



7. Übungsblatt

Abgabetermin: 08. Juni 2007, 13:00 Uhr

Prof. Dr. J. Henkel

Am Zirkel 2, Geb. 20.20
D-76131 Karlsruhe

Dr.-Ing. T. Asfour

Telefon: +49-721-608-7379
Fax: +49-721-608-8270
Email: asfour@ira.uka.de
<http://ti.ira.uka.de>

Aufgabe 1

(10 Punkte)

1. Erläutern Sie die Aufgaben der einzelnen Pipeline-Stufen der DLX-Pipeline für

- arithmetisch-logische Befehle
- Lade-/Speicher-Befehle
- bedingte Sprungbefehle mit PC-relativer Adressierung

Das folgende Programmstück soll in der DLX-Pipeline abgearbeitet werden.

```
S1:          add  $t1, $zero, $zero
S2:          lw   $t3, 0x1500($zero)
S3:    loop:  lw   $t4, 0x5000($t1)
S4:          add  $t5, $t4, $t3
S5:          sw   $t5, 0x400($t1)
S6:          addi $t1, $t1, 4
S7:          subi $t2, $t1, 0x400
S8:          bnez $t2, loop
S9:    end:   srli $t1, $t1, 2
S10:         sw   $t1, 0x2000($zero)
```

2. Bestimmen Sie alle echten Datenabhängigkeiten und alle Steuerflussabhängigkeiten im Programmstück.
3. Gehen Sie davon aus, dass *Forwarding*-Techniken implementiert sind. Die einzige Methode zur Behebung von Pipelinekonflikten sei das Einfügen von NOP-Befehlen (*No Operation*) in den Befehlsstrom.

Ergänzen Sie das Programmstück durch das Einfügen von möglichst wenigen NOP-Befehlen, so dass alle Pipelinekonflikte behoben werden.

4. Was sind Struktur- oder Ressourcenkonflikte? Können diese bei der Ausführung des Programmstücks in der DLX-Pipeline auftreten? Begründen Sie Ihre Antwort.
5. Warum können Ausgabeabhängigkeiten (*output dependence*) und Gegenabhängigkeiten (*anti-dependence*) in der DLX-Pipeline nicht zu Konflikten führen?
6. Alle heutigen Prozessoren behandeln Steuerflusskonflikte durch die Hardware. Eine einfache Methode besteht in der Spekulation, dass bedingte Sprünge nicht genommen werden.
- Warum ist diese Methode ineffizient?

Aufgabe 2

(7 Punkte)

1. Gegeben sei folgendes Programm, das auf einem Prozessor mit DLX-Pipeline ohne Forwarding ausgeführt werden soll:

```
S1:  addi  $t1, $zero, 10
S2:  sll   $t2, $t1, 4
S3:  or    $t3, $t1, $t2
S4:  addi  $t4, $zero, 5
S5:  sll   $t5, $t4, 4
S6:  or    $t6, $t4, $t5
S7:  or    $t7, $t3, $t6
S8:  and   $t8, $t3, $t6
```

- i.) Bestimmen Sie alle echten Datenabhängigkeiten im Programmstück.
ii.) Ein TI-Student hat folgende NOP-Befehle in das Programm eingefügt, um die Pipelinekonflikte zu beheben.

```
S1:  addi  $t1, $zero, 10
      NOP
      NOP
S2:  sll   $t2, $t1, 4
      NOP
      NOP
S3:  or    $t3, $t1, $t2
S4:  addi  $t4, $zero, 5
      NOP
      NOP
S5:  sll   $t5, $t4, 4
      NOP
      NOP
      NOP
S6:  or    $t6, $t4, $t5
      NOP
      NOP
      NOP
S7:  or    $t7, $t3, $t6
S8:  and   $t8, $t3, $t6
```

Verringern Sie die Anzahl der NOP-Befehle durch Umordnen der Befehle, ohne das Ergebnis zu verändern.

- iii.) Was steht nach der Ausführung des Programms in den Registern `$t7` und `$t8`?

Aufgabe 3

(8 Punkte)

Das folgende Programmstück soll in der DLX-Pipeline abgearbeitet werden.

```
loop:  lw   $t1, 4($t2)
       addi $t1, $t1, 0x10
       sw   $t1, 4($t2)
       addi $t2, $t2, 0x4
       sub  $t4, $t3, $t2
       bnez $t4, loop
```

Das Register \$t3 ist mit dem Wert \$t2+1000 initialisiert.

1. Geben Sie den Zustand der Pipeline bei einer **korrekten** Abarbeitung des Programmstücks an. Nehmen Sie an, dass kein *Forwarding* möglich ist, aber dass das Lesen und Schreiben eines Registers im gleichen Taktzyklus erlaubt ist. Bei Steuerflusskonflikten wird das *Pipeline flushing* verwendet.

Wieviele Taktzyklen sind zur Ausführung des Programmstücks notwendig?

2. Geben Sie den Zustand der Pipeline bei einer **korrekten** Abarbeitung des Programmstücks an, wenn dieses in einer modifizierten DLX-Pipeline abgearbeitet werden soll, bei der *Forwarding* implementiert ist, und einen Verzögerungsschleife (*delay slot*) hinter einem Sprungbefehl hat. Weiterhin besteht die Möglichkeit, bei drohenden Konflikten die Befehle des Programmstücks umzuordnen.

Wieviele Taktzyklen sind zur Ausführung des Programmstücks notwendig?