Universität Bremen AGRA Fachbereich 3 Mathematik / Informatik

Prof. Dr. Rolf Drechsler, drechsler@informatik.uni-bremen.de, MZH 4330
Christina Plump, cplump@uni-bremen.de, MZH 4206

3. Übungsblatt zur Vorlesung

Technische Informatik 1

```
S0: li   t3, a        # t2 := a (konstant)
S1: bgez t3, S3       # springe nach S3, falls t3 ≥ 0
S2: li   t2, b        # t2 := b (konstant)
S3: li   t4, c        # t4 := c
S4: rem  t3, t2, t4   # t3 := $t2 mod $t4
S5: add  t1, t1, t2   # t1 := $t1 + $t2
S6: mul  t4, t4, t1   # t4 := $t4 · $t1
```

Aufgabe 1 (3 Punkte)

Betrachtet das oben gegebene Assemblerprogramm unter den folgenden Annahmen:

a) a > 0 ist und

b) lediglich der Wert aus Register $t4$ in zeitlich auf S_6 folgenden Instruktionen verwendet wird.

Vereinfacht das Assemblerprogramm soweit wie möglich.

Aufgabe 2 (5 + 1 Punkte)

Unter der Annahme, dass in einem sequentiellen Programm mit n Anweisungen die Anweisung S_i vor der Anweisung S_j mit $0 \leq i < j \leq n-1$ steht, kann man *Datenabhängigkeiten* folgendermaßen klassifizieren:

- *True Dependence* (read after write): S_j liest eine Variable, die in S_i beschrieben wird.
- *Anti Dependence* (write after read): S_j schreibt auf eine Variable, die in S_i gelesen wird.
- *Output Dependence* (write after write): S_j schreibt auf eine Variable, die auch in S_i beschrieben wird.

a) Betrachtet das obige Programmstück. Bestimmt alle Datenabhängigkeiten in diesem Programmstück. Unterscheidet dabei zwischen *True*, *Anti* und *Output Dependencies* und gebt die betroffene Variable sowie die betroffenen Instruktionen an.

b) Für welche Zwecke kann die Information über Datenabhängigkeiten genutzt werden? Nennt die Zwecke, die Euch einfallen und begründet, warum die Information über die Datenabhängigkeiten hier genutzt werden können.

Aufgabe 3 (2 + 2 Punkte)

Betrachtet noch einmal das obige Assemblerprogramm. Nehmt an, dass die Befehle in einer fünf-stufigen Pipeline verarbeitet werden (Befehl holen, Befehl dekodieren/Operanden holen, Operation ausführen, Speicherzugriff und Operand speichern). Ein Schreibvorgang in das entsprechende Zielregister ist erst am Ende der *Operand-speichern* - Phase abgeschlossen. Bei Sprungbefehlen wird der Programmzähler am Ende der *Operation-ausführen* - Phase auf den neuen Wert gesetzt.

a) Wie viele und welche Pipeline-Konflikte können auftreten? Gebt jeweils die problematische Variable und die beteiligten Instruktionen an. Gebt die vollständige Pipeline an.

b) Behebt alle Pipeline-Konflikte durch Einfügen einer minimalen Anzahl von NOP-Befehlen. Gebt auch vollständige Pipeline an, die durch Einfügen der NOPs entsteht.

Aufgabe 4 $(1.5 + 1 + 2 + 2.5$ Punkte)

Im Folgenden sollt Ihr Euch mit der Beschleunigung, die das Pipelining bringt, beschäftigen und dabei noch weitergehende Techniken verwenden. Geht dabei davon aus, dass alle Pipeline-Stufen gleichlang dauern.

a) Berechnet für folgende drei Fälle die Dauer des oben genannten Programmes:

   a) vollständig ohne Pipelining-Struktur, d.h. ein Befehl wird erst gestartet, wenn der vorherige Befehl alle Stufen durchlaufen hat,

   b) mit maximalem Pipelining, aber ohne die Beachtung von Konflikten, d.h. keine Einfügung von NOPs oder ähnlichem, um diese zu umgehen, und

   c) mit der minimal nötigen Anzahl von NOPs, um alle Pipeline-Konflikte zu beheben.

b) Es wird eine neue Technik entwickelt, bei der der berechnete Wert schon nach der *Operation-ausführen*-Phase den nachfolgenden Instruktionen zur Verfügung gestellt wird. Wie ändert sich Euer Programm jetzt hinsichtlich der notwendigen Anzahl an NOPs? Gebt auch hier die vollständige entstandene Pipeline an und berechnet die Dauer.

c) Könnt Ihr in Eurer bisher schnellsten Version (also die mit der geringsten Dauer) noch Vorteile durch Umordnung von Befehlen herausholen? Es geht hier nur um die Umordnung von Befehlen, nicht um eine Zusammenfassung von Befehlen. Gebt, falls möglich, ein durch Umordnung von Befehlen noch schneller gewordenes Programm als vollständige Pipeline an und bestimmt auch hierzu die Dauer.

d) Bringt Eure bisher entwickelten Pipelines in eine Reihenfolge hinsichtlich Ihrer Dauer und berechnet die Beschleunigung im Vergleich zur Durchführung ohne Pipelining (vgl. Aufgabe 4a)-a)) , d.h. wenn ein Befehl stets erst nach der vollständigen Abarbeitung des vorherigen Befehls gestartet wird.

Abgabe: bis Donnerstag, den 11.05.2023, 08:15 Uhr per e-Mail an den Tutor und Christina Plump mit folgendem Betreff: [TI-Abgabe] <Tutorium> - <Gruppe>: Blatt <Blatt>