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

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Grundlagen der Technischen Informatik

Übungsblatt 2

Abgabefrist: 01.05.2013 8:30 Uhr

Ansprechpartner: Der Tutor ihrer Übungsgruppe

Geben Sie zu jeder Aufgabe Ihren Lösungsweg an!

Aufgabe 2.1 *Umwandlung zwischen Zahlensystemen* (2 Punkte)
Wandeln Sie die gegebenen natürlichen Zahlen in die angegebenen anderen Darstellungsformen um.

1. 237_{10} in die Binär- und Hexadezimaldarstellung
2. 111010_2 in die Dezimal- und Hexadezimaldarstellung

Achten Sie hier besonders darauf, dass der Lösungsweg ersichtlich ist!

Korrekturhinweise:
Pro Umwandlung einen halben Punkt.

Lösungsvorschlag:
Wir konvertieren 237 mit der Restwertmethode:

```
237 /  2 = 118, Rest 1    237 / 16 =  14, Rest 13 (D)
118 /  2 =  59, Rest 0     14 / 16 =   0, Rest 14 (E)
 59 /  2 =  29, Rest 1
 29 /  2 =  14, Rest 1
 14 /  2 =   7, Rest 0
  7 /  2 =   3, Rest 1
  3 /  2 =   1, Rest 1
  1 /  2 =   0, Rest 1
```

$237_{10} = 11101101_2 = \mathrm{ED}_{16}$

Eine Binärzahl übertragen wir durch Summenbildung ins Dezimalsystem:

$$111010_2 = (0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5)_{10} \quad = (2 + 8 + 16 + 32)_{10} = 58_{10}$$

Wir könnten nun wieder die Restwertmethode für die Darstellung als Hexadezimalzahl verwenden. Alternativ kann aber die Binärzahl in 4-Bit-Blöcke unterteilt werden und die Hexadezimalstellen abgelesen werden:

$$11\ 1010_2 = \underbrace{0011}_{3}\underbrace{1010}_{A}{}_2 = 3A_{16}$$

Aufgabe 2.2 *Zweierkomplementdarstellung* (5 Punkte)

1. Gehen Sie zunächst von natürlichen Zahlen (keine Zweierkomplementdarstellung) aus. Wie viele Zahlen lassen sich mit 7 Bit und wie viele mit 8 Bit darstellen? Geben Sie auch jeweils den darstellbaren Zahlenbereich mittels Dezimalzahlen an.

   Korrekturhinweise:
   Ein halber Punkt für die korrekte Lösung.

   Lösungsvorschlag:
   7 Bit: $2^7 = 128$, $\{0, \ldots, 127\}$ 8 Bit: $2^8 = 256$, $\{0, \ldots, 255\}$

2. Gehen Sie nun von der Zweierkomplementdarstellung aus. Wie viele positive und wie viele negative Zahlen (0 gilt hierbei als positiv) lassen sich mit 8 Bit darstellen? Geben Sie den positiven und negativen darstellbaren Zahlenbereich mittels Dezimalzahlen an.

   Korrekturhinweise:
   Ein halber Punkt für die korrekte Lösung.

   Lösungsvorschlag:
   Je $2^7 = 128$ negative und positive Zahlen, $\{-128, \ldots, -1\}$, $\{0, \ldots, 127\}$

3. Geben Sie eine allgemeine Formel für den Zahlenbereich der Zweierkomplementdarstellung in Abhängigkeit von der Bitanzahl n an. Die Angabe des Zahlenbereichs soll mittels Dezimalzahlen erfolgen.

   Korrekturhinweise:
   Hier reicht die Angabe des Bereiches ohne Begründung. Ein Punkt für die korrekte Lösung.

   Lösungsvorschlag:
   Wenn das n-te Bit gesetzt ist, ist die Zahl negativ. Darstellbar sind also 2^{n-1} negative

und 2^{n-1} nicht-negative Zahlen. Die kleinste darstellbare Zahl ist daher -2^{n-1}. Die nicht-negativen Zahlen bestehen aus der 0 sowie $2^{n-1}-1$ positiven Zahlen. Die größte darstellbare Zahl ist also $2^{n-1}-1$.

4. Berechnen Sie die 8-Bit-Zweierkomplementdarstellung der folgenden dezimalen Zahlen: 87, -2, -17

   Korrekturhinweise:
   Wir unterscheiden hier zwischen fünf Arbeitsschritten: Dreimal Umwandeln der Zahlen und zweimal Bildung des Zweierkomplements. Wenn mindesten zwei Schritte korrekt sind, gibt es 0,5 Punkte. Bei vier oder mehr korrekten Rechnungen gibt es 1 Punkt.

   Lösungsvorschlag:
   Zunächst bestimmen wir jeweils den Betrag in Basis 2:

```
87 / 2 =  43, Rest 1      2 / 2 = 1, Rest 0
43 / 2 =  21, Rest 1      1 / 2 = 0, Rest 1
21 / 2 =  10, Rest 1
10 / 2 =   5, Rest 0     17 / 2 = 8, Rest 1
 5 / 2 =   2, Rest 1      8 / 2 = 4, Rest 0
 2 / 2 =   1, Rest 0      4 / 2 = 2, Rest 0
 1 / 2 =   0, Rest 1      2 / 2 = 1, Rest 0
                          1 / 2 = 0, Rest 1
```

   Also $87_{10} = 1010111_2$, $2_{10} = 10_2$ und $17_{10} = 10001_2$. Diese Beträge füllen wir mit Nullen auf, sodass 8-Bit-Zahlen entstehen:
   01010111, 00000010, 00010001.
   Für die beiden negativen Zahlen werden die Beträge jeweils invertiert und um 1 erhöht:

```
~00000010 + 1 = 11111101 + 1 = 11111110
~00010001 + 1 = 11101110 + 1 = 11101111
```

   Wir erhalten also $87_{10} = 01010111_2$, $-2_{10} = 11111110$, $-17_{10} = 11101111_2$.

5. Überführen Sie zuerst die Dezimalzahlen in die 8-Bit-Zweierkomplementdarstellung und führen Sie dann die Berechnungen durch:

   (a) $14 + (-22)$

   (b) $103 + 28$

   Wird bei den Berechnungen das Overflow-Flag (OF) gesetzt? Wenn ja, bei welchen und woran erkennt man, dass es gesetzt wird? Welche Bedeutung hat das OF hier?

   Korrekturhinweise:
   Je einen halben Punkt für: die beiden Rechnungen, die Aussage, dass das OF bei b) gesetzt wird, sowie die Erklärung zum OF.

   Lösungsvorschlag:

```
(a) 00001110 + (~00010110 + 1) = 00001110
                                + 11101010
                                ----------
                                  11111000

(b)   01100111
    + 00011100
      --------
      10000011
```

Das Overflow-Flag wird gesetzt. Man erkennt es daran, dass ein Übertrag zur höchsten Stelle, aber nicht von der höchsten Stelle vorliegt. Das gesetzte Overflow-Flag bedeutet, dass der gültige Wertebereich verlassen wird. Man sieht es am falschen Ergebnis mit -125_{10} anstatt 131_{10}.

Aufgabe 2.3 *Caching* (9 Punkte)
Auf die angegebene Folge von Adressen wird in dieser Reihenfolge lesend zugegriffen:

16, 72, 20, 9, 183, 60, 19, 12, 148, 22, 8, 125, 183, 34, 47, 60

Der Speicher wird byteweise mit Adressen 0,1,2,3,... addressiert. Das Programm möchte jeweils ein Byte lesen. Die Größe des Adressraumes beträgt 16 Bit (Adressen sind also insgesamt 16 Bit lang).

Geben Sie für jede Teilaufgabe die allgemeine Zusammensetzung der Adresse analog zu Kapitel 2, Folie 34/35 an (also TAG, LINE, WORD, BYTE). Die Wortlänge sei 4 Byte.

Der Cache ist zu Beginn jeder Teilaufgabe leer, die Strategie zum Finden des zu ersetzenden Speicherplatzes im Cache ist LRU (*Least Recently Used*).

Geben Sie für jede Teilaufgabe die Entwicklung der Cache-Belegung in Tabellenform an. In Ihrer Tabelle soll jede Zeile einer Cache-Zeile entsprechen, die Sie mit den entsprechenden Spalten (Line, Valid, Tag, gecachter Adressbereich, Zugriffsreihenfolge) darstellen. Die zeitlich aufeinanderfolgenden Zugriffe auf dieselbe Speicherstelle im Cache werden durch Untereinanderschreiben der (somit teilweise veralteten) Cache-Inhalte innerhalb einer Zelle verdeutlicht. (Das Erstellen der Tabelle ist als reine Textdatei mit entsprechenden Trennzeichen am besten möglich.)

Ein Beispiel für einen Cache mit *n* Zeilen:

Line	**Valid**	**Tag**	**gecacheter Adressbereich**	**Zugriffsreihenfolge (für LRU)**
0.	1	2	16-23	1,3,7,10
1.	1	9	72-79	2
⋮				
n.	1	4	32-39	14

Dokumentieren Sie für jede Teilaufgabe außerdem in Form einer Liste, welcher Zugriff einen *Hit* oder *Miss* verursacht (z. B.: 16m, 72m, 20h,...).

Korrekturhinweise:
Die Unterteilung der Speicheradresse in WORD/BYTE war vielen Studenten unklar und wird für die Anwendung der Caching-Algorithmen in dieser Aufgabe nicht gebraucht. Daher sollte nur darauf geachtet werden, dass TAG/LINE richtig berechnet wurden. Für falsche/fehlende WORD/BYTE-Bereiche werden keine Punkte abgezogen.

Wir vergeben getrennt Punkte für die richtige Zusammensetzung der Speicheradresse und die korrekte Cache-Tabelle/Hit-Miss-Liste:

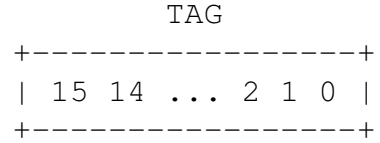
Teilaufgabe	Adresse	Tabelle/HM-Liste	Summe
1	0,5	0,5	1
2	0,5	1,5	2
3	1	2	3
4	1	2	3

Für die Zusammensetzung der Speicheradresse gibt es die angegebene Punktzahl, wenn TAG und LINE richtig sind.

Pro Fehler in der Cache-Tabelle oder Hit-Miss-Liste werden 0,25 Punkte vom in Spalte „Tabelle/HM-Liste“ gegebenen Wert abgezogen. Hierbei sollten Folgefehler nur einmal gezählt werden. Es gibt natürlich keine Minuspunkte.

1. *Fully Associative Cache, Speicherkapazität 16 Byte, Zeilenlänge 1 Byte:* In diesem Cache stehen 16 Zeilen, die je ein Byte aufnehmen können, zur Verfügung. Die Zeilen des leeren Caches werden der Reihe nach gefüllt, die Nummer der Zeile spielt für die Adressberechnung keine Rolle.

   Lösungsvorschlag:
   Zusammensetzung der Adresse:

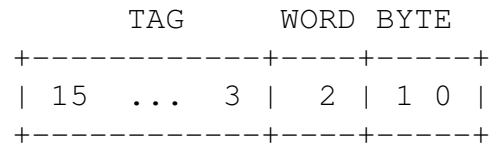


   16m, 72m, 20m, 9m, 183m, 60m, 19m, 12m, 148m, 22m, 8m, 125m, 183h, 34m, 47m, 60h

Line	Valid	Tag	gecacheter Adressbereich	Zugriffsreihen-folge (für LRU)
0.	1	16	16	1
1.	1	72	72	2
2.	1	20	20	3
3.	1	9	9	4
4.	1	183	183	5,13
5.	1	60	60	6,16
6.	1	19	19	7
7.	1	12	12	8
8.	1	148	148	9
9.	1	22	22	10
10.	1	8	8	11
11.	1	125	125	12
12.	1	34	34	14
13.	1	47	47	15
14.	0			
15.	0			

2. *Fully Associative Cache, Speicherkapazität 64 Byte, Zeilenlänge 8 Byte:*
   In diesem Cache stehen acht Zeilen, die jeweils acht Byte aufnehmen können, zur Verfügung. Die Zeilen des leeren Caches werden der Reihe nach gefüllt, die Nummer der Zeile spielt für die Adressberechnung keine Rolle.

   Lösungsvorschlag:

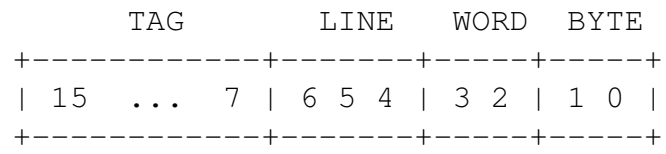


   16m, 72m, 20h, 9m, 183m, 60m, 19h, 12h, 148m, 22h, 8h, 125m, 183h, 34m, 47m, 60h

Line	Valid	Tag	gecacheter Adressbereich	Zugriffsreihen-folge (für LRU)
0.	1	2	16–23	1,3,7,10
1.	1	9	72–79	2
1.	1	5	40–47	15
2.	1	1	8–15	4,8,11
3.	1	22	176–183	5,13
4.	1	7	56–63	6,16
5.	1	18	144–151	9
6.	1	15	120–127	12
7.	1	4	32–39	14

3. *Direct Mapped Cache, Speicherkapazität 128 Byte, Zeilenlänge 16 Byte:*
   Nehmen Sie an, es wird eine direkte Abbildung in den Cache mit einer Zeilenlänge von 16 Byte vorgenommen.

   Lösungsvorschlag:



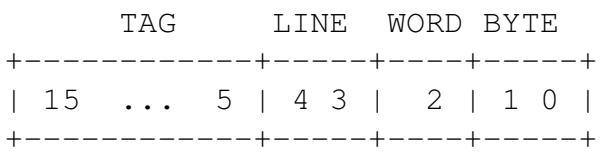
   16m, 72m, 20h, 9m, 183m, 60m, 19h, 12h, 148m, 22m, 8h, 125m, 183m, 34m, 47h, 60m

Line	Valid	Tag	gecacheter Adressbereich	Zugriffsreihen-folge (für LRU)
0.	1	0	0–15	4,8,11
1.	1	0	16–31	1,3,7
	1	1	144–159	9
	1	0	16–31	10
2.	1	0	32–47	14,15
3.	1	1	176–191	5
	1	0	48–63	6
	1	1	176–191	13
	1	0	48–63	16
4.	1	0	64–79	2
5.	0			
6.	0			
7.	1	0	112–127	12

4. *2-Way Set-Associative Cache, Speicherkapazität 64 Byte, Zeilenlänge 8 Byte:*

   Nehmen Sie an, es wird ein 2-Way Set-Associative Cache mit einer Zeilenlänge von acht Byte verwendet.

   Lösungsvorschlag:



   Da es sich um einen 2-Way Set-Associative Cache mit insgesamt 64 Byte Kapazität handelt, bei dem in jedem Eintrag 8 Byte gecachet werden, gibt es hier *vier* Zeilen mit jeweils zwei Einträgen zu je 8 Byte.

   16m, 72m, 20h, 9m, 183m, 60m, 19h, 12h, 148m, 22h, 8h, 125m, 183m, 34m, 47m, 60h

Line	V.	T.	Adressen	Zugriff	V.	T.	Adressen	Zugriff
0.	1	1	32–39	14	0			
1.	1	2	72–79	2	1	0	8–15	4,8,11
	1	1	40–47	15				
2.	1	0	16–23	1,3,7,10	1	5	176–183	5
					1	4	144–151	9
					1	5	176–183	13
3.	1	1	56–63	6,16	1	3	120–127	12