HEINRICH-HEINE-UNIVERSITÄT DÜSSELDORF
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR RECHNERNETZE
PROFESSOR DR. M. MAUVE

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Grundlagen der Technischen Informatik

Lösungsvorschlag Übungsblatt 1

Aufgabe 1.1 *Rechnerleistung* (2 Punkte)
Auf Rechner *A* dauert das Bearbeiten einer Anweisung 10 ns, auf Rechner *B* 5 ns. Kann man sagen, dass Rechner *B* schneller ist? Formulieren Sie Ihre Argumentation schriftlich in etwa 5 Sätzen.

Lösungsvorschlag:

Nein, diese Aussage kann so nicht getroffen werden. Denn auch wenn Rechner *A* länger für die Bearbeitung einer Anweisung braucht, so kann es doch sein, dass Rechner *B* über eine kürzere oder schlimmstenfalls gar keine Pipeline verfügt. Die Verarbeitungszeit einer einzelnen Anweisung ist nur einer von mehreren Faktoren, letztlich hängt die Performance eines Rechners maßgeblich von der Fähigkeit ab, Anweisungen parallelisieren zu können.

Kommentar für die Tutoren:

Für die richtige Antwort und Begründung jeweils 1 Punkt.

Aufgabe 1.2 *Pipelining* $(2 \cdot 0,5 + 2 \cdot 0,5 + 3 \cdot 1 + 2 \cdot 0,5 = 6$ Punkte)
Ausgangspunkt des Pipelining ist die Aufspaltung einer Instruktion in Stufen. Das Pipelining setzt voraus, dass für die zugrundeliegende Befehlsarchitektur eine Folge von Teilschritten gefunden wird, die für *alle* Befehle gleich ist. Für die folgende Aufgabe wird angenommen, dass alle Befehle in den folgenden fünf Stufen bearbeitet werden:

1. **Instruction Fetch (IF):** Befehl holen
2. **Instruction Decode (ID):** Befehl dekodieren und gleichzeitig die Quelloperanden aus dem Registerblock lesen
3. **Execute (EX):** Führe eine arithmetische bzw. logische Operation mit den Operanden aus
4. **Memory Access (MEM):** Daten holen (*LOAD*) oder Speichern (*STORE*)
5. **Write Back (WB):** Ergebnis in einem prozessorinternen Register speichern

In der unten stehenden Tabelle sind die Rechenzeiten für die einzelnen Stufen der Pipeline angegeben.

IF	ID	EX	MEM	WB
110 ps	130 ps	90 ps	120 ps	70 ps

Nehmen Sie an, dass sowohl das Lesen als auch das Schreiben der Register zwischen den einzelnen Pipeline-Stufen (d.h. der Puffer IF/ID, ID/EX usw.) eine zusätzliche Verzögerung von jeweils 15 ps erzeugt.

a) Wie groß ist die minimal zulässige Taktperiode der Pipeline-Architektur? Wie groß ist die minimal zulässige Taktperiode einer entsprechenden Einzeltakt-Architektur ohne Pipeline?

Lösungsvorschlag:

- *Pipeline-Architektur:* 160 ps
  Die Taktperiode richtet sich nach der langsamsten Stufe, in diesem Fall ist es ID mit 130 ps. Hinzu kommen $2 \cdot 15$ ps für das Lesen und Schreiben der Register.
- *Einzeltakt-Architektur:* 110 ps + 130 ps + 90 ps + 120 ps + 70 ps = 520 ps

Kommentar für die Tutoren:

Auch wenn wir in der Aufgabenstellung und in der Musterlösung von 15ps Lesen vor der Stufe und 15ps Schreiben nach der Stufe ausgegangen sind, ist auch eine Interpretation der Aufgabenstellung mit nur einmal 15ps pro Pipelinestufe als richtig zu bewerten. Gleiches gilt für die Folgeaufgaben.

b) Wie groß ist die Ausführungszeit einer Instruktion in der Pipeline-Architektur? Wie groß ist die Ausführungszeit in der Einzeltakt-Architektur? In welcher Architektur dauert die Ausführung einer einzelnen Instruktion länger?

Lösungsvorschlag:

- *Pipeline-Architektur:* $5 \cdot 160 = 800$ ps
- *Einzeltakt-Architektur:* 110 ps + 130 ps + 90 ps + 120 ps + 70 ps = 520 ps

Die Ausführung einer einzelnen Instruktion dauert in der Pipeline-Architektur also länger.

c) Wie lange braucht die Pipeline-Architektur, um ein Programm mit 50 Instruktionen ohne Abhängigkeiten und Sprünge auszuführen? Wie groß ist der Speed-up im Vergleich zur Ausführung auf einer Einzeltakt-Architektur?

Lösungsvorschlag:

- Laufzeit *Pipeline-Architektur:* Zu Beginn werden 4 Taktzyklen gebraucht, um Pipeline zu füllen, danach nur noch ein Taktzyklus pro Instruktion. Also $4 \cdot 160$ ps + $1 \cdot 50 \cdot 160$ ps = 8 640 ps = 8,64 ns
- Laufzeit *Einzeltakt-Architektur:* 520 ps pro Instruktion, also $50 \cdot 520$ ps = 26 000 ps = 26,00 ns
- Speed-up: 26 ns/8,64 ns $\approx$ 3,0093

d) Nehmen Sie an, Sie können eine der fünf Stufen der beschriebenen Pipeline in zwei Unterstufen mit halber Rechenzeit aufteilen. Welche Stufe würden Sie wählen? Wie groß ist die neue minimale Taktperiode?

Lösungsvorschlag:

Die ID Stufe ist die beste Wahl, da sie die längste Rechenzeit hat. Die neue Taktperiode richtet sich dann nach der MEM Stufe und beträgt 120 ps + $2 \cdot 15$ ps = 150 ps.

Aufgabe 1.3 *Fehlererkennung und Fehlerkorrektur*($0,5+0,5+0,5+1,5+1+1+1+2=8$ Punkte)

1. Berechnen Sie die gerade zweidimensionale Parität:

```
1 1 1 0 | _
1 0 0 1 | _
0 1 1 1 | _
0 0 1 0 | _
--------+--
_ _ _ _ | _
```

Lösungsvorschlag:

```
1 1 1 0 | 1
1 0 0 1 | 0
0 1 1 1 | 1
0 0 1 0 | 1
--------+--
0 0 1 0 | 1
```

2. Sie erhalten ein mit zweidimensionaler Parität gesichertes Codewort, in dem genau ein Bit gekippt worden ist. Wurde gerade oder ungerade Parität verwendet? Identifizieren Sie den Bitfehler und beheben Sie ihn!

```
1 0 1 1 | 1
0 1 1 0 | 0
1 0 1 0 | 0
1 1 1 0 | 1
--------+--
1 0 0 1 | 1
```

Lösungsvorschlag:

g steht für gerade Parität
u steht für ungerade Parität

```
1 0 1 1 | 1 g
0 1 1 0 | 0 g
1 0 1 0 | 0 g
1 1 1 0 | 1 g
--------+--
1 0 0 1 | 1 u
g g g g   u
```

Da nur ein Bit gekippt ist, muss gerade Parität verwendet worden sein und das Bit unten rechts umgekippt sein.
Das korrigierte Codewort hat also die Form:

```
1 0 1 1 | 1 g
0 1 1 0 | 0 g
1 0 1 0 | 0 g
1 1 1 0 | 1 g
--------+--
1 0 0 1 |>0<g
g g g g   g
```

3. Berechnen Sie die Hamming-Distanzen der folgenden Paare von Codewörtern:
   (i) 1001 1111 1101 1000 und 1001 0101 1011 0110
   (ii) 0010 1101 0110 1011 und 0010 1111 1001 0011

Lösungsvorschlag:

```
(i) 1001 1111 1101 1000
    1001 0101 1011 0110
    ================== XOR
    0000 1010 0110 1110 => 7 Unterschiede
                        => Hamming-Distanz 7
```

(i)
```
0010 1101 0110 1011
0010 1111 1001 0011
==================== XOR
0000 0010 1111 1000 => 6 Unterschiede
                    => Hamming-Distant 6
```

4. Berechnen Sie die Hamming-Distanz des folgenden Codes mit vier Codewörtern:
   0000 1101 1000 1111, 1001 1001 1001 1011,
   1001 1111 1000 1111, 1111 0000 0000 0101

   Lösungsvorschlag:

   Zunächst müssen wir die Hamming-Distanzen zwischen allen Codewortkombinationen bestimmen.

```
a) 0000 1101 1000 1111
b) 1001 1001 1001 1011
c) 1001 1111 1000 1111
d) 1111 0000 0000 0101

a) 0000 1101 1000 1111
b) 1001 1001 1001 1011
----------------------
   1001 0100 0001 0100 => Hamming-Distanz 5

a) 0000 1101 1000 1111
c) 1001 1111 1000 1111
----------------------
   1001 0010 0000 0000 => Hamming-Distanz 3

a) 0000 1101 1000 1111
d) 1111 0000 0000 0101
----------------------
   1111 1101 1000 1010 => Hamming-Distanz 10

b) 1001 1001 1001 1011
c) 1001 1111 1000 1111
----------------------
   0000 0110 0001 0100 => Hamming-Distanz 4

b) 1001 1001 1001 1011
d) 1111 0000 0000 0101
----------------------
   0110 1001 1001 1110 => Hamming-Distanz 9

c) 1001 1111 1000 1111
d) 1111 0000 0000 0101
----------------------
   0110 1111 1000 1010 => Hamming-Distanz 9
```

Die Hamming-Distanz des gesamten Codes ist das Minimum aller Hamming-Distanzen zwischen den einzelnen Codewörtern. Für diesen Code ist die Hamming-Distanz also 3.

5. Berechnen Sie die nötigen Redundanzbits, um das folgende Datenwort mittels des passenden (n,k)-Hammingcodes gegen Bitfehler zu sichern. Geben Sie das vollständige Codewort mit Daten- und Redundanzbits an. `110 0101 1100`

   Lösungsvorschlag:

   11 Datenbits erfordern 4 Prüfbits um einen (n,k) Hamming-Code zu erzeugen. Damit ergibt sich folgendes Muster: $p_1 p_2 d_1 p_3 d_2 d_3 d_4 p_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11}$. Die Prüfbits sind gerade Paritätbits über die folgenden Bereiche:

   ```
          __1_100_1011100
   P1     X-X-X-X-X-X-X-X
   P2     -XX--XX--XX--XX
   P3     ---XXXX-------XXX
   P4     ----------XXXXXXX
   wird zu 1011 1000 1011 100
   ```

   Das vollständige Codewort inclusive Redundanzbits lautet 1011 1000 1011 100.

6. Nehmen Sie an, Sie haben das folgende (11,4)-Hamming-Code-gesicherte Codewort empfangen. Prüfen Sie das Codewort und korrigieren Sie es, wenn nötig (unter der Annahme, dass höchstens ein Bit gekippt ist). `101 0011 1101 0101`

   Lösungsvorschlag:

   Mit dem gleichen Schema wie eben überprüfen wir nun die Richtigkeit der

   ```
                1010 0111 1010 101
          P1    X-X-X-X-X-X-X-X Paritätsbit falsch
   Paritätsbits: P2  -XX--XX--XX--XX Paritätsbit falsch
          P3    ---XXXX-------XXX Paritätsbit richtig
          P4    ----------XXXXXXX Paritätsbit falsch
   ```

   Also ist ein Übertragungsfehler bei Bit $2^0 + 2^1 + 2^3 = 1 + 2 + 8 = 11$ aufgetreten. Das korrigierte Codewort lautet `101 0011 1100 0101`.

7. Im Codewort eines (n,k)-Hamming-Codes kippt das (n+k)-te Bit um. Welche Prüfbits weisen dadurch eine falsche Parität auf?

   Lösungsvorschlag:

   Die (n+k)-te, also die letzte Bitposition eines (n,k)-Hamming-Codeworts ist stets um eins kleiner als eine Zweierpotenz. Stellt man diese Bitposition binär dar, so sind hier alle k Bits auf 1 gesetzt. Somit fließt das letzte Bit des Codeworts in die Berechnung sämtlicher Prüfbits ein. Kippt das (n+k)-te Bit um, so sind alle k Prüfbits falsch.

8. Zeigen Sie, dass es keinen Code mit Hamming-Distanz $2c$ geben kann, der alle c-Bitfehler korrigieren kann!

   Lösungsvorschlag:

   Beweis durch Widerspruch: Angenommen, es existiert ein Code mit Hamming-Distanz $2c$, der alle c-Bitfehler beheben kann. Dann gibt es mindestens zwei Codewörter X und Y, welche die Hamming-Distanz $2c$ haben. Man kann also Codewort X ins Codewort Y durch Kippen von $2c$ Bits überführen. Wir kippen im Codewort X zunächst c dieser $2c$ Bits und erhalten das ungültige Codewort Z. Der so konstruierte c-Bitfehler kann vom Code nicht behoben werden, da es (mindestens) zwei mögliche gültige Codewörter gibt, nämlich X und Y, die in Folge eines c-Bitfehlers zu Z werden. Z kann also nicht eindeutig korrigiert werden.