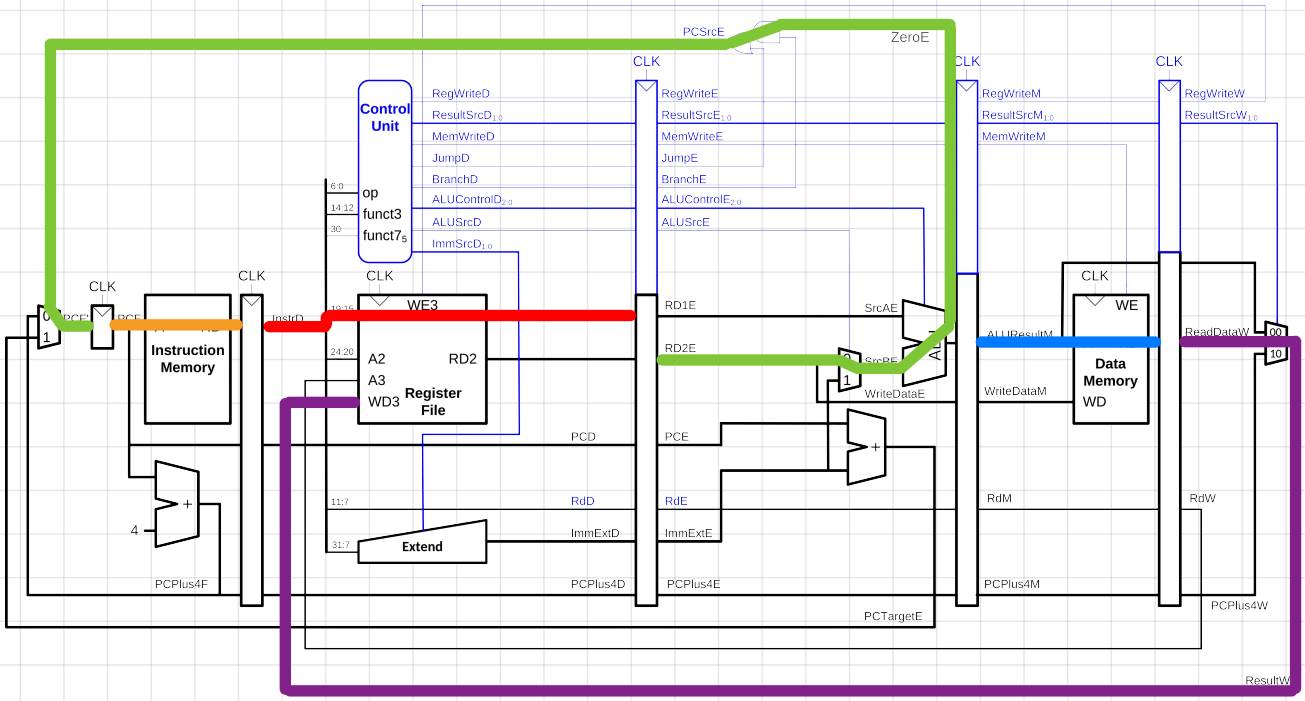
ERA – Übungsblatt 09

1. a)



Oben eingezeichnet sind die kritischen (längsten) Pfade für jede Pipelinestufe. Zum Bestimmen der Pfadlänge addieren wir die respektiven Zeiten aus der gegebenen Tabelle.

Fetch: $t_{RegRead} + t_{MemRead} + t_{RegSetup} = 230ps$

Decode: $t_{RegRead} + t_{RFRead} + t_{RegSetup} = 180ps$

Execute: $t_{RegRead} + t_{mux} + t_{ALU} + t_{two\text{-}or} + t_{mux} + t_{RegSetup} = 230ps$

Memory: $t_{RegRead} + t_{MemRead} + t_{RegSetup} = 230ps$

Writeback: $t_{RegRead} + t_{mux} + t_{RFSetup} = 180ps$

$t_{RegRead}$ und $t_{RegSetup}$ stehen jeweils für das Lesen bzw. Schreiben in die Registerbanktile, welche die Signale zwischen den Pipelinestufen weiterleiten. Daher sind sie auch Bestandteil einer jeden Stufe bis auf Writeback, weil dort nach dem Schreiben in die Registerbank der Pfad zu Ende ist.

b) Wir wählen den Takt unseres pipelined Prozessor so, dass die längsten Pfade noch hineinpassen.

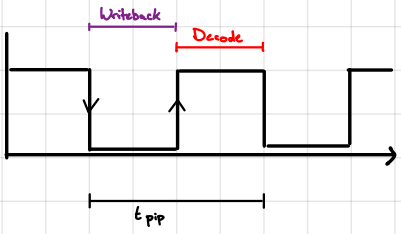
Also $t_{pip} = max[t_{Decode}, t_{Execute}, \dots] = 230ps$

Theoretisch kann ein Speedup von #Pipelinestufen, also 5 erreicht werden, weil wir 5 Instruktionen parallel verarbeiten können.

$$Speedup = \frac{t_{sc}}{t_{pip}} = \frac{750ps}{230ps} = 2{,}59$$

Wir erreichen nur ein kleineres Speedup, weil die Stufen nicht gleich lange sind und somit einige „schnelle" Stufen (bspw. Writeback) auf „langsame" Stufen (bspw. Memory) warten müssen. Ein Prozessor mit 100 Stufen hätte also theoretisch einen Speedup von 100, dieser würde in der Praxis aber nicht erreicht werden, da das Aufteilen in 100 gleich lange Stufen schwer umsetzbar ist und zusätzlichen Overhead ($t_{RegRead}$ und $t_{RegSetup}$) bedeuten würde.

c) Unser Clock-Signal schaut wie folgt aus:
Wenn die Registerbank bei fallender Flanke geschrieben und bei steigender Flanke gelesen wird, so beeinflusst das die Phasen Writeback und Decode. Beide müssen dann nämlich schon in einem halben Taktzyklus fertig sein. Wir müssen also unsere Taktlänge anpassen, da $2 \cdot (t_{RFRead} + t_{RegSetup})^* = 300\,ps > 280\,ps$.
Also $t_{pip}' = 300ps$.



* $t_{RegRead}$ ist von der Modifikation unbeeinflusst und kann schon in der ersten Taktthälfte erledigt werden

2. siehe ML

3. a) Abhängigkeiten:
- s1, s2 bzgl. t1
- s2, s3 bzgl. t0
- s2, s4 bzgl. t0 (hinfällig weil redundante Abhängigkeit behandelt wird)
- s2, s5 bzgl. t0 (später hinfällig wegen NOP)
- s3, s4 bzgl. t3
- s3, s5 bzgl. t3 (hinfällig)
- s3, s6 bzgl. t3 (hinfällig)
- s5, s6 bzgl. t3

Takt	F	D	E	M	WB
1	s1	–	–	–	–
2	s2	s1	–	–	–
3	s3	s2	s1	–	–
4	NOP	s3	s2	s1	–
5	s4	NOP	s3	s2	s1
6	s5	s4	NOP	s3	s2
7	s6	s5	s4	NOP	s3
8		s6	s5	s4	NOP
9			s6	s5	s4
10				s6	s5
					s6

Forward t1 Memory → Execute
Forward t0 Memory → Execute
Forward t3 Writeback → Execute (Ergebnis von Lw erst nach Memory verfügbar)
Forward t2 Memory → Execute

b) Aus der Tabelle ist ersichtlich, dass 7 Taktzyklen benötigt werden, bis alle Instruktionen gelaufen wurden (≙ alle Instruktionen waren/sind in Fetch)

c) siehe ML