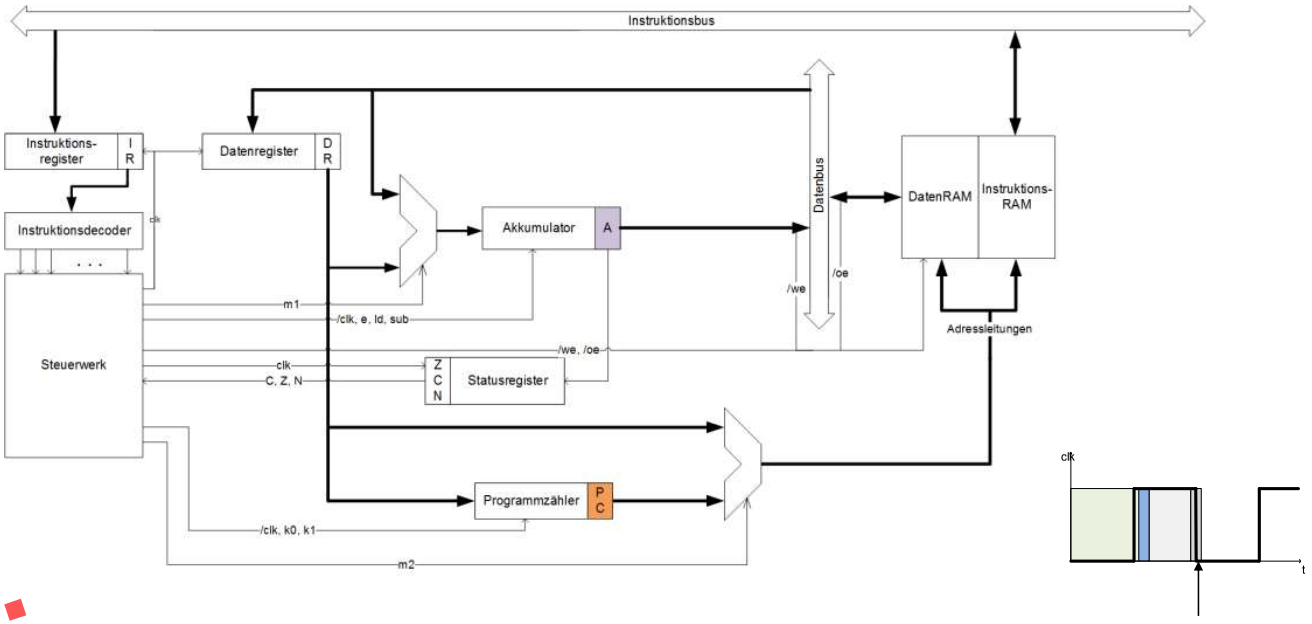
`LDA #n, ADD #n, SUB #n`



HM

Execute Phase für Immediate Adressierung

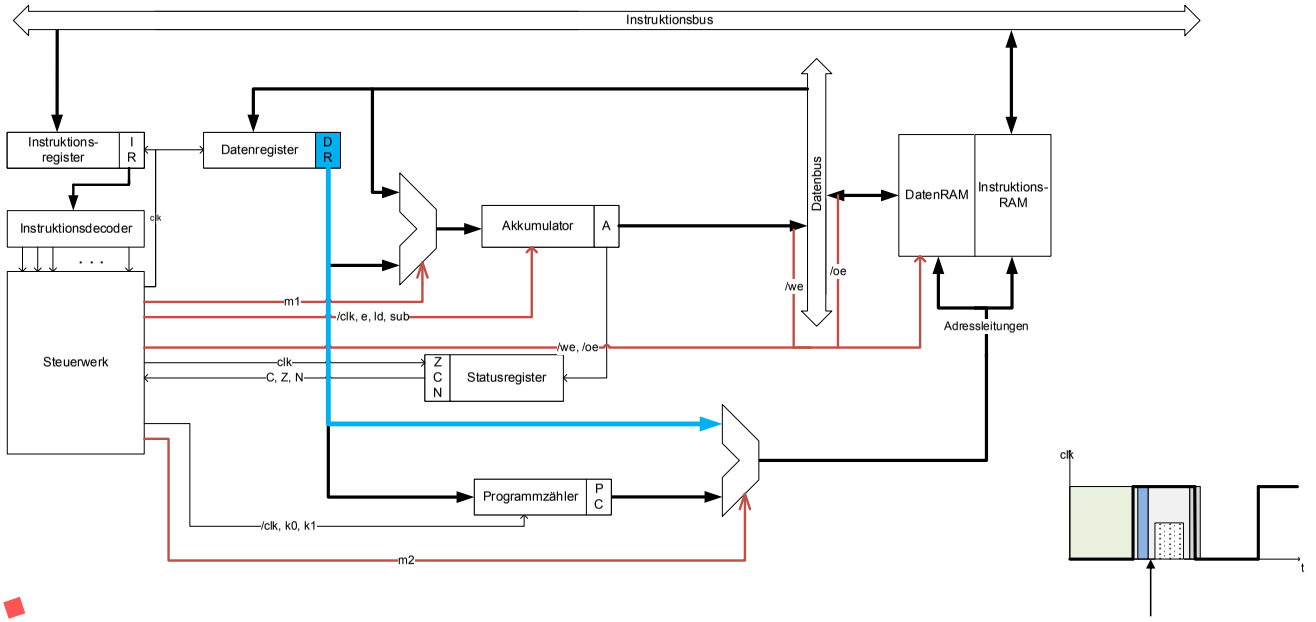
`LDA #n, ADD #n, SUB #n`



HM

Execute Phase für Operand aus Speicher

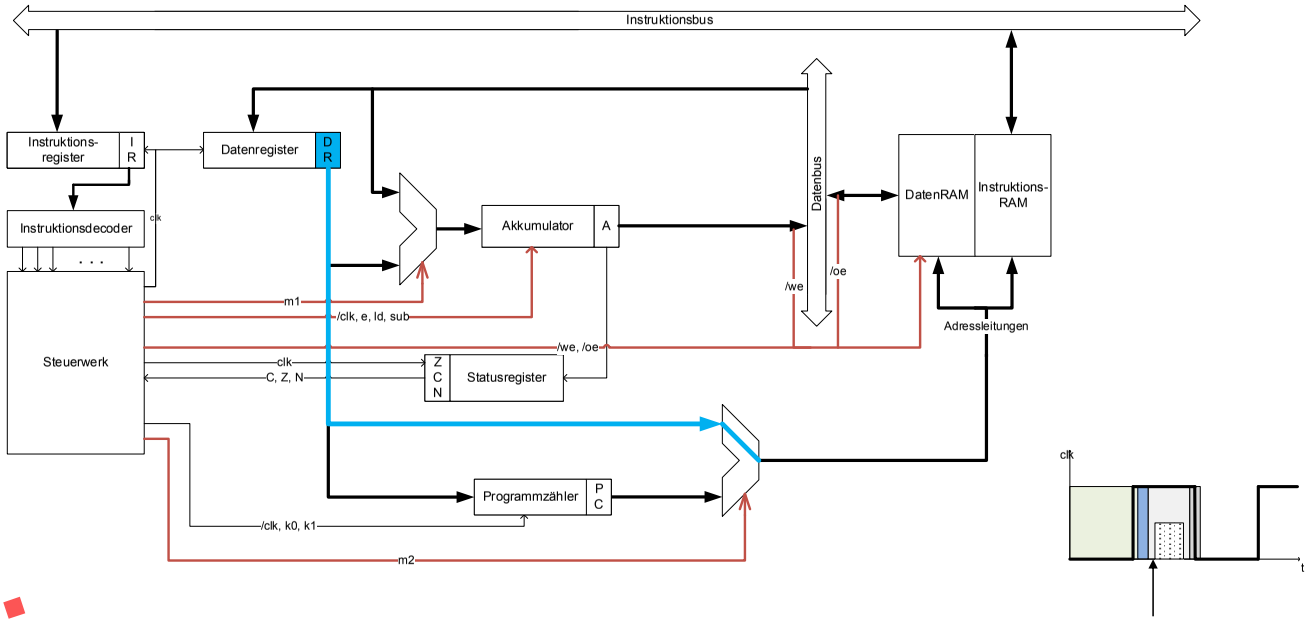
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

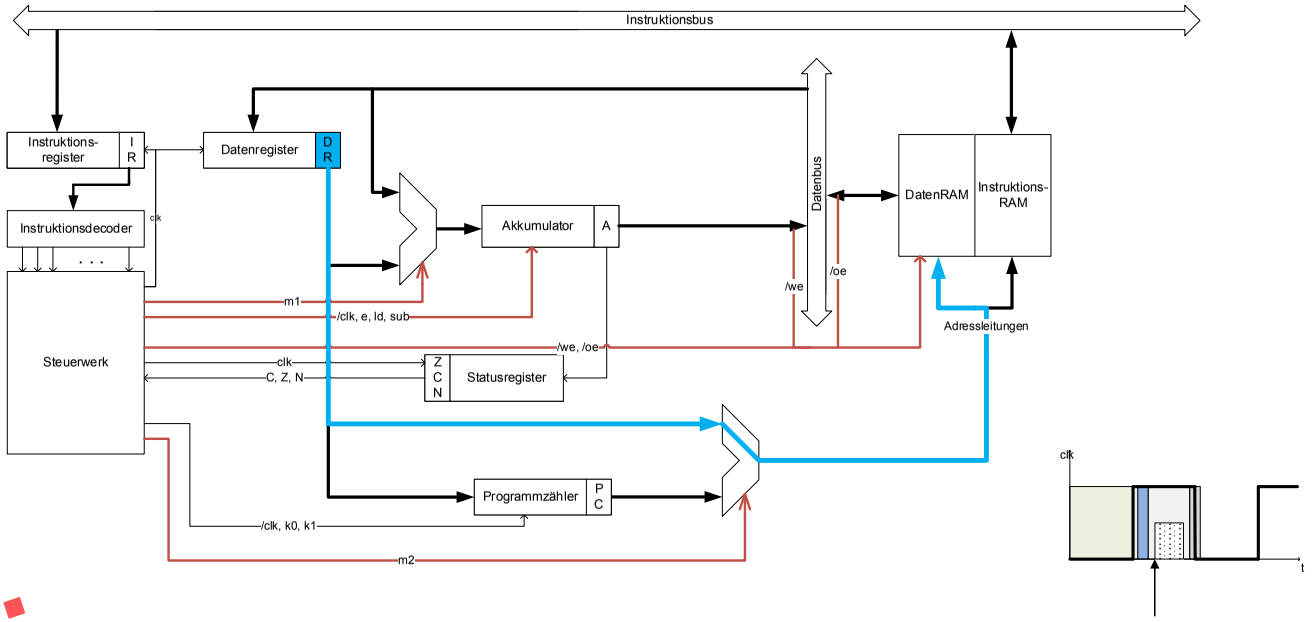
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

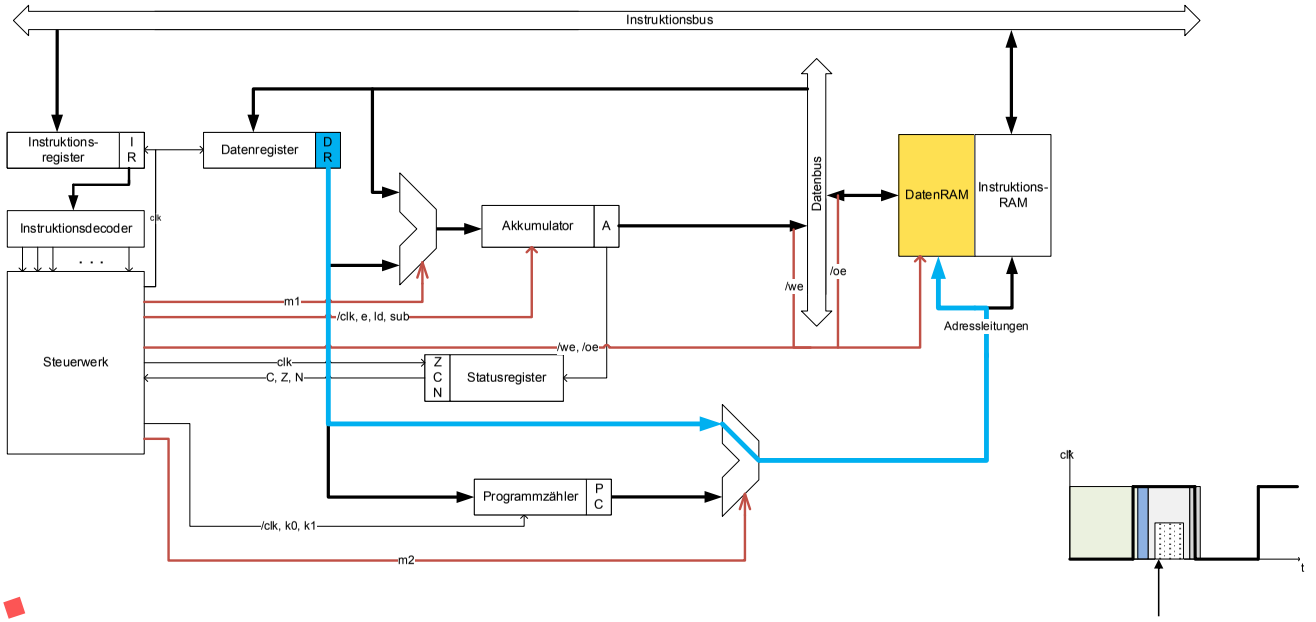
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

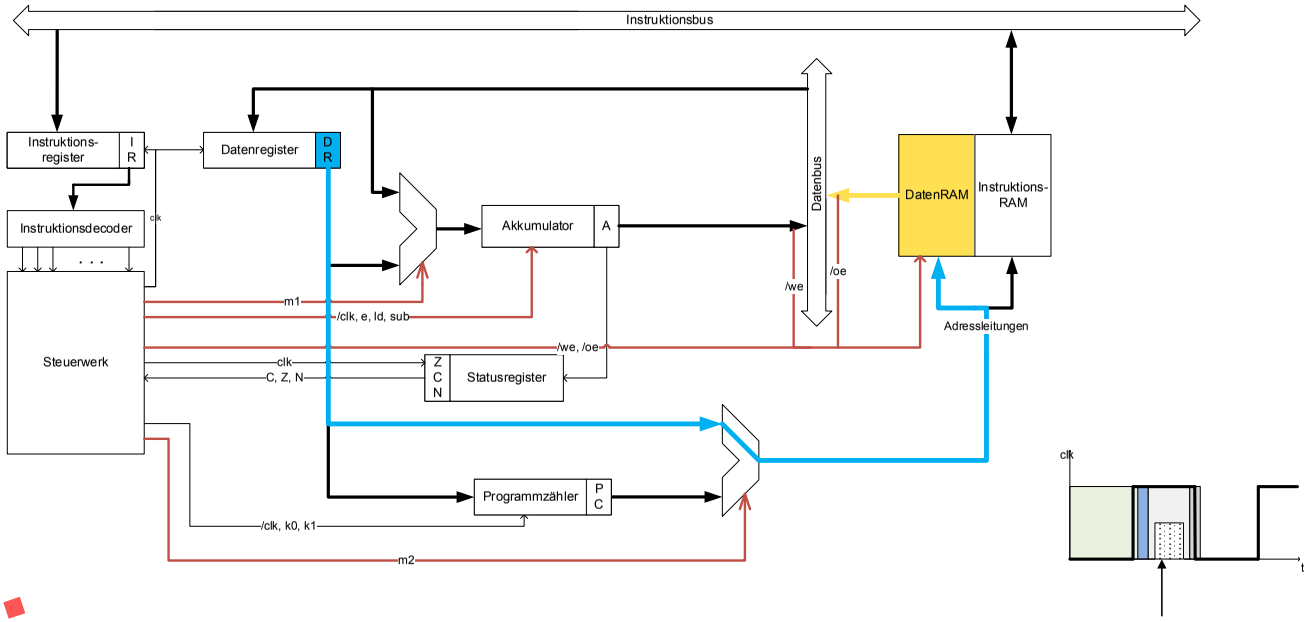
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

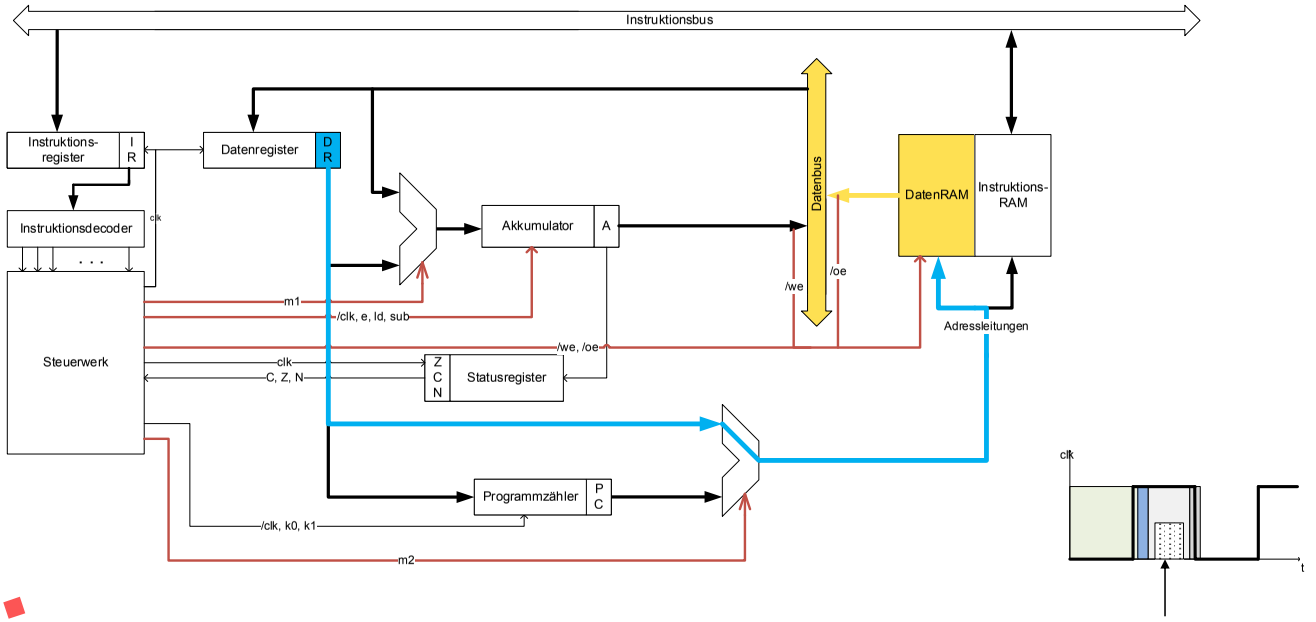
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

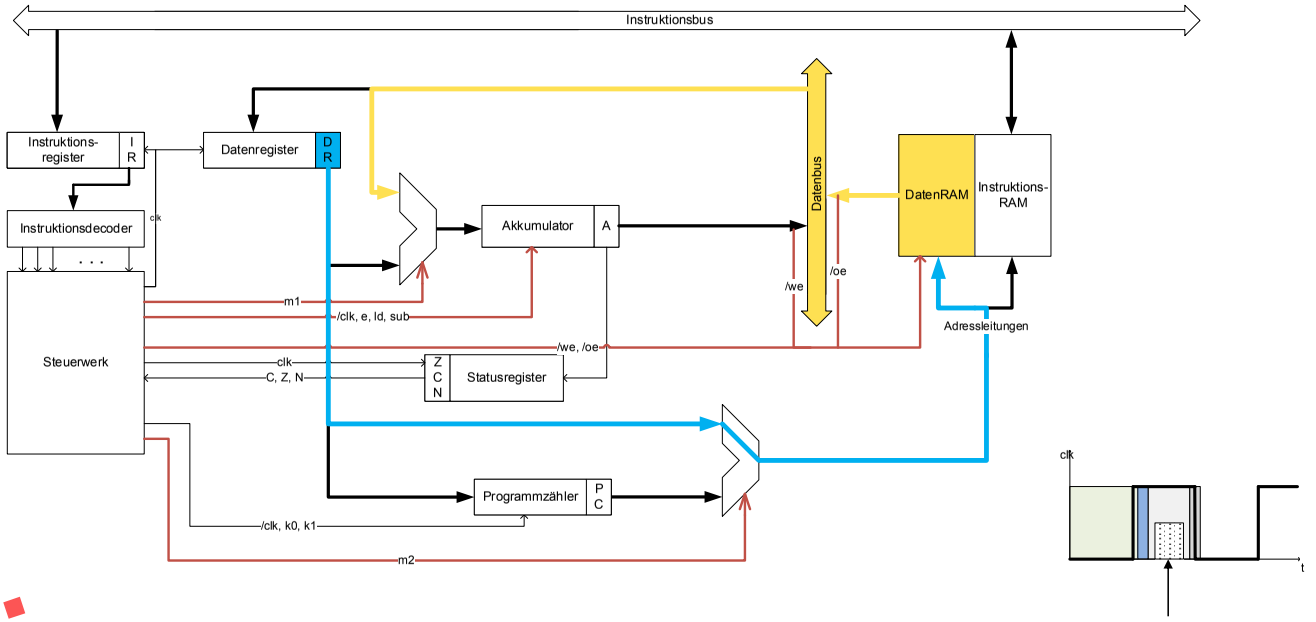
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

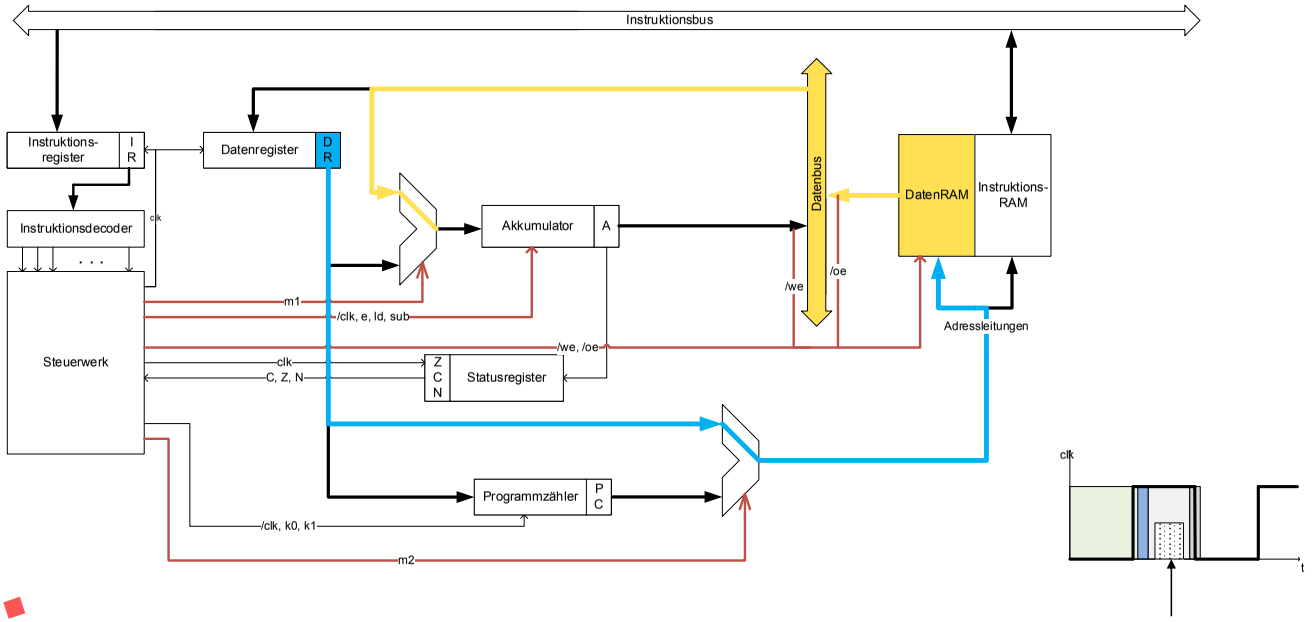
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

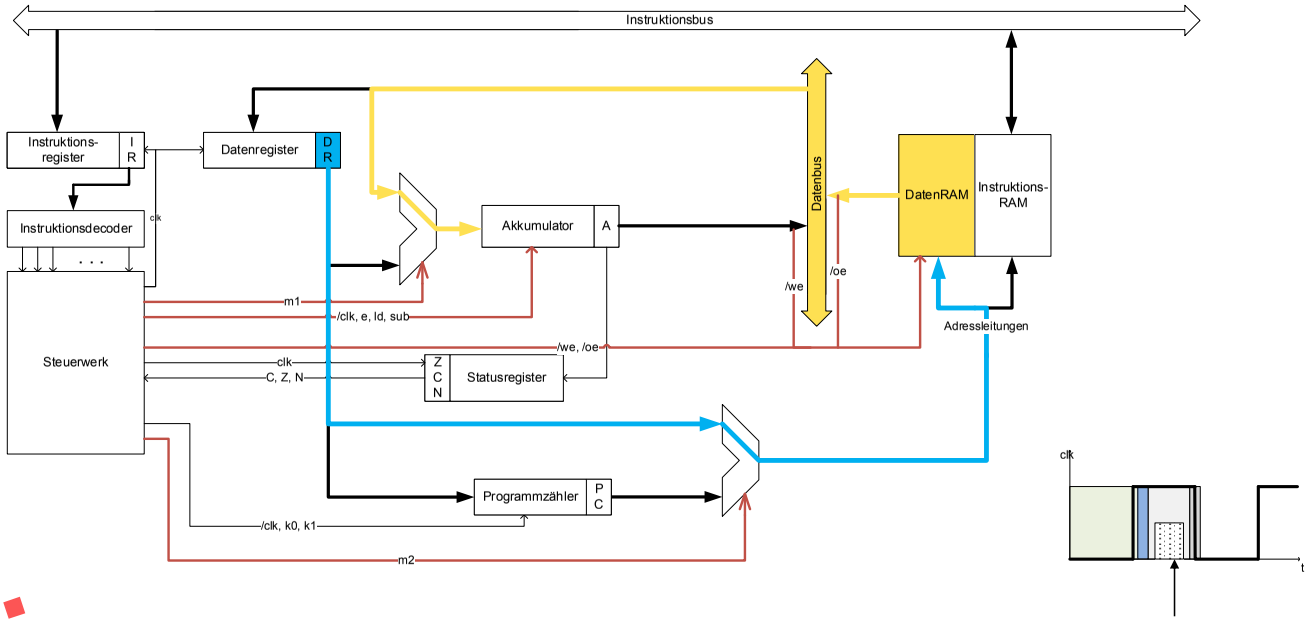
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

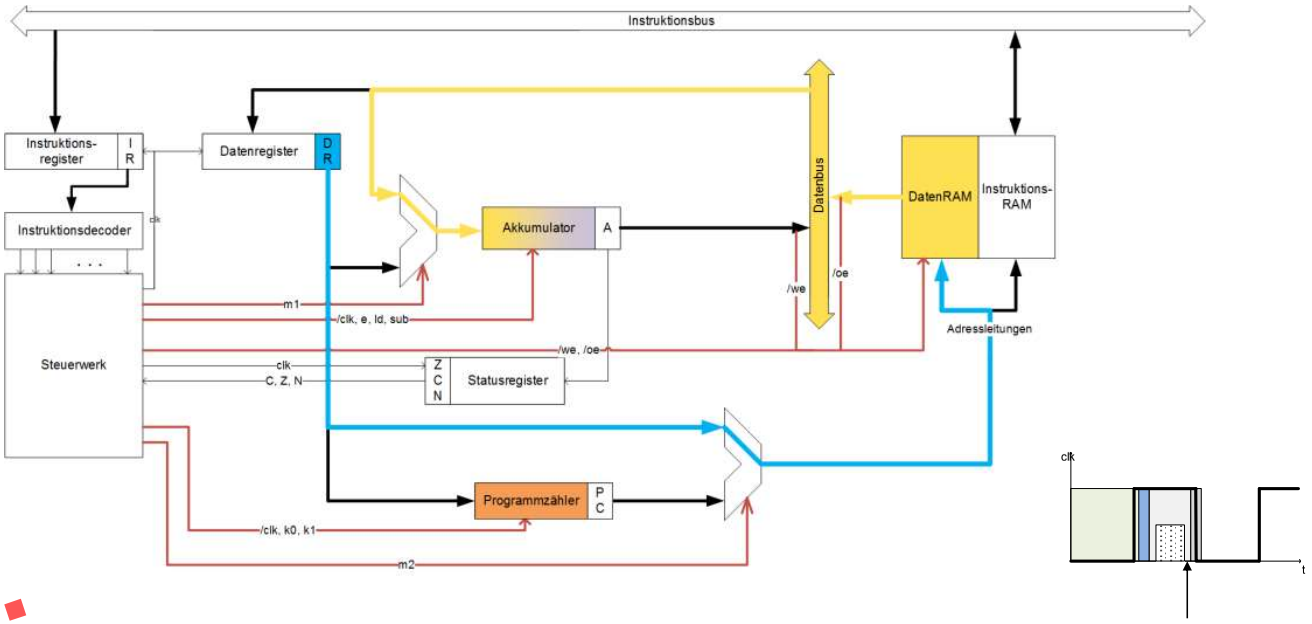
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Operand aus Speicher

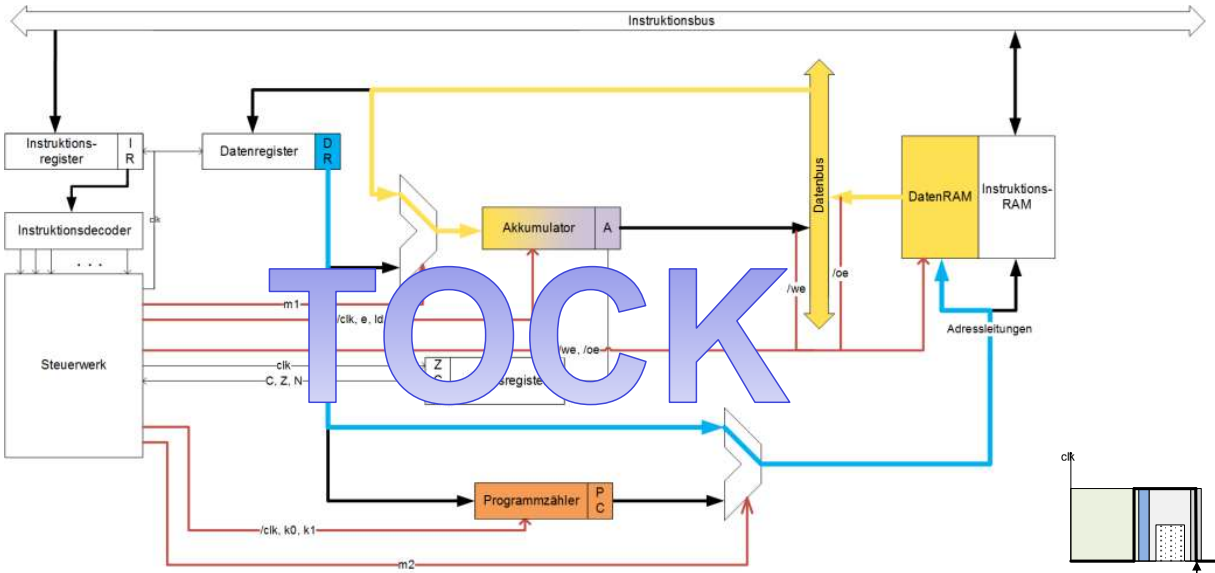
`LDA (n), ADD (n), SUB (n)`

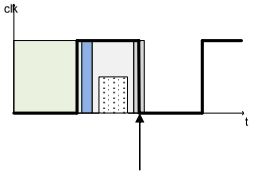


HM

Write Back Phase für Operand aus Speicher

`LDA (n), ADD (n), SUB (n)`

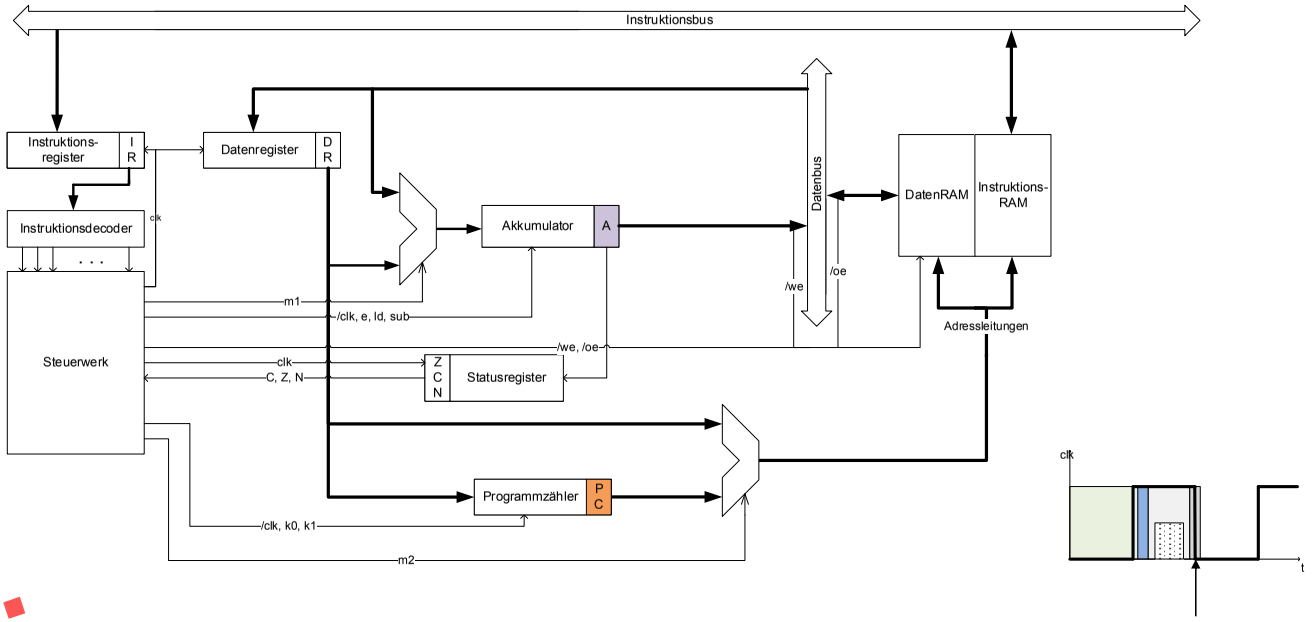




HM

Write Back Phase für Operand aus Speicher

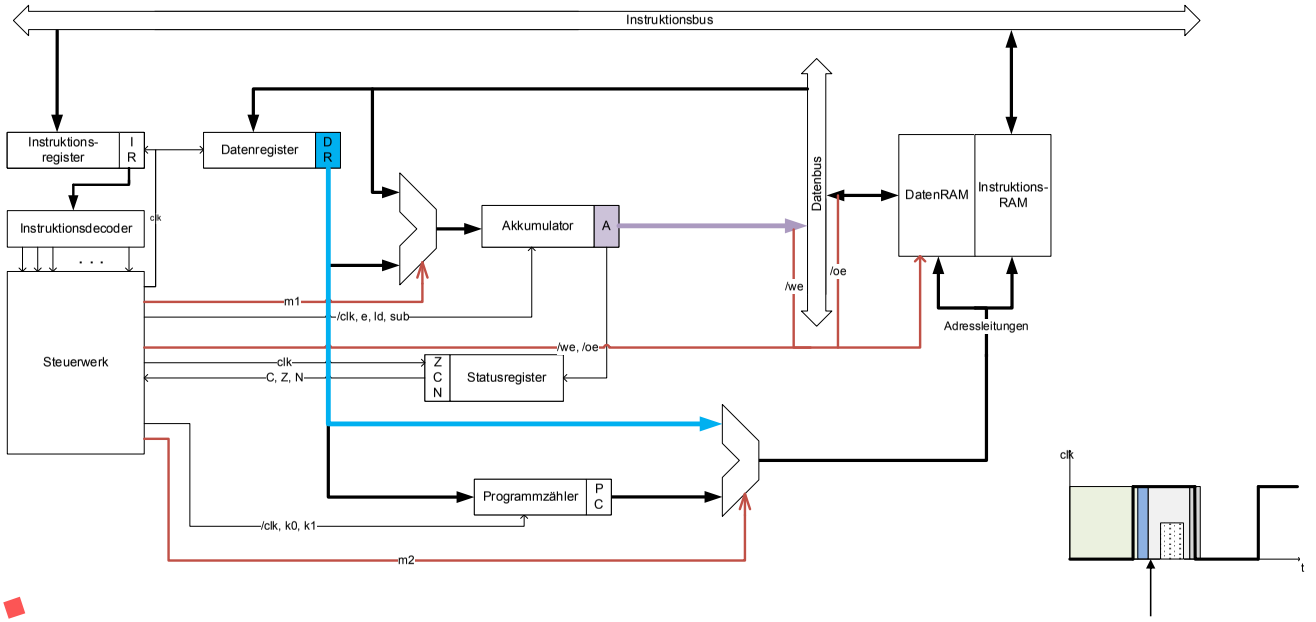
`LDA (n), ADD (n), SUB (n)`



HM

Execute Phase für Store in den Speicher

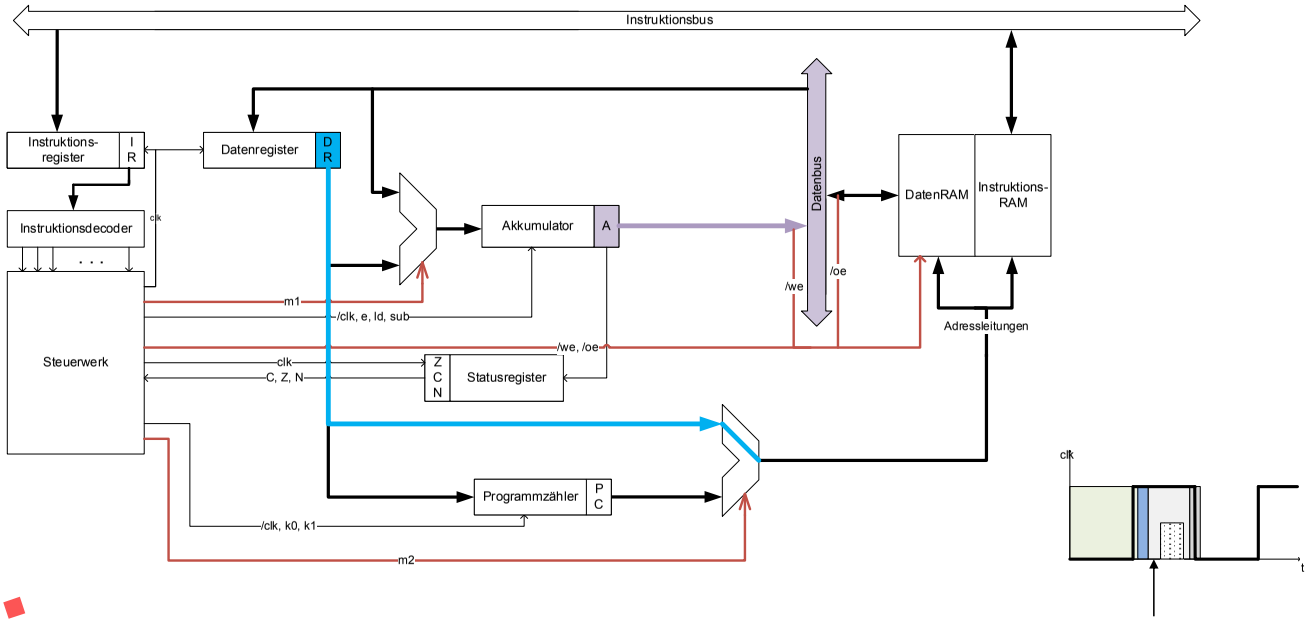
`STA n`



HM

Execute Phase für Store in den Speicher

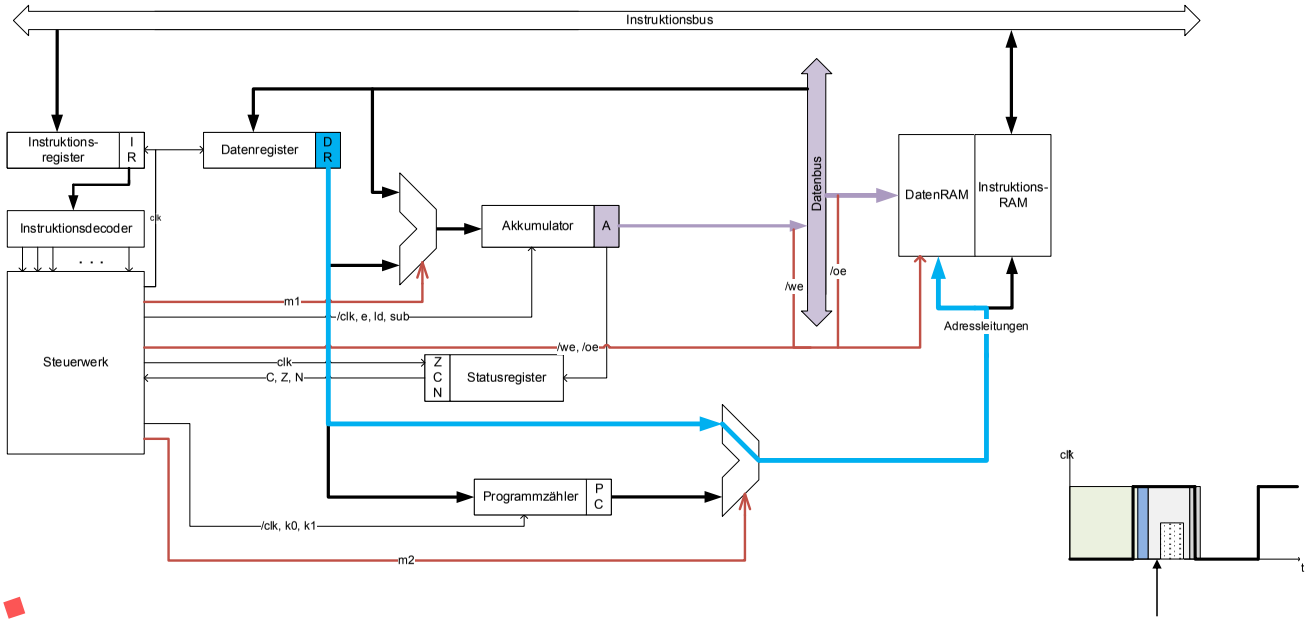
`STA n`



HM

Execute Phase für Store in den Speicher

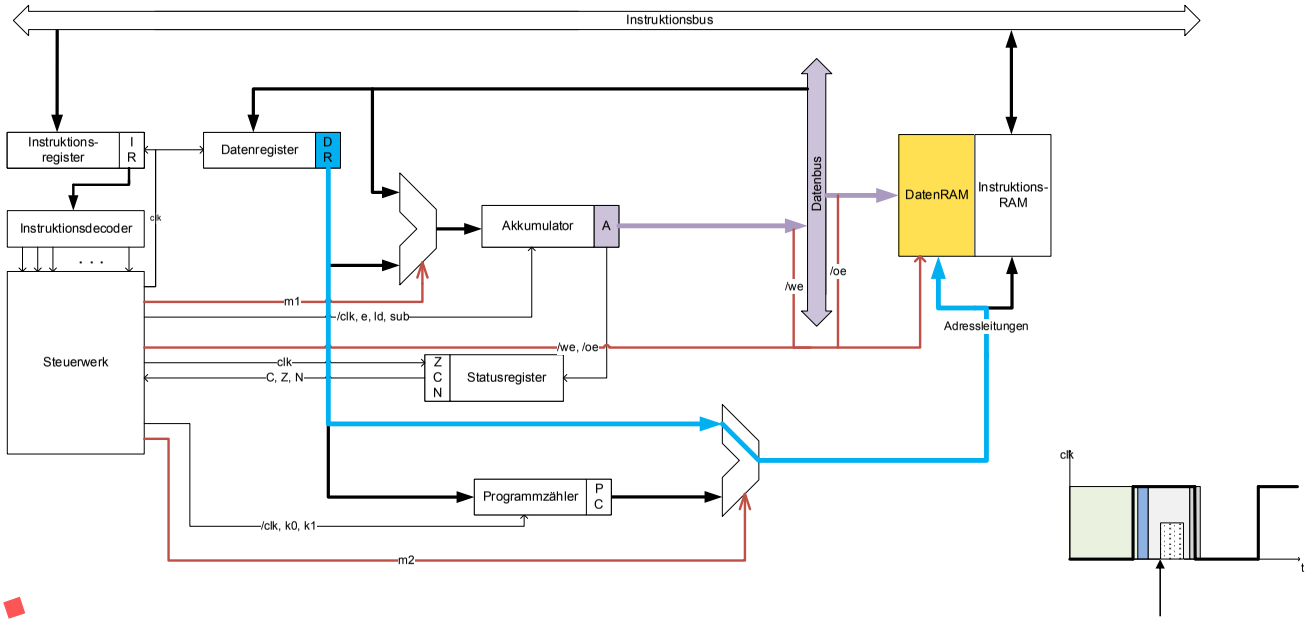
`STA n`



HM

Execute Phase für Store in den Speicher

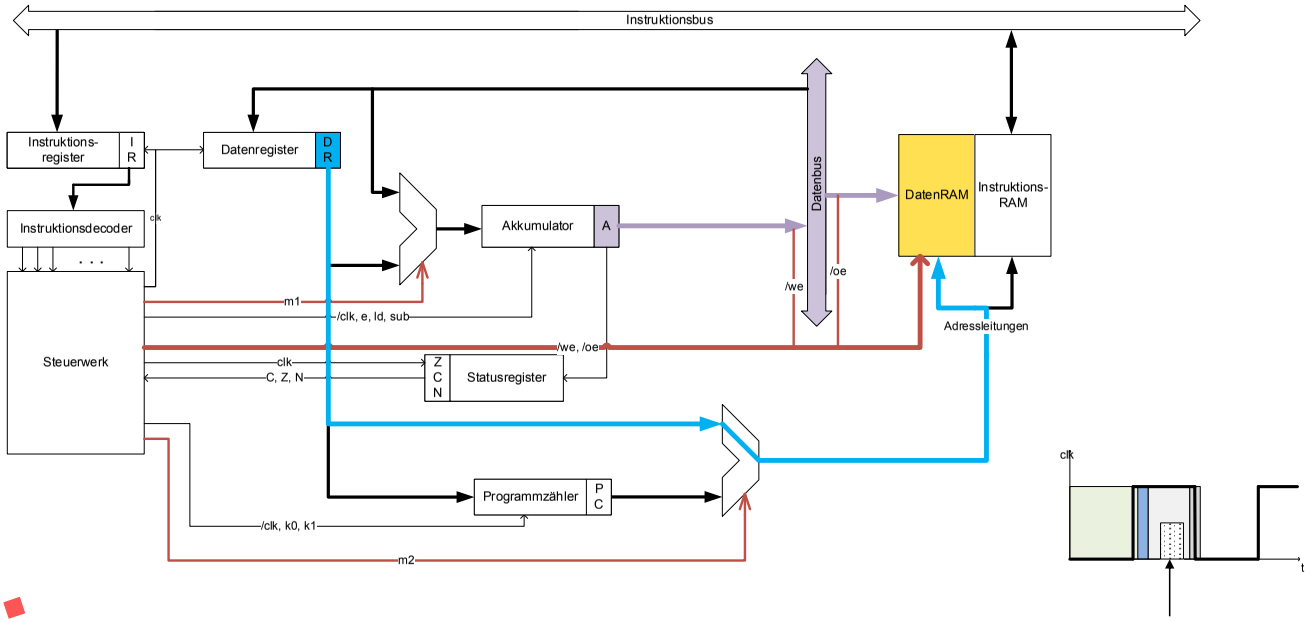
`STA n`



HM

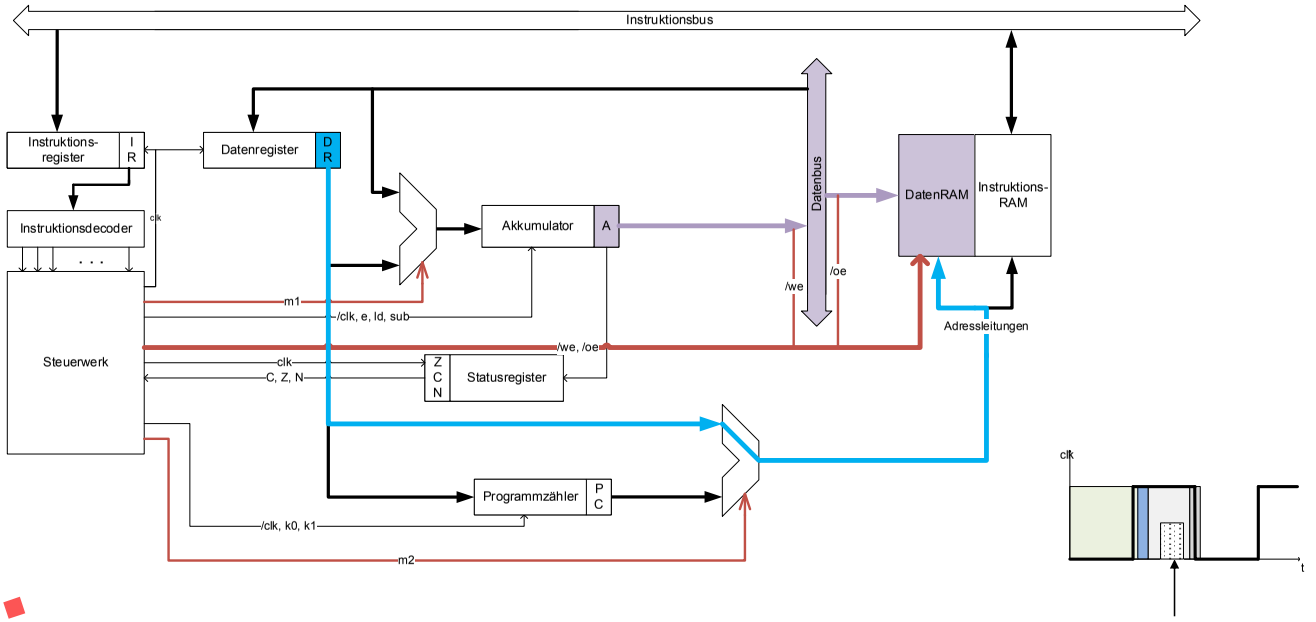
Execute Phase für Store in den Speicher

`STA n`



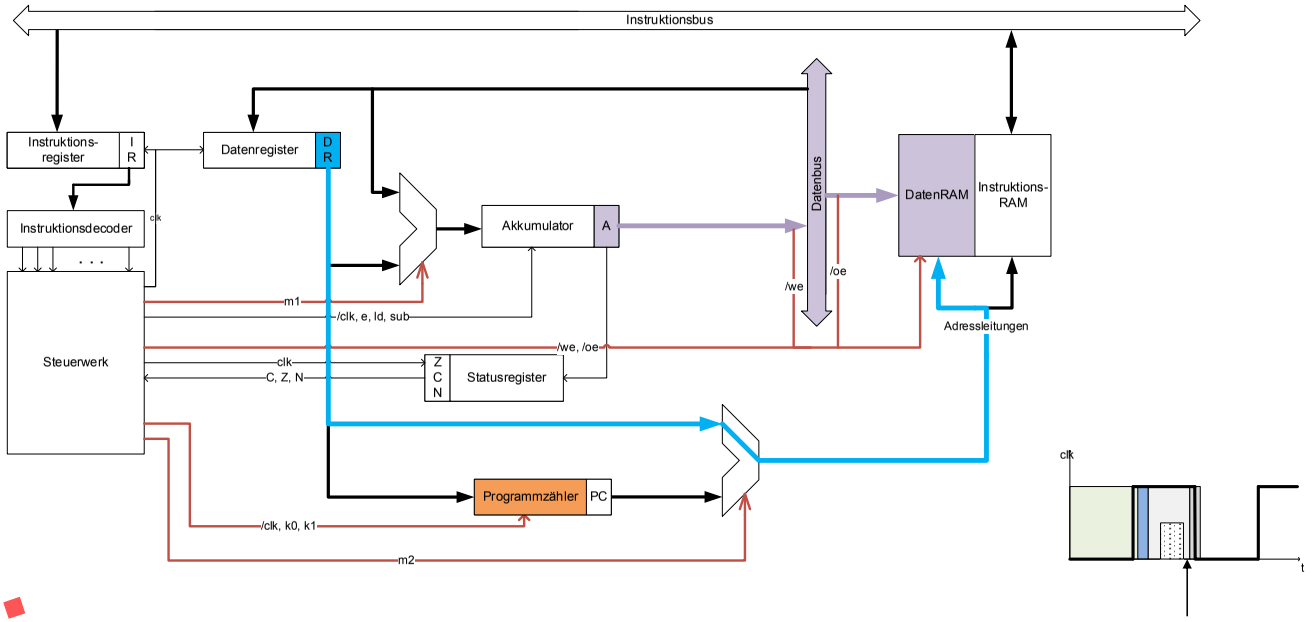
Execute Phase für Store in den Speicher

`STA n`



Execute Phase für Store in den Speicher

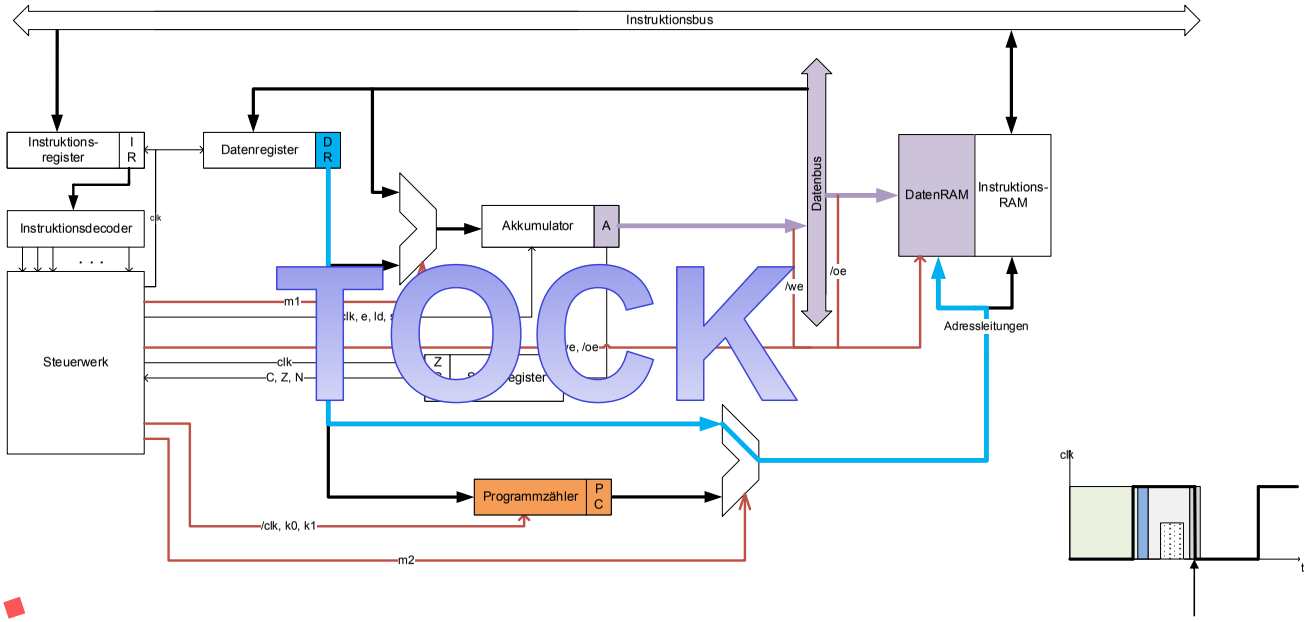
`STA n`



HM

Write Back Phase für Store in den Speicher

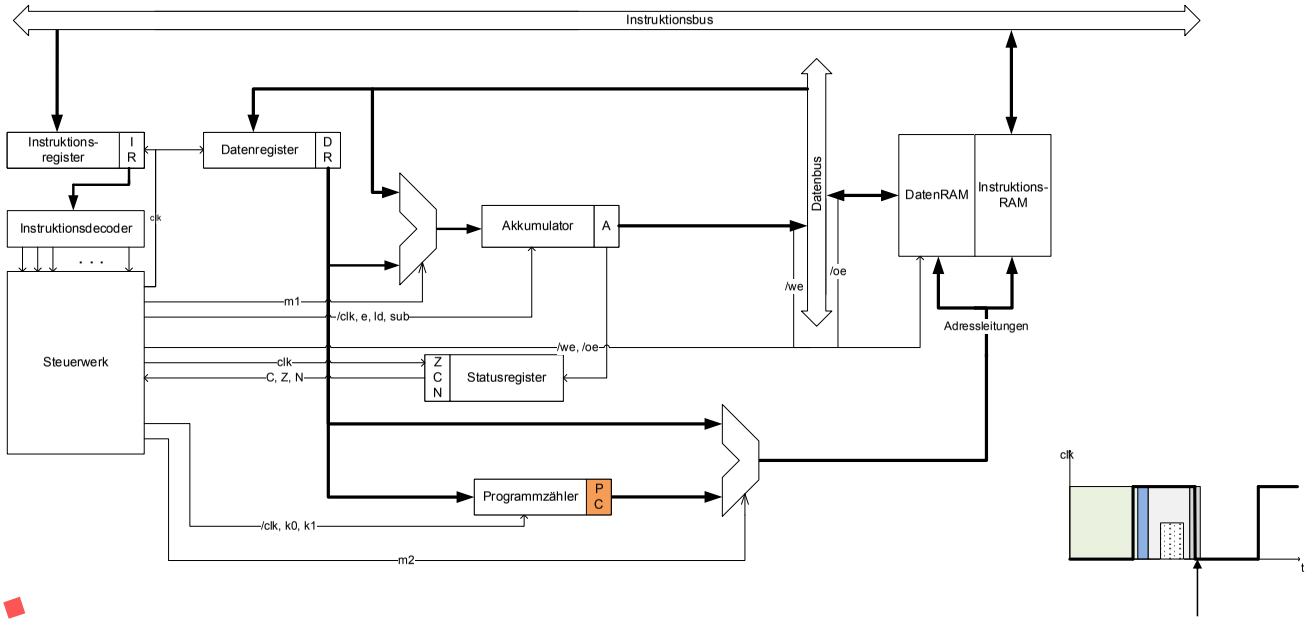
`STA n`



HM

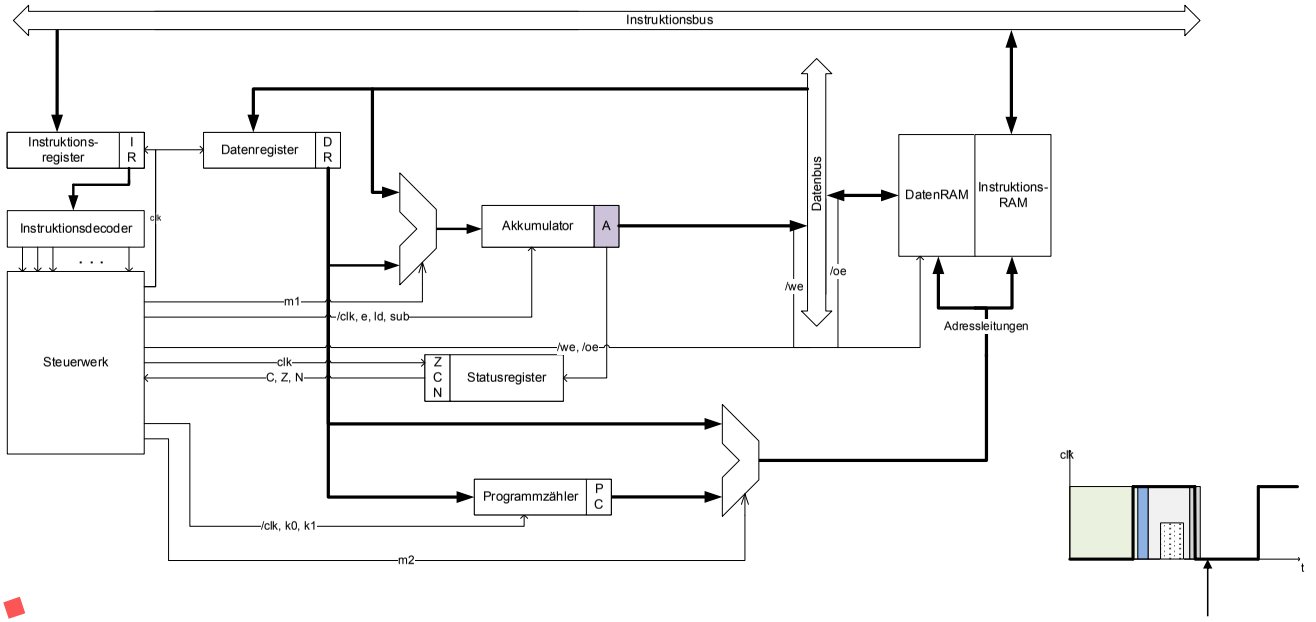
Write Back Phase für Store in den Speicher

`STA n`



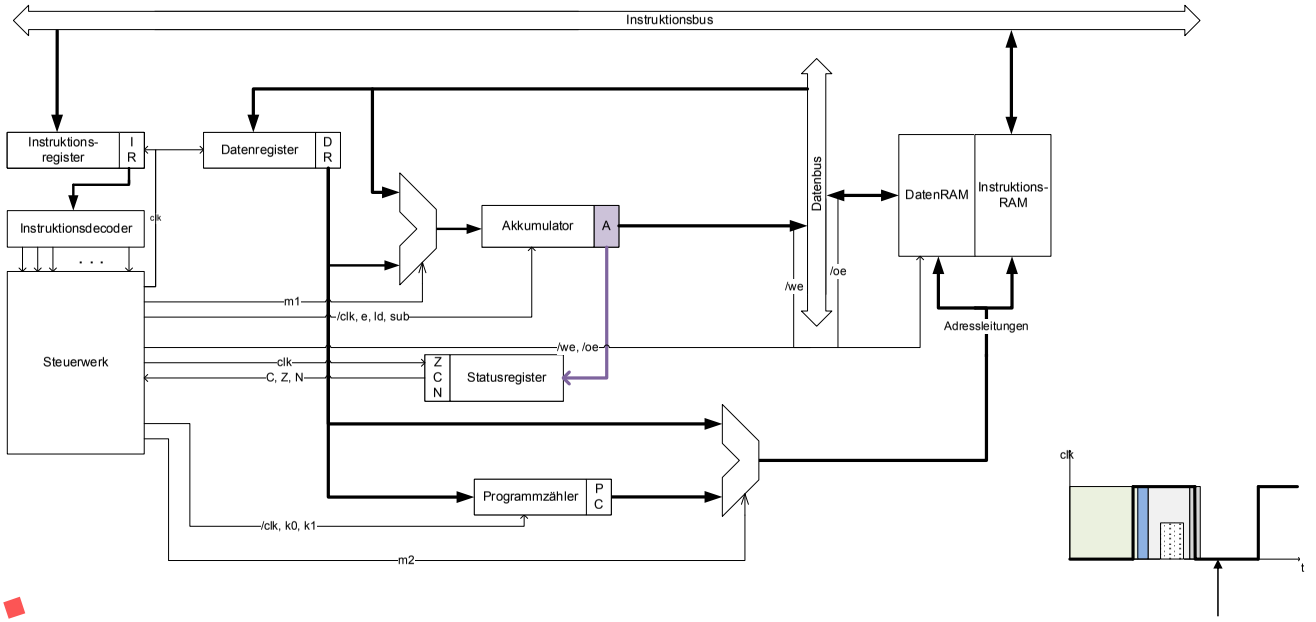
HM

Update des Status Registers



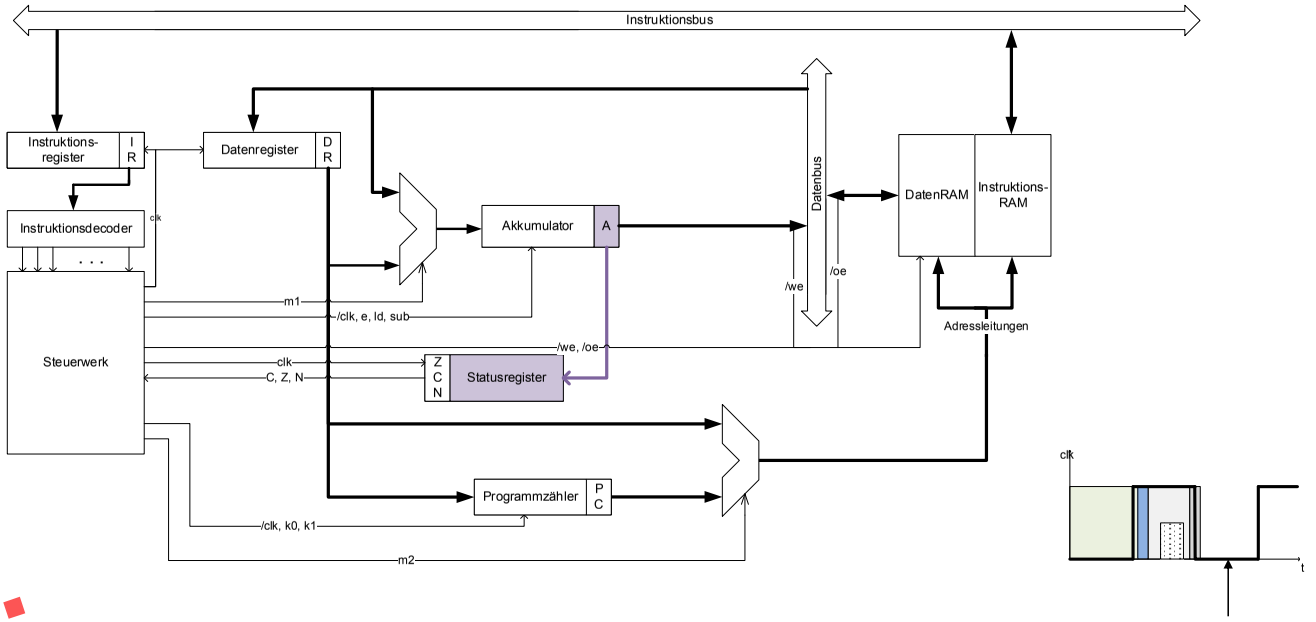
HM

Update des Status Registers



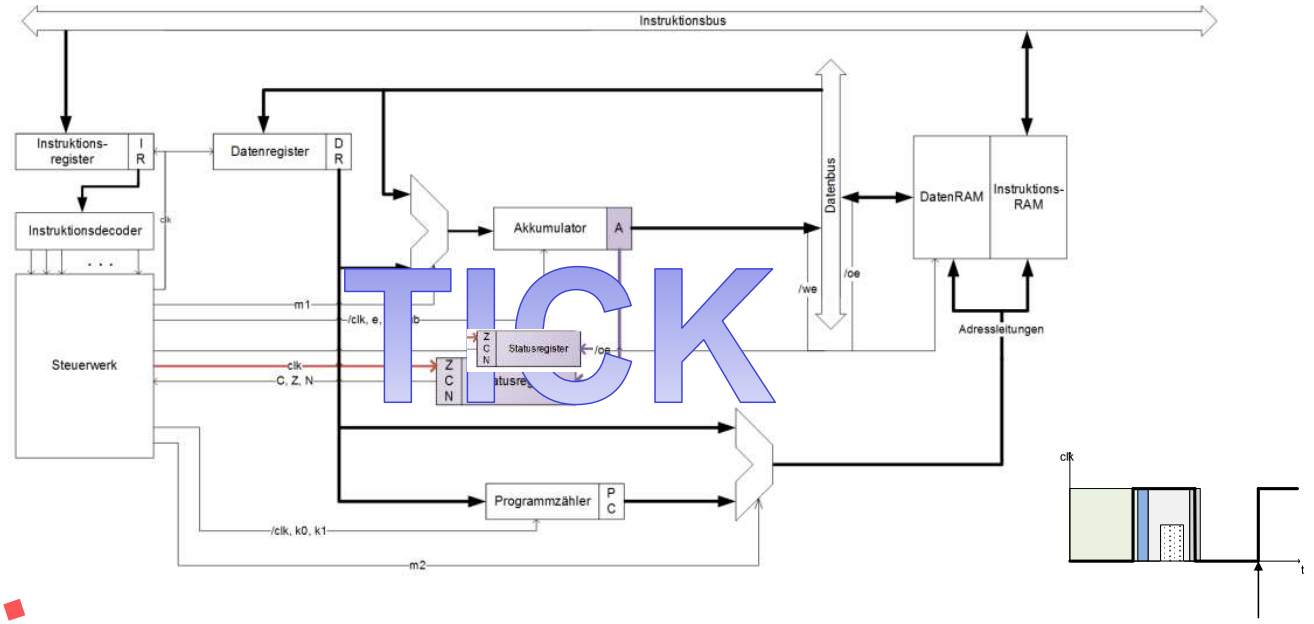
HM

Update des Status Registers



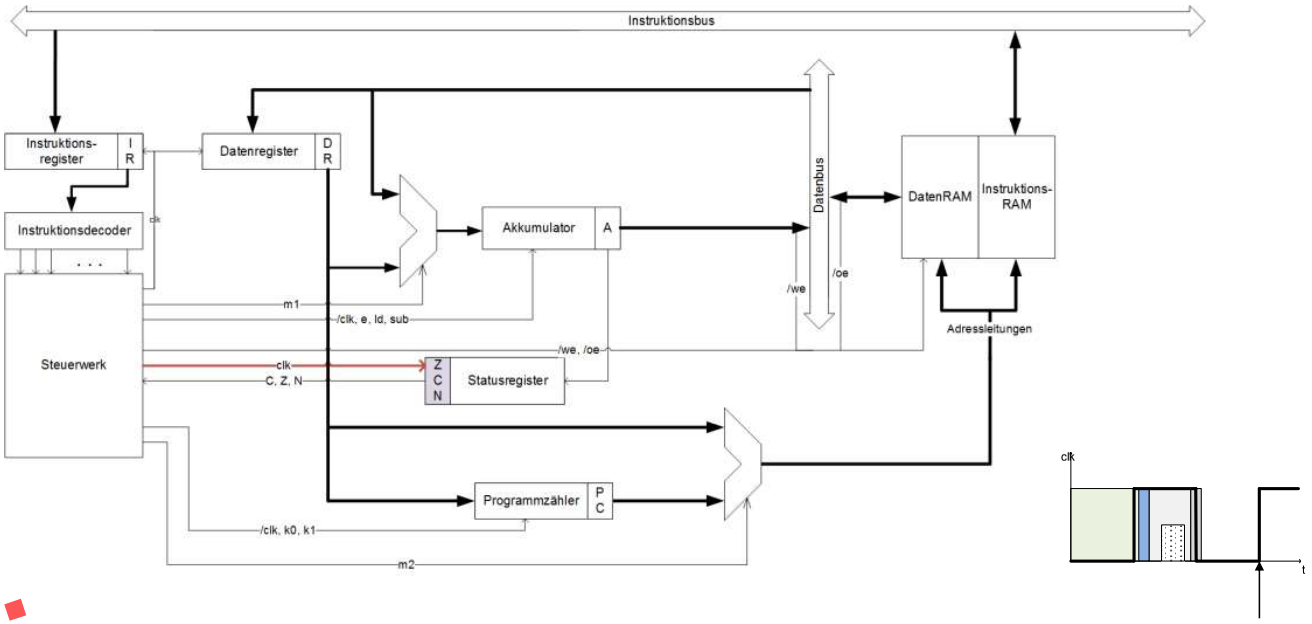
HM

Update des Status Registers

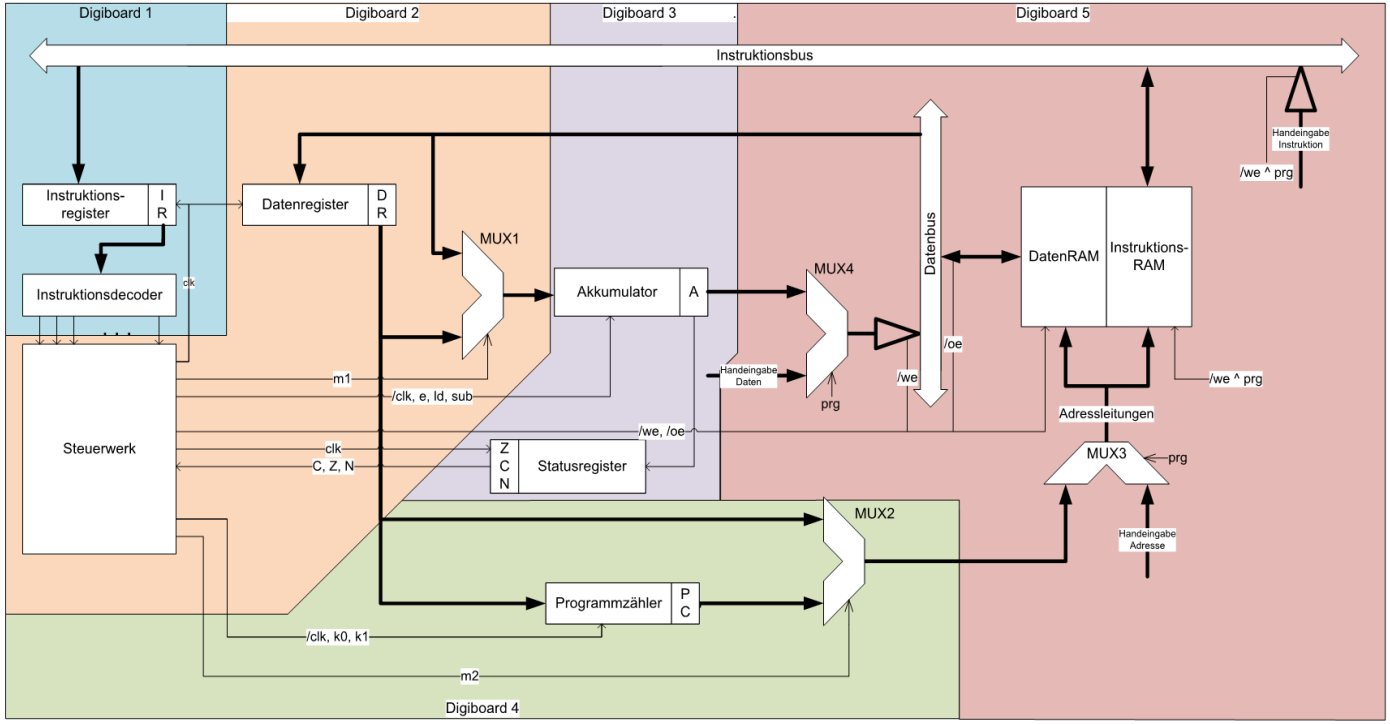


HM

Update des Status Registers

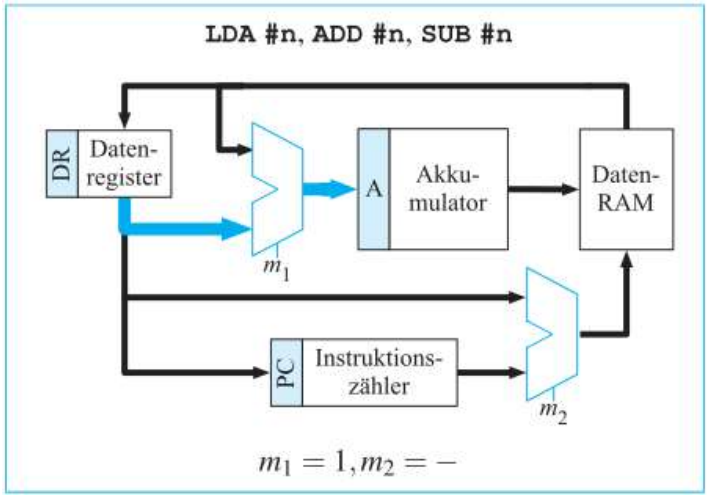


HM


Digiboard 1
Digiboard 2
Digiboard 3
Digiboard 5
Instruktionsbus
Handeingabe Instruktion
/we ^ prg
Instruktions-register
I R
Datenregister
D R
MUX1
Datenbus
DatenRAM
Instruktions-RAM
Instruktionsdecoder
clk
Akkumulator
A
MUX4
Handeingabe Daten
prg
/we
/oe
/we ^ prg
Steuerwerk
m1
/clk, e, ld, sub
/we, /oe
Adressleitungen
clk
C, Z, N
Z C N
Statusregister
MUX3
prg
MUX2
Handeingabe Adresse
Programmzähler
P C
/clk, k0, k1
m2
Digiboard 4

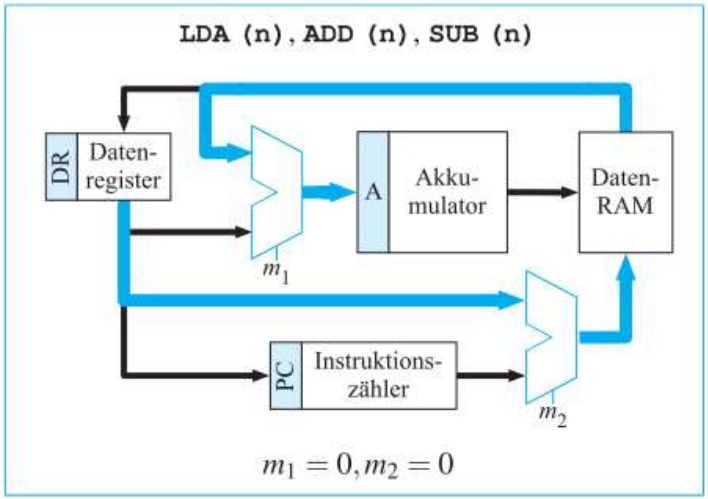
Datenfluss innerhalb des Modellrechners

- Immediate Werte, d.h. **#n** im Befehl ist ein Wert enthalten!



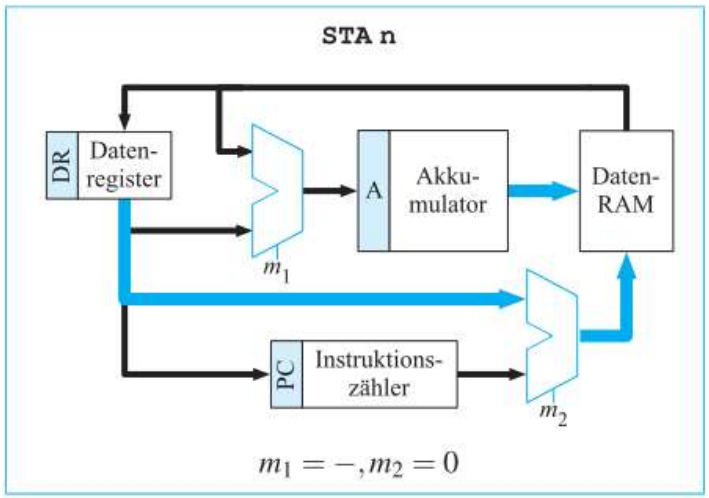
Datenfluss innerhalb des Modellrechners

- Adresse ist im Befehl enthalten!



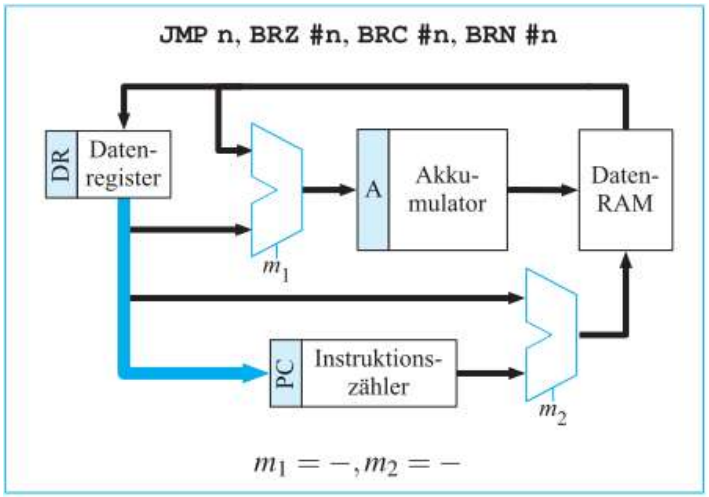
Datenfluss innerhalb des Modellrechners

- Adresse ist im Befehl enthalten + Speichern ins RAM!



Datenfluss innerhalb des Modellrechners

- Sprungbefehle (Sprungziel im Befehl enthalten)!



Datenfluss innerhalb des Modellrechners