Benjamin Çoban 3526251
Michael Throner 4019521
Philipp Huber 3776454

Aufgabe 1.

a) Der Tomasulo Algorithmus implementiert Dynamic Scheduling auf Hardware Ebene. Der Algorithmus arbeitet in-order-issue, out-of-order execution and writeback. Die Ergebnisse werden direkt an die Funktionseinheiten übermittelt, an der Stelle kein Schreiben ins Register. Das *Dynamic Register Renaming* wird durch die **Reservation Stations** implementiert, für jede Funktionseinheit eine Station. Die Reservation Station dient als Entkopplung der Befehlsströme, um die Abhängigkeiten und Konflikte in den Griff zu kriegen. Der **Common Data Bus (CDB)** verbindet dabei die Funktionseinheiten mit dem Register und den Reservation Stations. Das Umbennenen des Registers erfolgt durch einen größeren Platz in den Stations als im Register, in welchem die Registeradressen dynamisch abgebildet werden.
Durch das Renaming gilt das *Single Assignment Prinzip*, welches den **WAW- & WAR-Konflikten** entgegenwirkt, somit entstehen in den Arten von Konflikten keine Wartezyklen mehr. Zum lösen eines WAW-Konflikts gehört dazu, zu entscheiden, wann man ein Ergebnis ins Register zurückschreiben darf.
Echte Datenabhängigkeiten, welche **RAW-Konflikte** auslösen, werden durch Ergebniszurückschreiben in andere Reservation Stations angegangen. Allerdings ist hier weiterhin mit Wartezyklen zu rechnen, da static single assignment Namens- und somit Schreibkonflikte löst, aber echte Datenabhängigkeiten weiterhin bestehen bleiben.

b)

Aufgabe 2.

Load und Store Befehle können nicht out-of-order ausgeführt werden, wenn es Datenabhängigkeiten über den Speicher gibt. Für ein Beispiel ist folgendes Listing zu betrachten:

```
1 LDR r1, 0x0001
2 LDR r2, 0x0001
3 ADD r3, r1, r2
4 STR r3, [r2,#8]
5 LDR r4, [r1,#8]
```

Im obigen Beispiel werden die ersten beiden Registereinträge mit einem immediate value geladen, der Hex 0x0001 (gleicher Wert, Thema unnötig viele Registereinträge). $r3$ ist die Summe aus $r1, r2$, welches dann in den Speicher geschrieben wird. Daraufhin wird aus eben der Speicherzelle $r4$ gelesen, was so aber nicht ersichtlich ist, da im Code keine synaktische, sondern eine rein semantische Abhängigkeit vorliegt.
Für Load und Store Befehle gibt es zwei Möglichkeiten, sie außerhalb der Programmreihenfolge starten zu lassen. Eine Möglichkeit ist ein **Load-Store Buffer**, eine spezielle Reservation Station, die immer in Programmreihenfolge abgearbeitet wird. Eine andere Möglichkeit wäre, die Adressen, die eventuell für Datenabhängigkeiten sorgen, zu berechnen und die Loads und Stores, die unabhängig sind, außerhalb der Programmreihenfolge laufen zu lassen. So oder so, gibt es Abhängigkeiten bei Load oder Store Befehlen, kann keine out-of-order execution stattfinden.