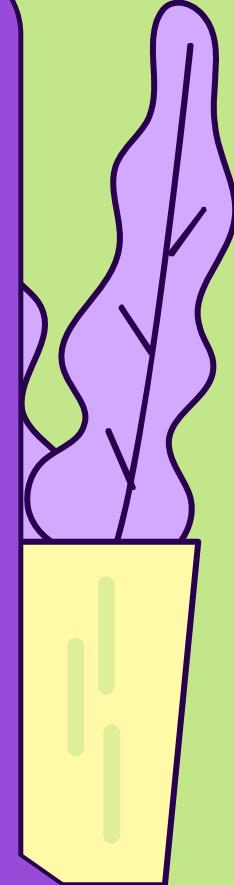




Herzlich willkommen
zum neunten ERA
Tutorium des
Semesters!



ERA Tutorium - Mo-1400-2 > Zehnte Woche

DEC 13



Berrak Kılıç 🔥

13:50

Es wird am 23.12 kein Tutorium geben.



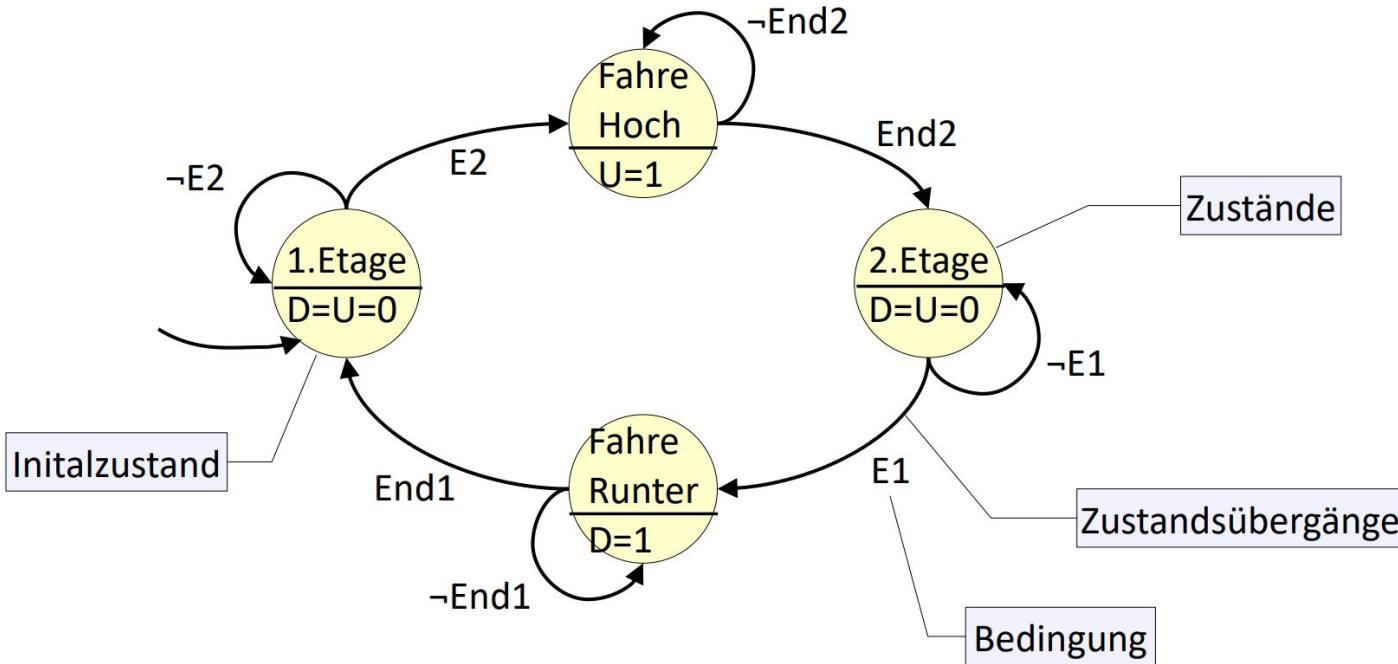
Salomon Glaser

Endlicher Automat

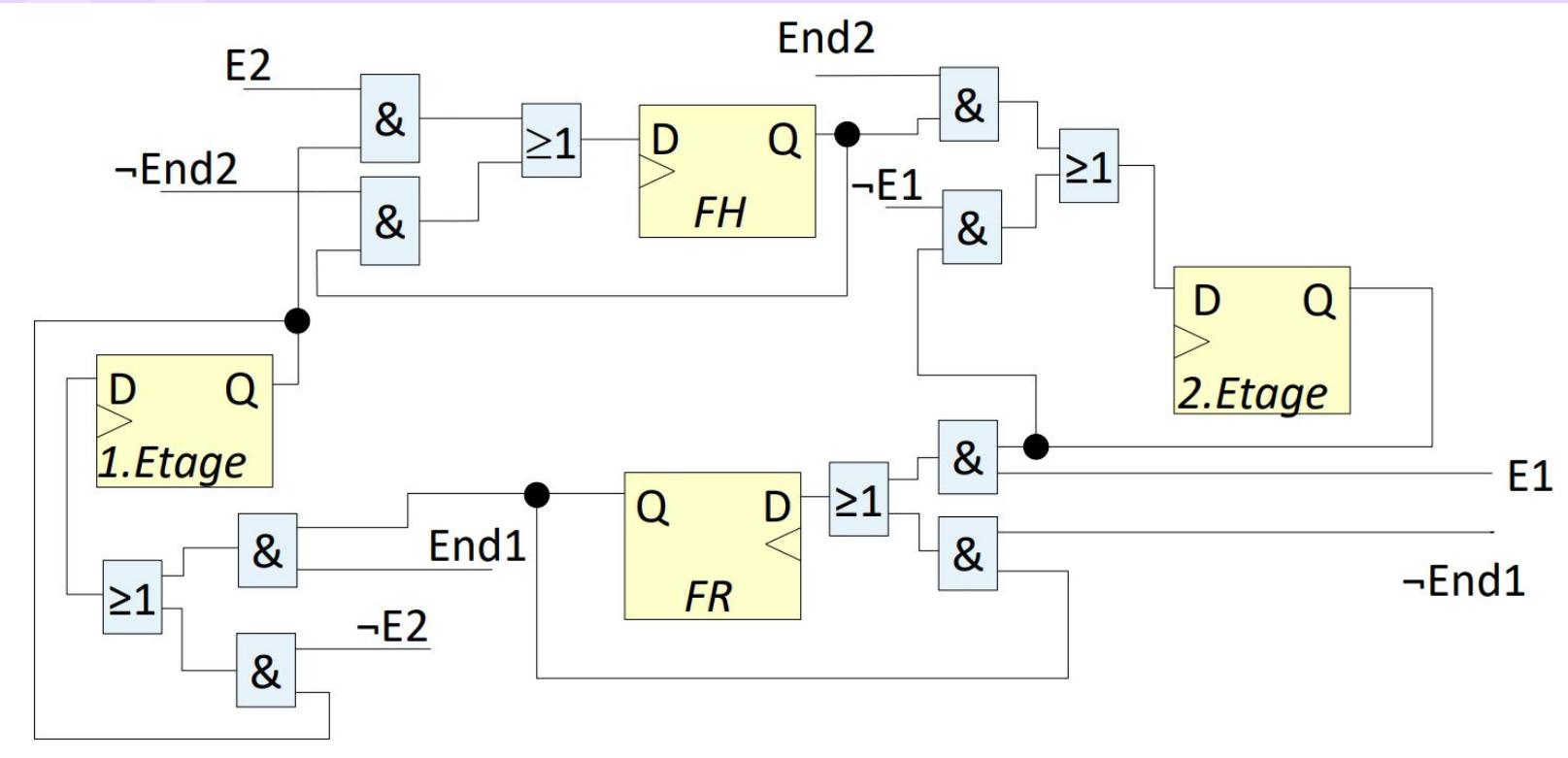
- Repräsentiert die Funktion einer sequenziellen Schaltung
- Mathematische Darstellung: $A(I, O, S, s_0, \delta, \lambda)$
 - I ...Menge der Eingaben (inputs)
 - O ...Menge der Ausgaben (outputs)
 - S ...Menge der Zustände (states)
 - s_0 ...Initialzustand (initial state)
 - $\delta: S \times I \rightarrow S$...Zustandsübergangsfunktion (transition relation)
 - $\lambda: S \times I \rightarrow O$...Ausgabefunktion (output function)

Zustandsdiagramm

System befindet sich immer in exakt einem Zustand

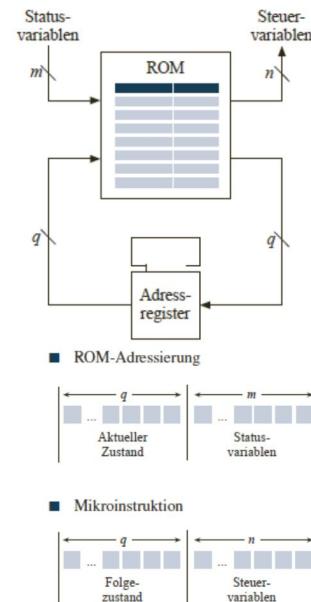


- Naheliegende Realisierung des Automaten:
 - Jeder Zustand entspricht einem Flipflops
 - Alle FF arbeiten mit dem gleichen Takt
 - Genau ein FF ist 1, alle anderen 0
(*One-Hot* oder 1-aus-N Kodierung)
 - 1 „wandert“ zwischen den Flipflops
 - Wechsel nur wenn entsprechende Bedingung erfüllt ist
(Und-Gatter als Tor-Schaltung)

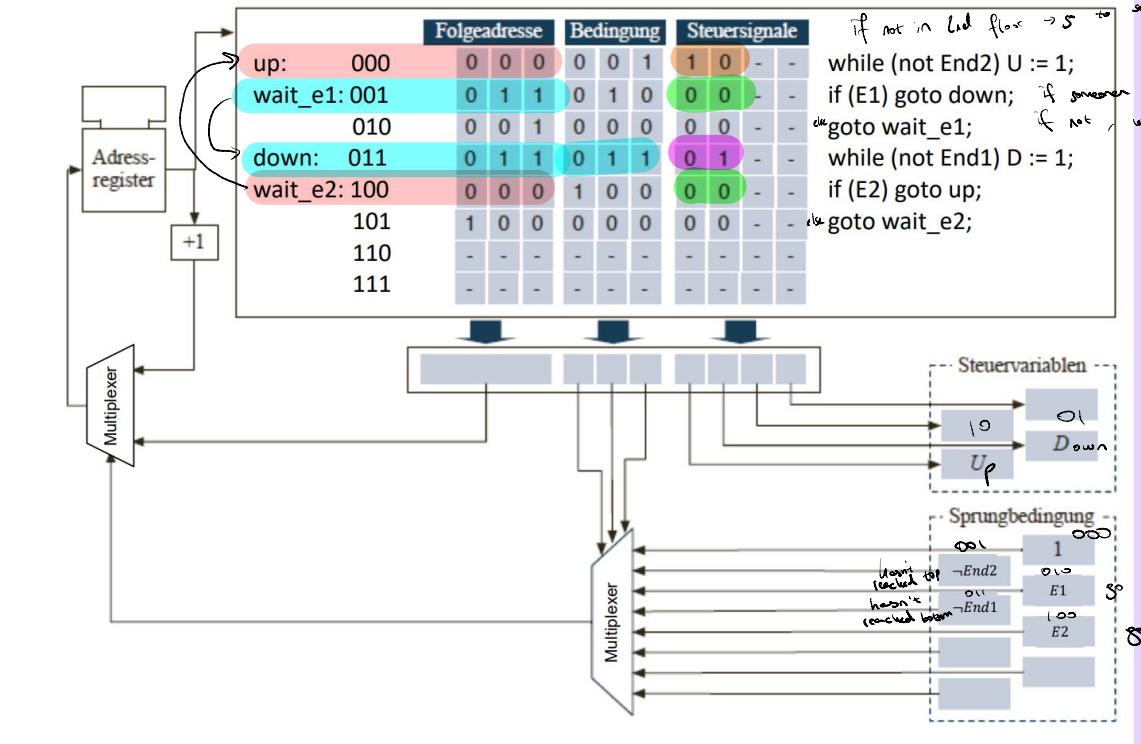


Mikroprogrammierte Steuerwerke

- Strukturierte Implementierung von Automaten
- Realisierung der Zustandsübergangsfunktion durch einen Speicher
- Adresse wird gebildet aus
 - Zustand Z (q Bit)
 - Eingangssignale (Statusvariablen) X (m Bit)
- Ausgang liefert Mikroinstruktion
 - Folgezustand Z' (q Bit)
 - Ausgangssignale (Steuervariablen) Y (n Bit)
- Größe des Speichers: $2^{(q+m)*(q+n)}$ Bit

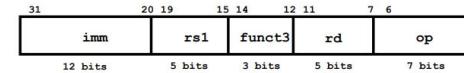


Aufzugsteuerung



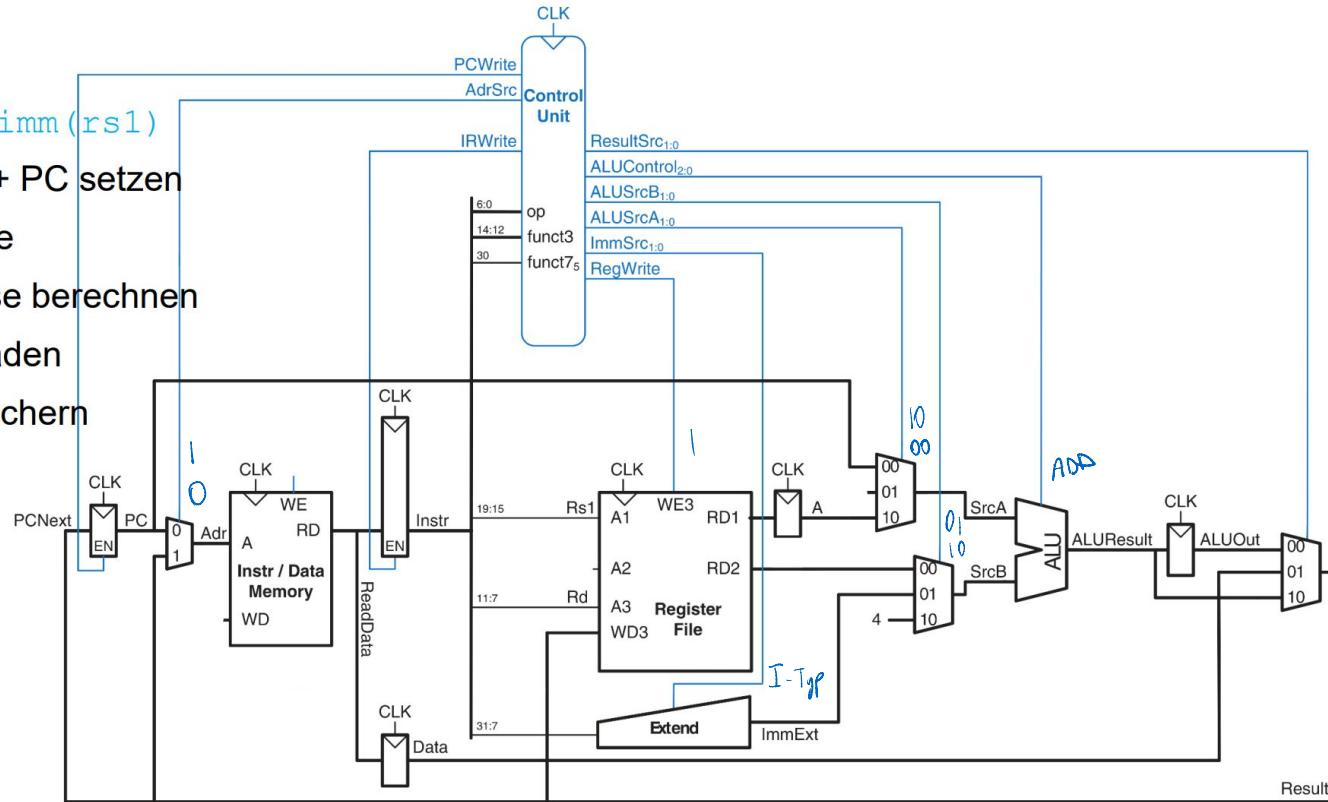
second floor
if not in last floor → S to
while (not End2) U := 1;
if (E1) goto down; if greater than or equal to S, go down
else goto wait_e1; if not, wait until they want to
while (not End1) D := 1;
if (E2) goto up;
else goto wait_e2;

Befehl Load Word

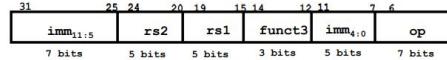


`lw rd, imm(rs1)`

- Fetch + PC setzen
- Decode
- Adresse berechnen
- Wert laden
- Abspeichern

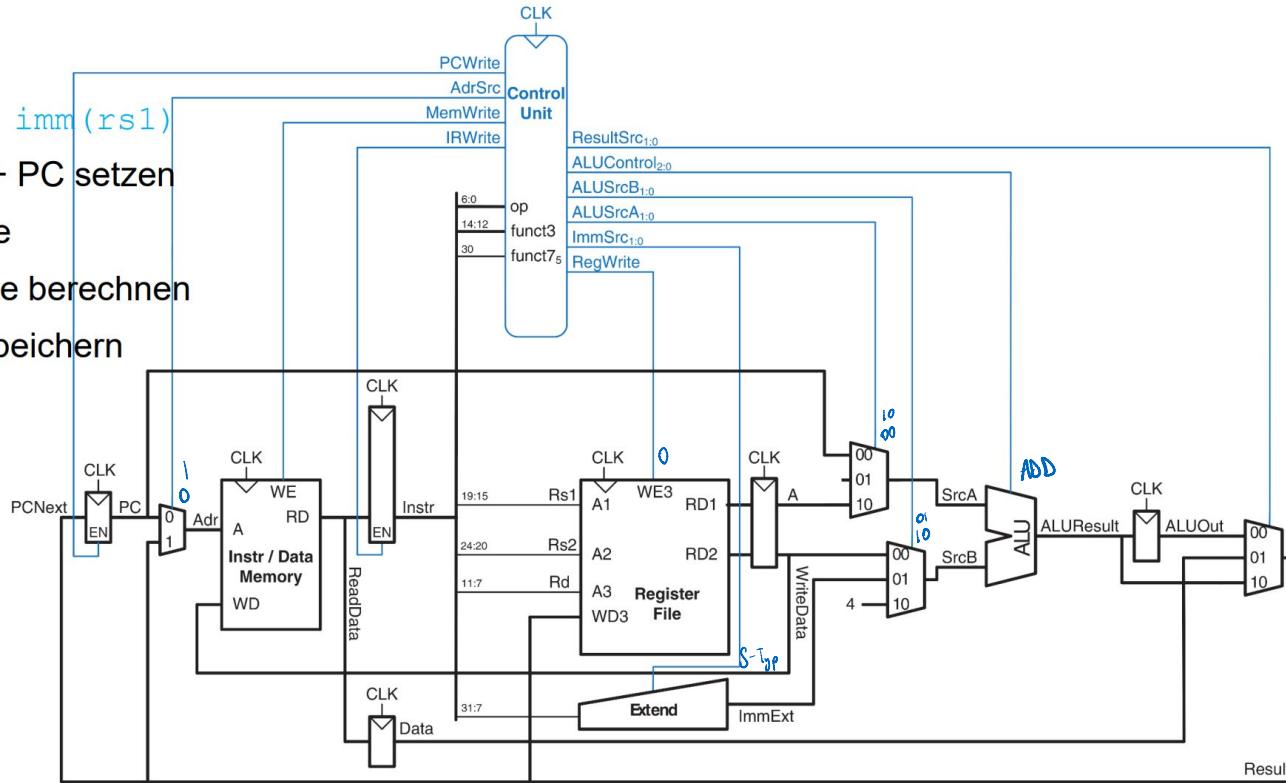


Befehl Store Word

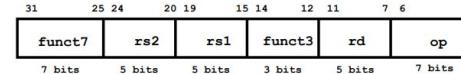


sw rs2, imm(rs1)

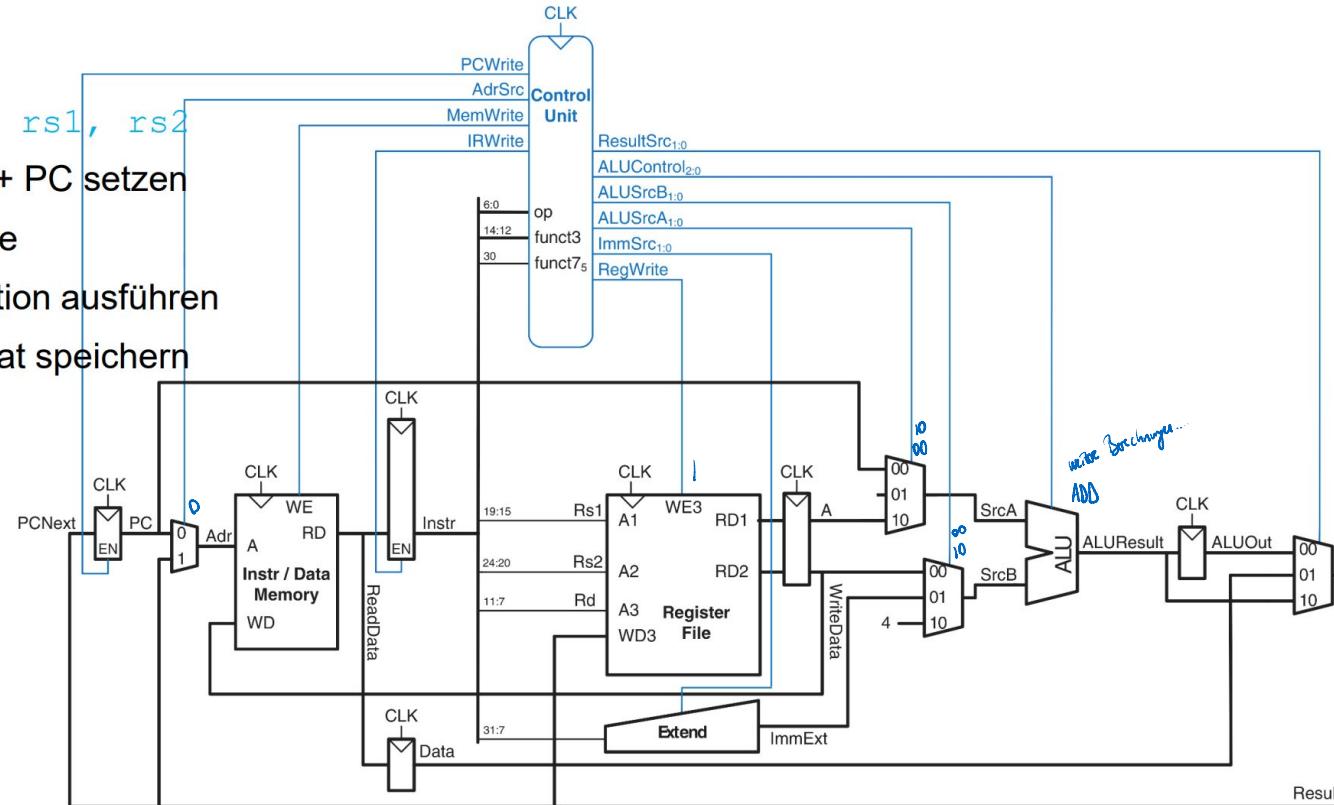
- Fetch + PC setzen
- Decode
- Adresse berechnen
- Wert speichern



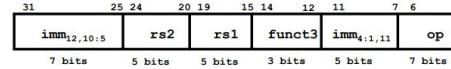
R-Typ Befehle



- add rd, rs1, rs2
- Fetch + PC setzen
 - Decode
 - Operation ausführen
 - Resultat speichern

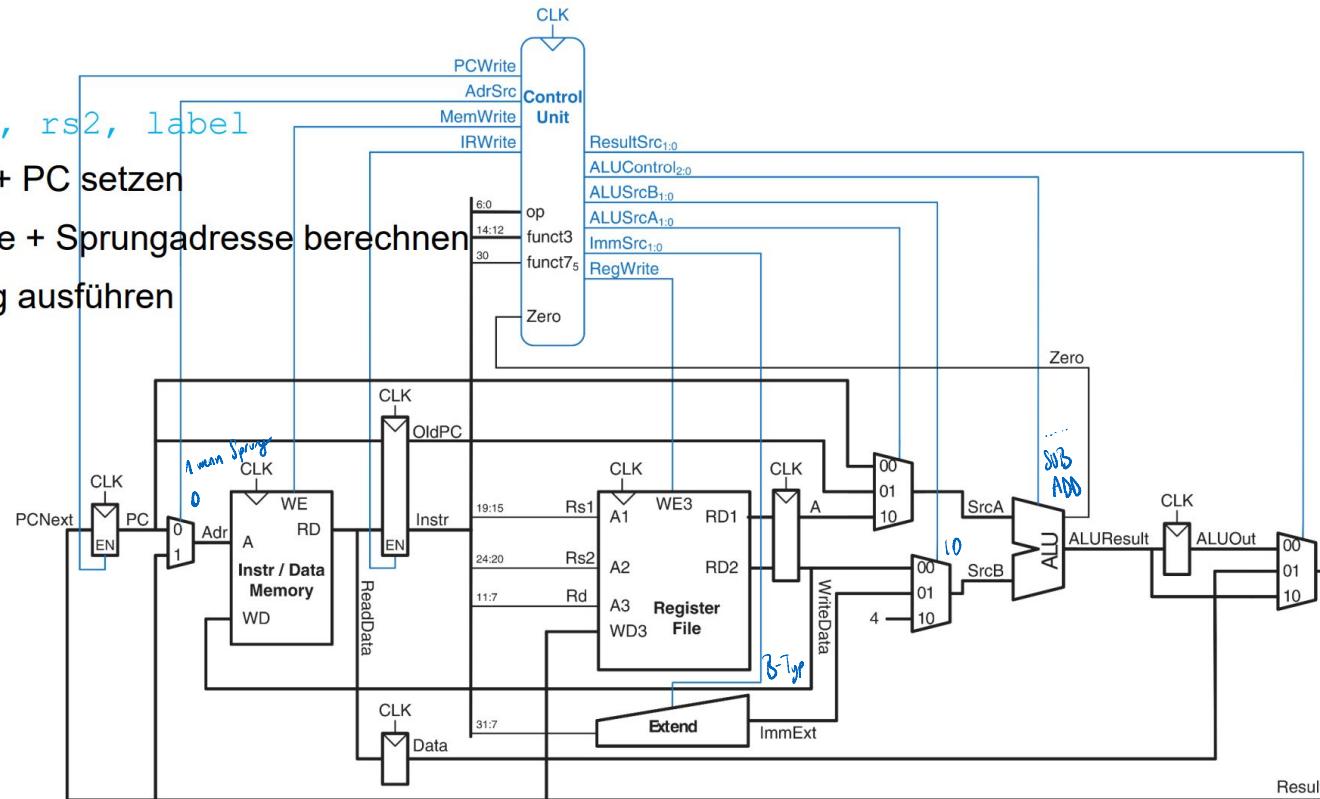


Branch Befehle

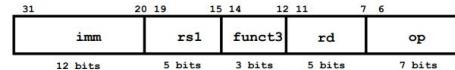


`beq rs1, rs2, label`

- Fetch + PC setzen
- Decode + Sprungadresse berechnen
- Sprung ausführen

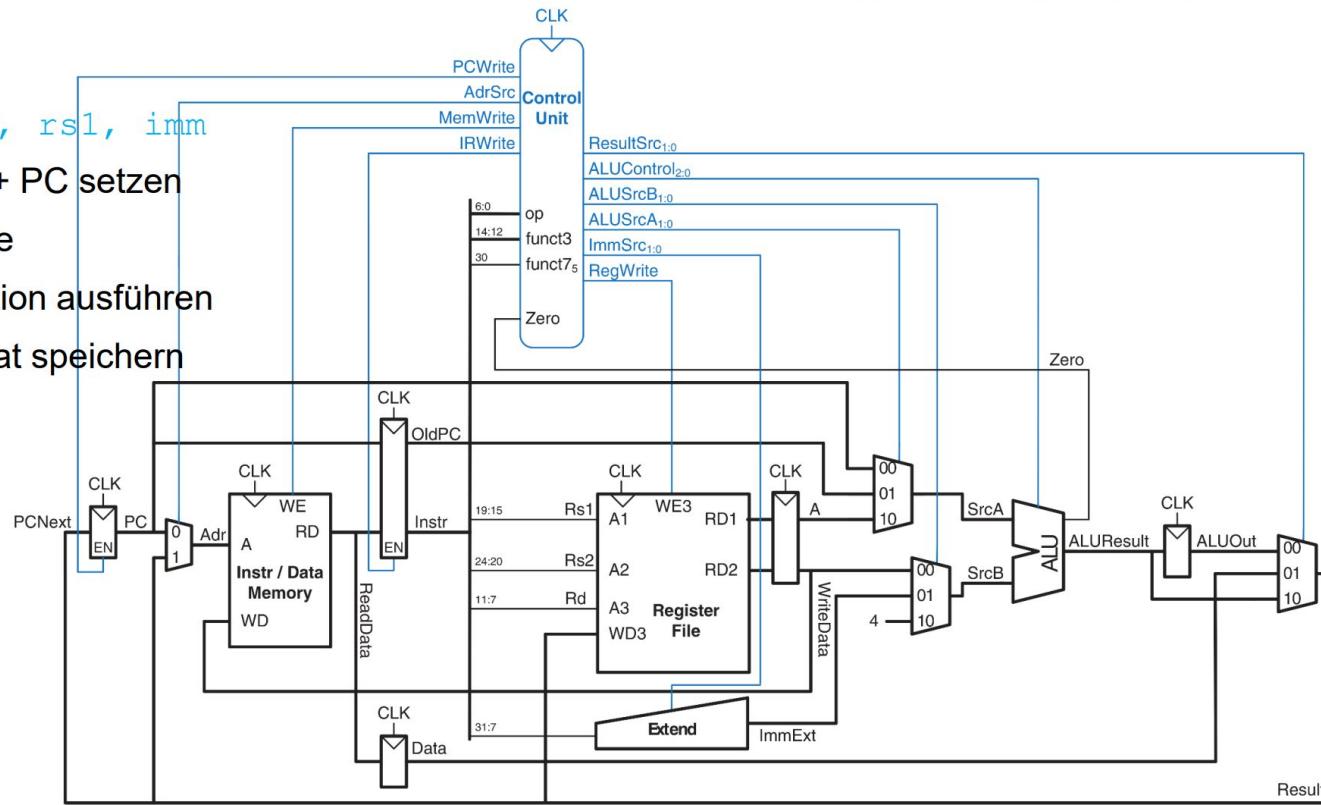


I-Type Befehle



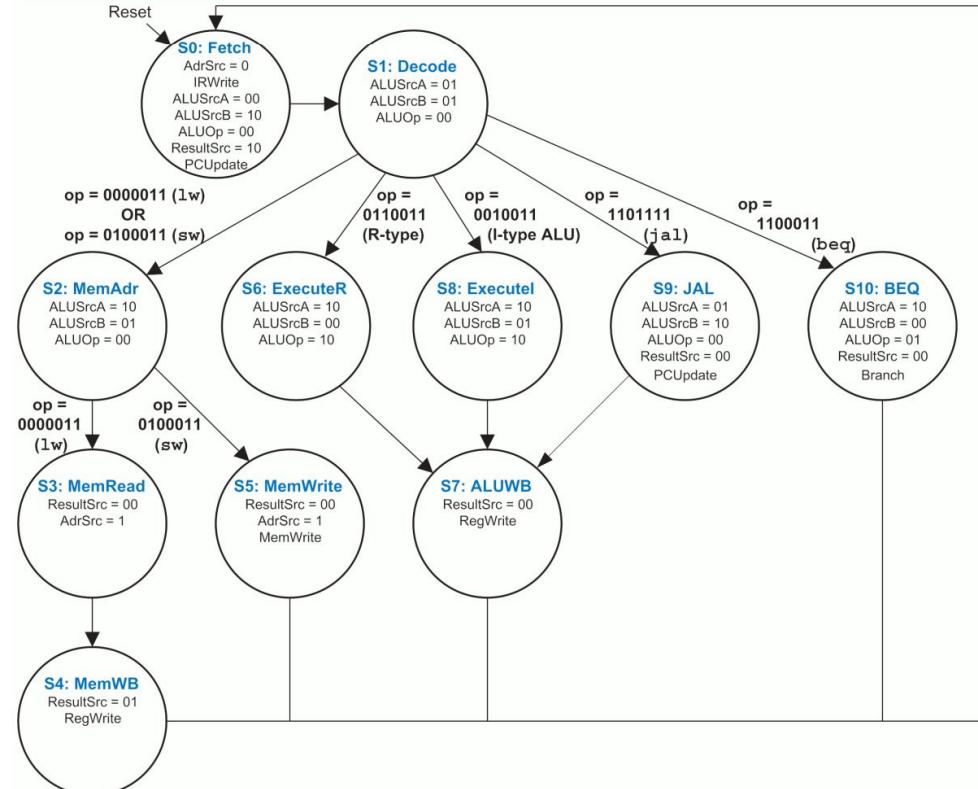
addi rd, rs1, imm

- Fetch + PC setzen
- Decode
- Operation ausführen
- Resultat speichern



Multicycle: Vollständiges Seq. Steuerwerk

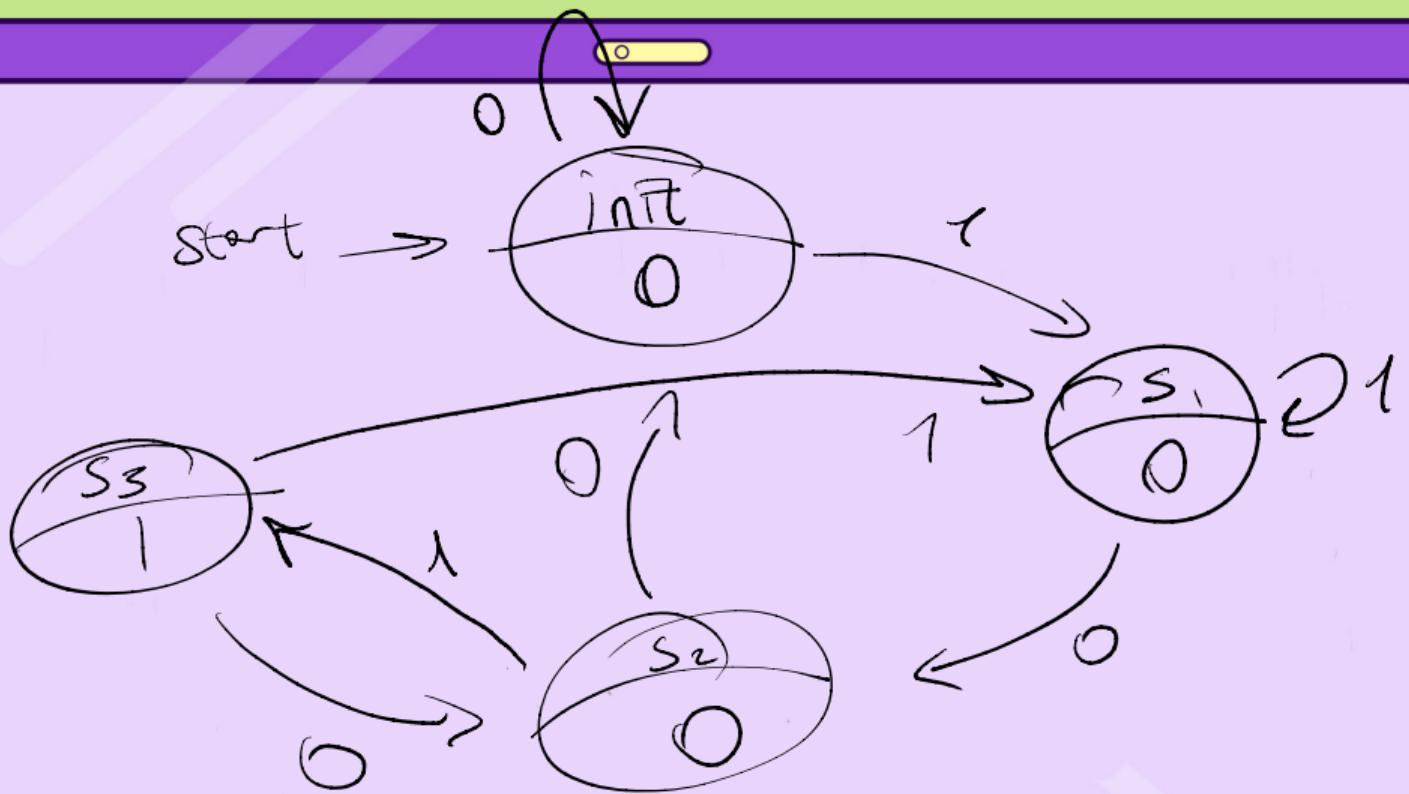
Zustand	Datenpfad
Fetch	Instr \leftarrow Mem[PC]; PC \leftarrow PC + 4
Decode	ALUOut \leftarrow PCTarget
MemAdr	ALUOut \leftarrow rs1 + imm
MemRead	Data \leftarrow Mem[ALUOut]
MemWB	rd \leftarrow Data
MemWrite	Mem[ALUOut] \leftarrow rd
ExecuteR	ALUOut \leftarrow rs1 op rs2
Executel	ALUOut \leftarrow rs1 op imm
ALUWB	rd \leftarrow ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC = ALUOut
JAL	PC = ALUOut; ALUOut = PC+4



1 Pattern Recognizer

Entwirf eine Schaltung, welche die Bitfolge 101 in einem Bitstrom I erkennt. Die Schaltung sollte dazu immer $O = 1$ ausgeben, wenn die Bitfolge aufgetreten ist. Ansonsten sollte $O = 0$ ausgegeben werden. Führe dazu folgende Schritte durch:

- Entwirf das Zustandsdiagramm als Moore-Automat.



- b) Beschreibe den endlichen Automaten formal als 6-Tupel $A(I, O, S, s_0, \delta, \lambda)$, wobei I die Menge der Eingänge, O die Menge der Ausgänge, S die Menge der Zustände, s_0 den Anfangszustand, δ die Übergangsfunktion und λ die Ausgabefunktion bezeichnet.

$$I = \{0, 1\}$$

$$O = \{0, 1\}$$

$$S = \{\text{init}, s_1, s_2, s_3\}$$

$$\lambda(\text{init}) = 0 \quad \lambda(s_2) = 0$$

$$\lambda(s_1) = 0 \quad \lambda(s_3) = 1$$

$$\delta : S \times I \rightarrow S$$

δ	0	1
init	init	s_1
s_1	s_2	s_1
s_2	init	s_3
s_3	s_2	s_1

c) Überführe das Zustandsdiagramm in eine Wahrheitstabelle und anschließend in Boolesche Terme.

$$TAT = 00 \quad S_1 = 01 \quad S_2 = 10 \quad S_3 = 11$$

I	FF1	FF0	FF1'	FF0'	O
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	1	1	1

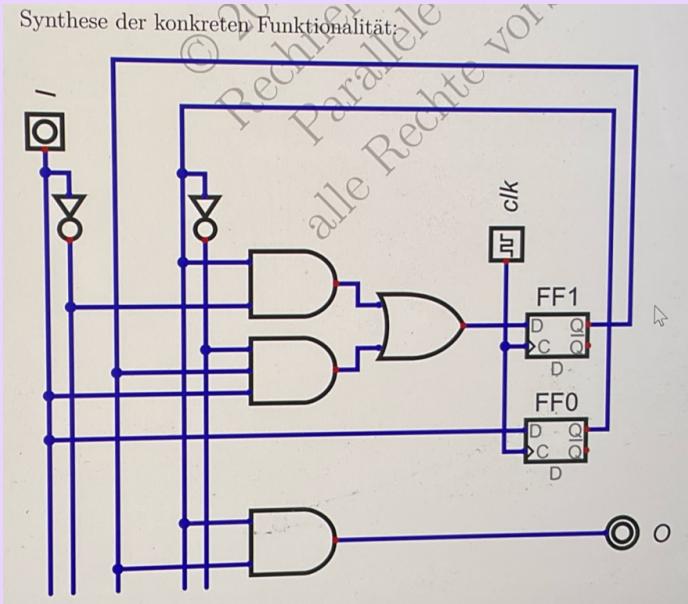
$$\bar{I} \cdot \bar{FF1} \cdot \bar{FF0}$$

$$FF1' = \bar{I} \cdot \bar{FF1} \cdot \bar{FF0} + \bar{I} \cdot FF1 \cdot \bar{FF0} + I \cdot FF1 \cdot \bar{FF0}$$

$$FF0' = \bar{I} \cdot \bar{FF1} \cdot \bar{FF0} + I \cdot \bar{FF1} \cdot \bar{FF0} + \bar{I} \cdot FF1 \cdot \bar{FF0} + I \cdot FF1 \cdot \bar{FF0} = I$$

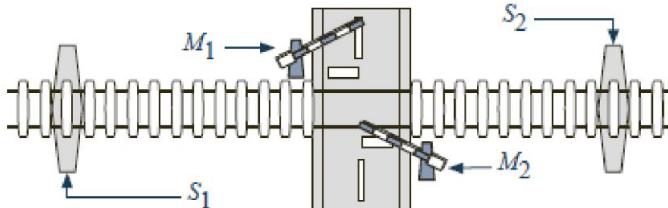
$$O = FF1 \cdot FF0$$

- d) Entwirf die entsprechende Schaltung in Digital, einmal Synthese der konkreten Funktionalität und einmal als mikroprogrammiertes Steuerwerk.



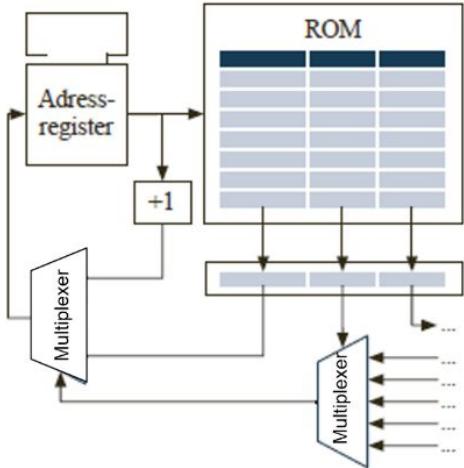
2 Bahnübergang

Die Aufgabe ist es, die Steuerung des folgenden Bahnübergangs zu automatisieren:

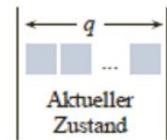


Auf der eingleisigen Strecke sind zwei binäre Sensoren S1 und S2 eingebaut, die genau dann den Wert 1 liefern, wenn sich ein Zug über ihnen befindet. Als Ausgänge stehen die beiden Steuervariablen M1 und M2 zur Verfügung, mit deren Hilfe sich die beiden Bahnschranken öffnen und schließen lassen. Ist M1 bzw. M2 gleich 1, so schließt sich die obere bzw. die untere Schranke und verharrt in diesem Zustand. Wechselt M1 bzw. M2 von 1 auf 0, so öffnet sich die obere bzw. untere Schranke und bleibt in diesem Zustand, bis das entsprechende Signal wieder auf 1 wechselt.

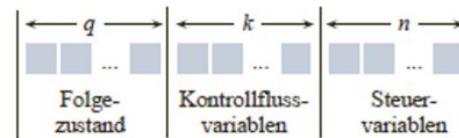
- Entwirf ein adressmodifizierendes mikroprogrammiertes Steuerwerk zur Steuerung der Bahnschranken. Verwende hierzu die konkrete Ausgestaltung aus der Vorlesung.



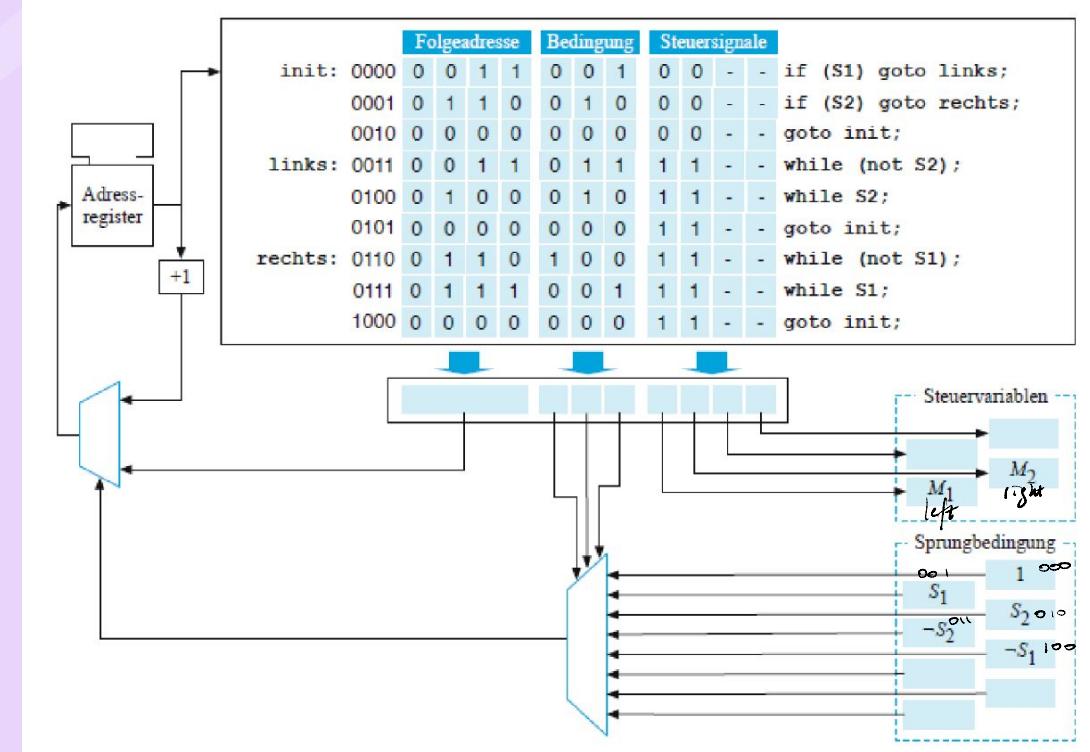
■ ROM-Adressierung



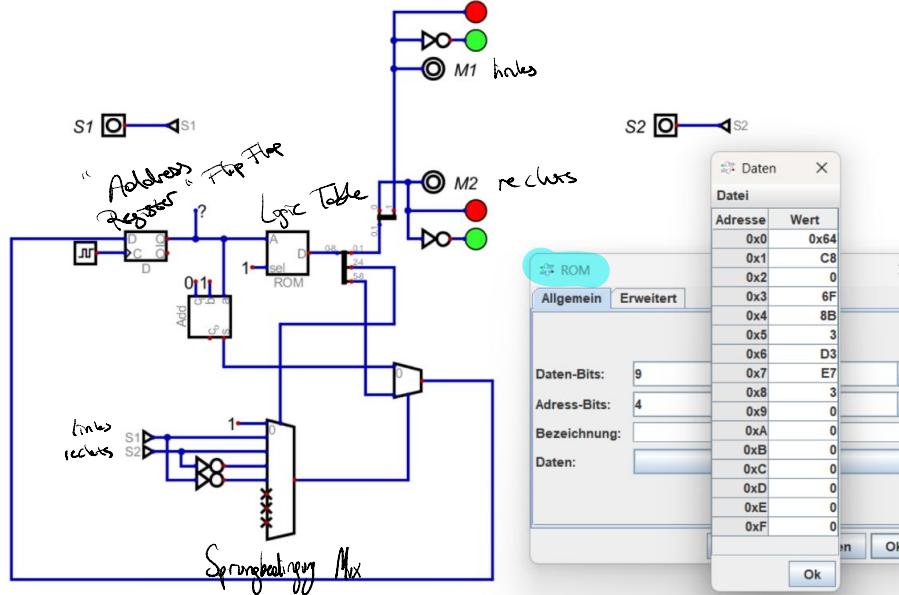
■ Mikroinstruktion



Hinweis: Es darf davon ausgegangen werden, dass der gezeigte Gleisabschnitt von maximal einem Zug gleichzeitig belegt wird. Beachte aber, dass das Gleis in beide Richtungen befahren werden kann.



b) Entwirf die entsprechende Schaltung in Digital.





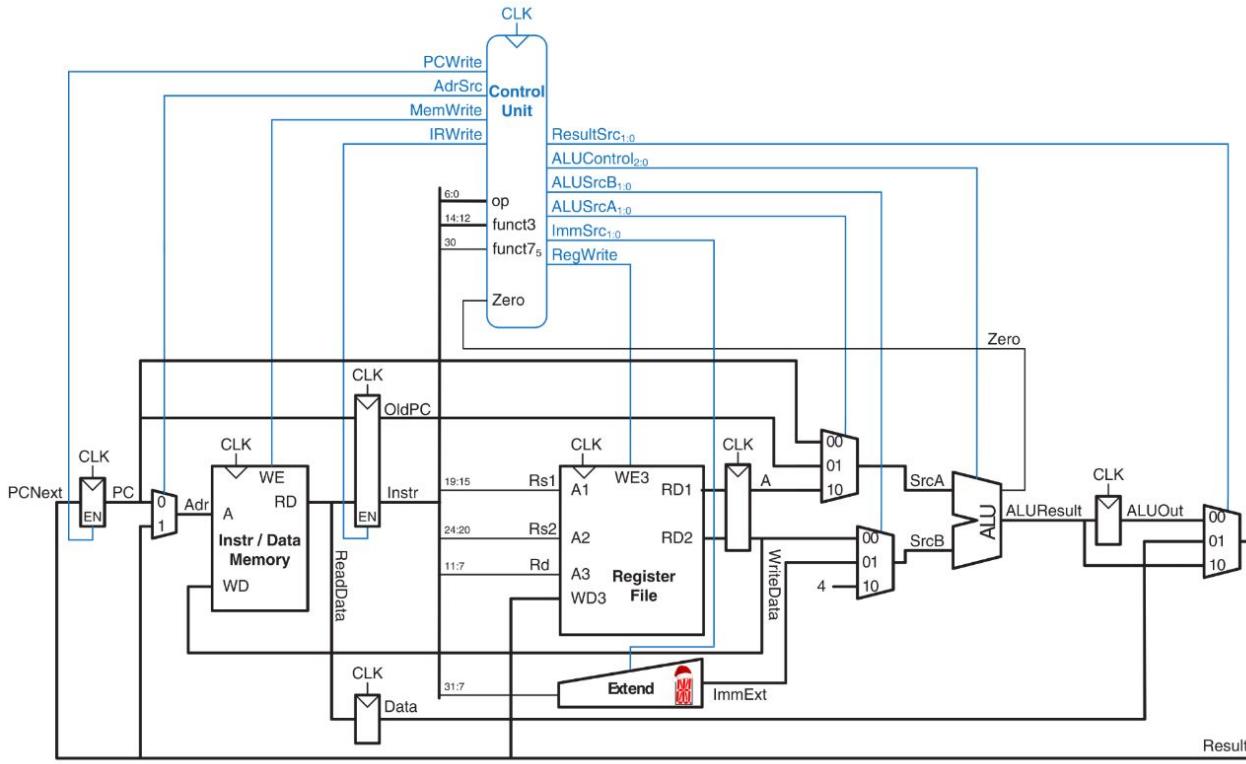
- c) Funktioniert dein Entwurf auch, wenn die Annahme, dass der gezeigte Gleisabschnitt von maximal einem Zug gleichzeitig belegt wird, entfällt?

Nein. Entfällt die Annahme, so ist das Problem ungleich schwerer zu lösen. Um korrekt zu entscheiden, wann die Schranke wieder geöffnet werden darf, muss ein Zähler eingefügt werden, der Auskunft über die Anzahl der Züge gibt, die sich zwischen den beiden Sensoren befinden. Eine Implementierung mit Hilfe eines einfachen mikroprogrammierten Steuerwerks ist dann nicht mehr möglich.

4 Prozessorerweiterung

Der Prozessor in Abbildung 3 soll erweitert werden sodass er auch den Befehl `lui` ausführen kann.

- Erweiter den Prozessor, sodass er `lui` durchführen kann. Falls einzelne Komponenten ebenfalls angepasst werden müssen, so reicht es aus in Worten oder als Tabelle anzugeben, wie sich das Verhalten ändert.

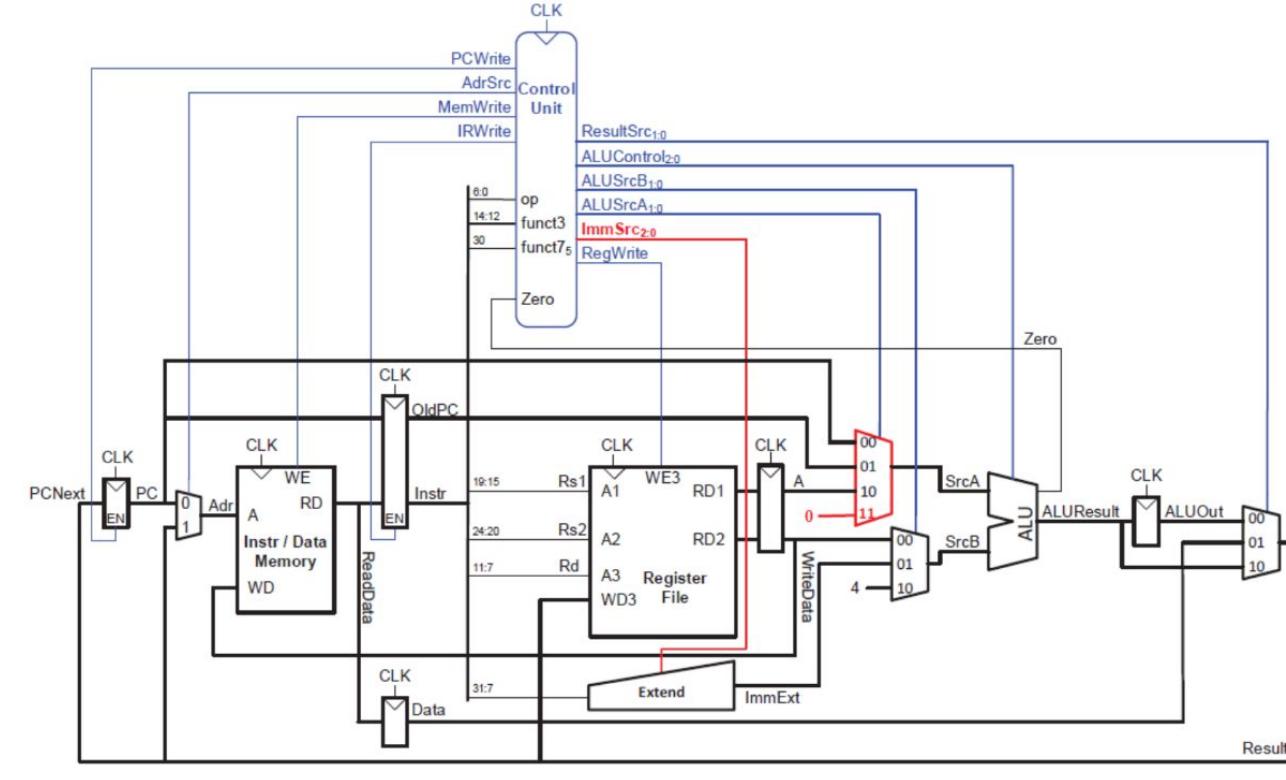


Zunächst muss die Extend-Unit angepasst werden, sodass diese auch U-Typ Instruktionen (z.B. `lui`) erlaubt.

Enhanced *ImmSrc* encoding to support `lui`

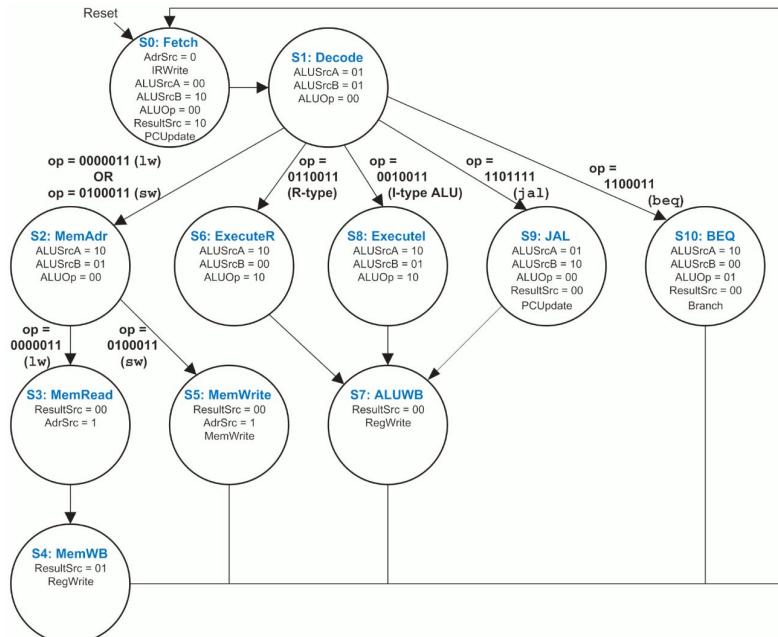
<i>ImmSrc</i>	<i>ImmExt</i>	Type	Description
000	{ {20 { <i>Instr</i> [31]}}, <i>Instr</i> [31:20] }	I	12-bit signed immediate
001	{ {20 { <i>Instr</i> [31]}}, <i>Instr</i> [31:25], <i>Instr</i> [11:7] }	S	12-bit signed immediate
010	{ {20 { <i>Instr</i> [31]}}, <i>Instr</i> [7], <i>Instr</i> [30:25], <i>Instr</i> [11:8], 1'b0 }	B	13-bit signed immediate
011	{ {12 { <i>Instr</i> [31]}}, <i>Instr</i> [19:12], <i>Instr</i> [20], <i>Instr</i> [30:21], 1'b0 }	J	21-bit signed immediate
100	<i>Instr</i> [31:12], 12'b0	U	20-bit signed immediate

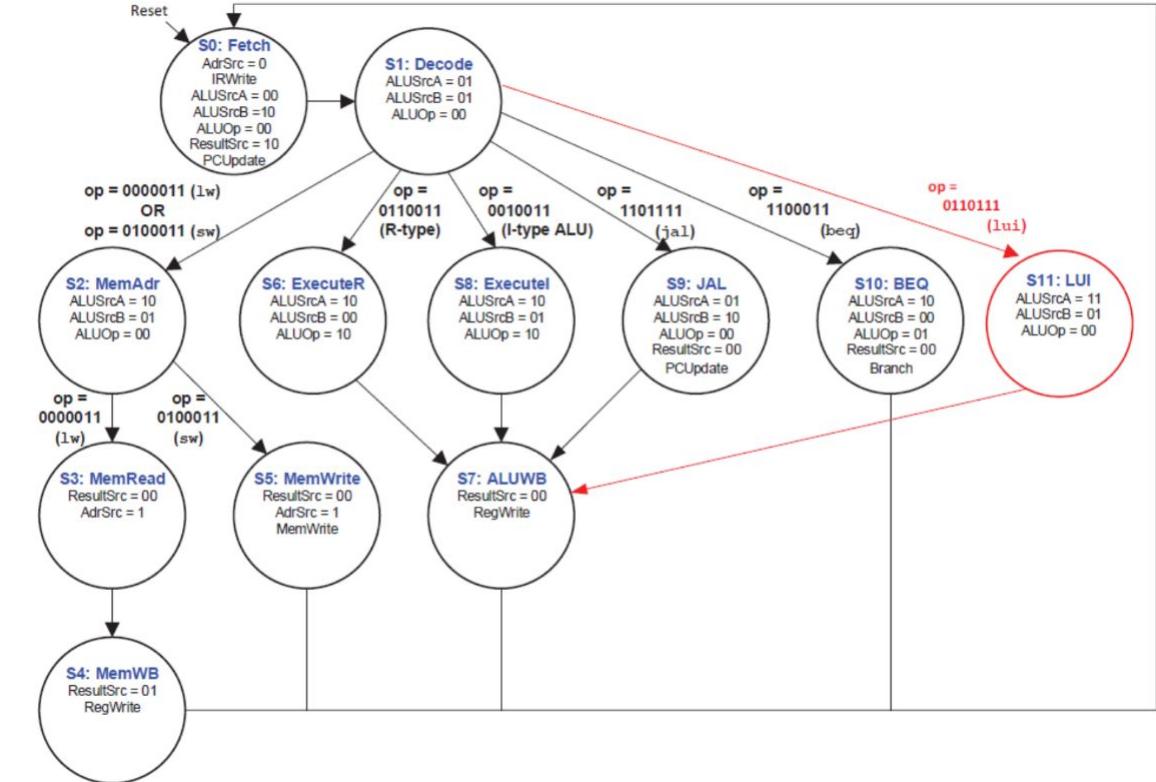
Hierdurch wird das ImmSrc-Signal auf 3bit erweitert. Außerdem ändern wir das Schaltbild so, dass auch 0 als oberer Eingang für die ALU möglich ist, denn wir wollen das Immediate unverändert durch die ALU durchleiten.



HA Vorbereitung

Passe den Automaten des sequentiellen Steuerwerks in Abbildung 4 ebenfalls an. Falls neue Zustände und Signale eingeführt oder verändert werden, beachte, dass zusätzliche Signale eventuell in anderen Zuständen ebenso berücksichtigt werden müssen.



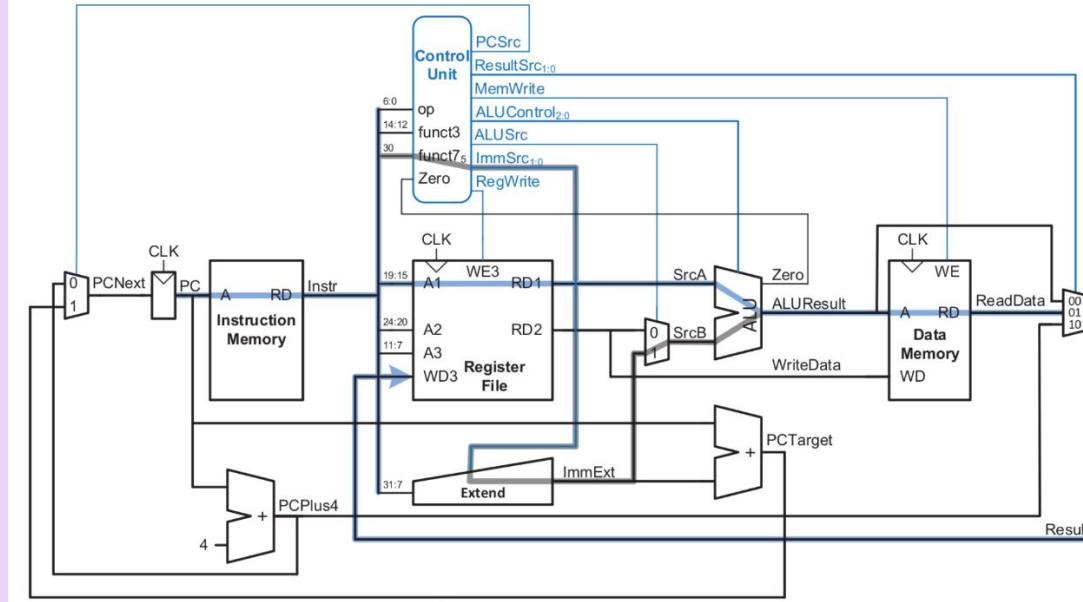


3 Prozessor Performance

Nach Anlegen eines Inputs braucht es eine gewisse Zeit bis eine Veränderung an den Outputs eines digitalen Bauteils sichtbar ist. Der Delay ist die Zeit die ein Input braucht um verarbeitet zu werden. Folgende Tabelle zeigt Delays einzelner Bauteile die im RISC-V Prozessor verwendet werden. Delays für Bauteile die hier nicht angegeben sind, dürfen als 0 angenommen werden.

- Bestimme die minimale Taktlänge T_{c_single} des Single-Cycle RISC-V Prozessors.

Der Befehl mit dem längsten kritischen Pfad ist der **lw** Befehl. Der kritische Pfad des **lw** ist in folgender Abbildung gegeben.



Element	Parameter	Delay (ps)
Register read	t_{RegRead}	40
Register setup	t_{RegSetup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{\text{AND-OR}}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{MemRead}	200
Register file read	t_{RFRead}	100
Register file setup	t_{RFSetup}	60

$$T_{\text{c_single}} = t_{\text{RegRead}} + t_{\text{mem}} + \max[t_{\text{RFRead}}, t_{\text{dec}} + t_{\text{ext}} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{\text{RFsetup}}$$

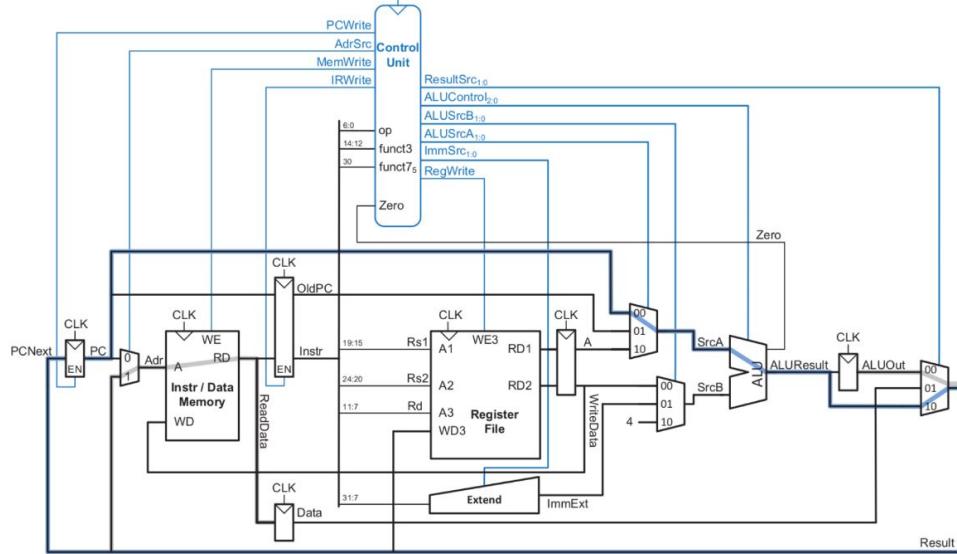
Der kritische Pfad verläuft durch die Registerbank und nicht durch die Control Unit. Das heißt die Gleichung vereinfacht sich wie folgt:

$$T_{\text{c_single}} = t_{\text{RegRead}} + 2t_{\text{mem}} + t_{\text{RFRead}} + t_{\text{ALU}} + t_{\text{mux}} + t_{\text{RFsetup}}$$

Mit den Zahlen aus Tabelle 1 ergibt sich also eine Taktrate von $T_{\text{c_single}} = 750\text{ps}$

- Bestimme die minimale Taktlänge T_{c_multi} des Multicycle RISC-V Prozessors.

zwei potentielle kritische Pfade im Prozessor die als blaue und graue Linien in der folgenden Abbildung eingezeichnet sind:



$$T_{c_multi} = t_{RegRead} + t_{dec} + 2t_{mux} + \max[t_{ALU}, t_{mem}] + t_{RegSetup}$$

Mit den Delays aus Tabelle 1 ergibt sich also $T_{c_multi} = 375\text{ps}$.

- Angenommen der Prozessor führt ein Programm aus, das aus 25% `lw`, 10% `sw`, 11% `beq`, 2% `jal` und 52 % R- oder I-Typ ALU Instruktionen besteht¹. Was ist die vorraussichtliche Laufzeit wenn das Programm aus 10^{11} Instruktionen besteht. Berechne das Ergebnis sowohl für den Single-Cycle RISC-V als auch den Multicycle RISC-V.

Beim Single-Cycle Prozessor dauert jeder Befehl genau 1 Taktzyklus. Da das gegebene Programm dann 10^{11} Taktzyklen benötigt, ergibt sich eine Laufzeit von $10^{11} \cdot 750\text{ps} = 75\text{s}$.

Beim Multicycle Prozessor benötigen verschiedene Befehle unterschiedlich lange. Die Anzahl der Taktzyklen die ein Befehl benötigt kann aus Abbildung 4 entnommen werden. Damit ergeben sich folgende Taktzyklen um verschiedene Befehle auszuführen:

- **lw:** 5
- **sw:** 4
- **beq:** 3
- **jal:** 4
- **R- und I-Typ:** 4

Für das gegebene Programm ergibt sich damit eine durchschnittliche Länge von $0.25 \cdot 5 + (0.1 + 0.02 + 0.52) \cdot 4 + 0.11 \cdot 3 = 4.14$ Taktzyklen pro Befehl und damit eine Laufzeit von $10^{11} \cdot 4.14 \cdot 375\text{ps} = 155\text{s}$

Das Delay eines der verwendeten Komponenten soll verringert werden um die Taktrate des Multicycle RISC-V Prozessors zu beschleunigen. Welches Bauteil sollte neu entwerfen werden um die größte Verbesserung zu erhalten? Wie schnell (Delay in ps) sollte das Bauteil dann mindestens sein, also ab welchem Delay bewirken Verbesserungen nichts mehr? Wie ändert sich die Laufzeit des Programms aus der vorigen Aufgabe mit den Anpassungen?

Die größte Verbesserung kann durch Beschleunigen des lesenden Zugriffs auf den Speicher erzielt werden. Die maximale Verbesserung die erzielt werden kann ist 80ps (von 200ps auf 120ps). Den Speicher schneller als die ALU zu machen, bringt keine weitere Beschleunigung.

Die neue Taktlänge ist 295ps. Damit ergibt sich eine Laufzeit von $10^{11} \cdot 4.14 \cdot 295\text{ps} = 122\text{s}$.

Viel Glück

bei PGDP!