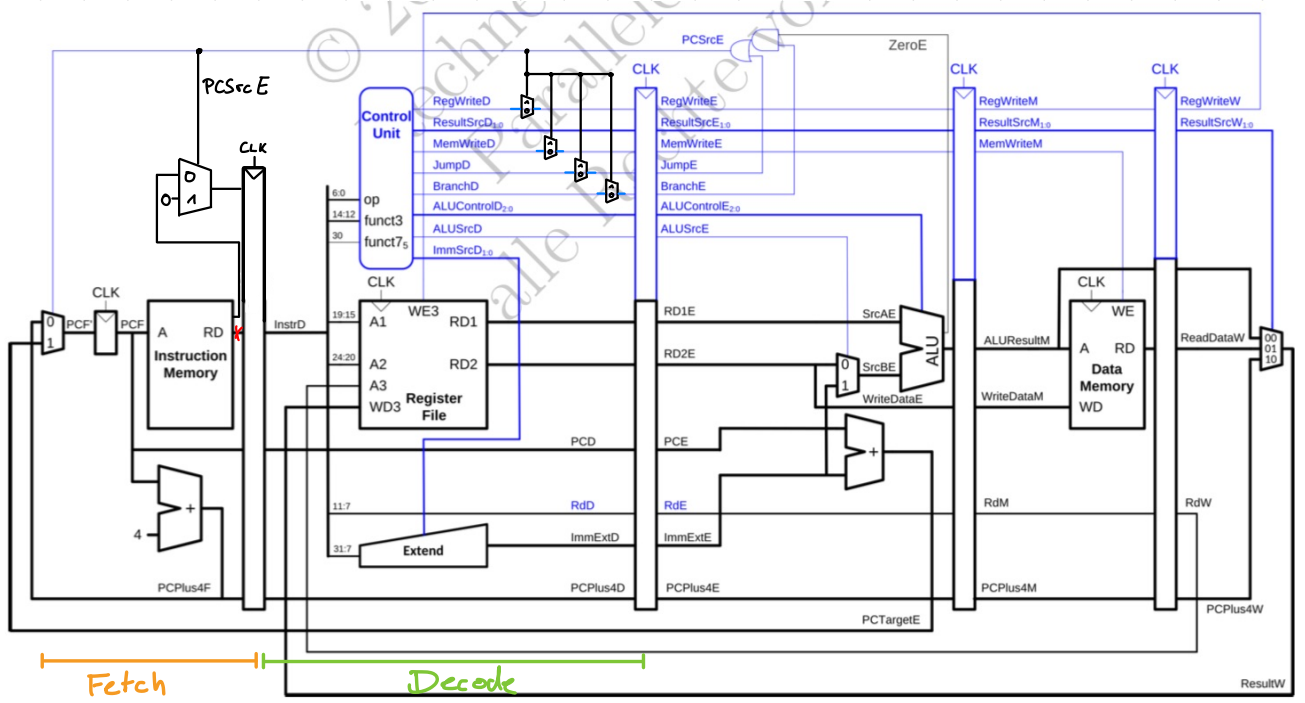
ETA - Übungsblatt 11

1. a,b,c) siehe ML und Mitschrift WS 23/24

2.



Wir sollen die Flush-Funktionalität einer Hazard Unit für Jumps und Branches implementieren. Ob gesprungen wird, ist allerdings erst in der Execute-Stage bekannt (Branches müssen die dazugehörige Bedingung überprüfen). Dann werden allerdings schon die nächsten zwei Instruktionen in Fetch und Decode geladen. Falls zu einer anderen Stelle im Programm gesprungen wird, dürfen diese nicht ausgeführt werden, d.h. sie müssen geflusht werden. Abhängig vom PCSrcE-Signal können wir diese Funktionalität implementieren:

- Fetch: In der Fetch-Stage können wir bei einem Sprung einfach ein NOP (32 Nullen) als nächste Instruktion weiterleiten. Dafür wird nur ein Multiplexer benötigt.
- Decode: In der Decode-Phase gestaltet sich das Flushen aufwendiger: Lediglich ein NOP basierend auf PCSrcE in InstrD hineinzuschalten würde den kritischen Pfad der Execute-Stage durch die Registerbank führen und würde daher einen weitaus längeren Taktzyklus erfordern (vgl. Aufgabe 1). Daher müssen wir auf andere Weise verhindern, dass die fälschlich geladene Instruktion Inhalte von Speicher/Registern ändert. Dies wird erzielt indem wir alle Signale, die eine solche Zustandsänderung hervorrufen können, explizit auf 0 setzen. Davon betroffen sind RegWriteD, MemWriteD, JumpD und BranchD.

3. a,b,c) siehe ML und Mitschrift WS 23/24