Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

7. Aufgabenblatt mit Lösungsvorschlag

13.12.2010

Hinweis: Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 2 (10.01. bis 14.01.) bei den Tutoren in **physikalischer Form** (handschriftlich oder gedruckt) abzugeben. Bei allen Abgaben ist der Name des Tutors und die Übungsgruppe deutlich anzugeben. Bei Teamabgaben wird nur eine Lösung eingereicht, die alle Namen der Teammitglieder enthält.

Aufgabe 1: Schnelltest

Fragen	*Antworten*
1. *Gegeben sei ein vollassoziativer 32 Byte-Cache mit 4 Wort Blöcken und 32 Bit Wortgröße. Auf die Daten im Hauptspeicher wird in folgender Sequenz zugegriffen:* `0x001, 0x004, 0x008, 0x004, 0x01E, 0x004, 0x0FF`*. Die Ersetzungsstrategie soll LRU sein. Welcher Block wird beim letzten Zugriff der Sequenz ersetzt?*	☐ *Keiner* ☐ *0x000 - 0x00F* ■ *0x010 - 0x01F* ☐ *0x0F0 - 0x0FF* ☐ *0x100 - 0x10F*
2. *Ein vierfach satzassoziativer Cache habe 32 Blöcke. Die Wortgröße betrage 32 Bit. Wie groß muss dann der Cache sein (ohne Valid-Bit oder Tag-Bits)?*	☐ *256 Bytes* ☐ *512 Bytes* ☐ *512 KBytes* ☐ *32 Bytes* ■ *128 Bytes*
3. *Welche Aussagen über Caches sind korrekt?*	☐ *Ein vierfach satzassoziativer Cache ist doppelt so groß wie ein zweifach satzassoziativer Cache.* ☐ *Je höher die Assoziativität, desto höher ist in der Regel die Miss-Rate.* ■ *Je höher die Assoziativität, desto aufwändiger ist das Suchen.* ☐ *Je höher die Assoziativität, desto mehr Index-Bits sind nötig.*
4. *Welche Aussagen zur Daisy Chain sind korrekt?*	■ *Andere Geräte können „aushungern“.* ☐ *Andere Geräte können nie „aushungern“.* ☐ *Subsystem mit höchster Priorität befindet sich am Ende der grant-Kette.* ☐ *Schnelligkeit, da grant-Signal direkt an E/A-Gerät gesendet wird.*

Aufgabe 2: Direct-Mapped Cache

Betrachtet wird ein Direct-Mapped Cache. Folgende Eigenschaften sind gegeben:

- *16-Bit Adressgröße*
- *Blockgröße 256 Bytes, ansprechbar zu 8 Bit*
- *16 Cache-Blöcke (Sets)*

a) *Ermitteln Sie die Bitbreite von Tag, Index und Blockoffset.*

Lösungsvorschlag:

Tag 4 Bit, Blockoffset 8 Bit, Index 4 Bit

b) *Ermitteln Sie für die folgenden Zugriffe, ob es sich um Hits oder Misses handelt. Tragen Sie in die untere Tabelle jeweils ein, wie sich der Cacheinhalt verändert. Der erste Eintrag ist vorgegeben.*

Lösungsvorschlag:

Zugriff	0x0D3A	0x017D	0x0A9C	0x2752	0x510B	0x78B3	0x0D8D
Hit/Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit

0x0AD0	0x32C7	0x1888	0x4F0E	0x6D6A	0x14B3	0x014B	0x0AA3
Hit	Miss	Miss	Miss	Miss	Miss	Miss	Hit

Index	Valid	Tag	Daten
0			
1	x	~~0~~ ~~5~~ 0	~~Mem[0x0100 - 0x01FF]~~ ~~Mem[0x5100 - 0x51FF]~~ Mem[0x0100 - 0x01FF]
2	x	3	Mem[0x3200 - 0x32FF]
3			
4	x	1	Mem[0x1400 - 0x14FF]
5			
6			
7	x	2	Mem[0x2700 - 0x27FF]
8	x	~~7~~ 1	~~Mem[0x7800 - 0x78FF]~~ Mem[0x1800 - 0x18FF]
9			
A	x	0	Mem[0x0A00 - 0x0AFF]
B			
C			
D	x	~~0~~ 6	~~Mem[0x0D00 - 0x0DFF]~~ Mem[0x6D00 - 0x6DFF]
E			
F	x	4	Mem[0x4F00 - 0x4FFF]

c) *Im Folgenden wird ein zweifach satzassoziativer Cache mit denselben Werten wie oben (und derselben Gesamtzahl von Cacheblöcken) betrachtet. Welche Bitbreiten ändern sich hierdurch? Ändert sich die Gesamtgröße des Caches (einschließlich Verwaltungsinformationen)?*

Lösungsvorschlag:

Der Cache hat nur noch 8 Zeilen, wodurch der Index nur noch aus 3 Bit besteht und der Tag ein Bit größer wird.

Der Cache wird insgesamt größer, denn die Tag-Felder werden größer.

d) *Inwieweit ist eine solche Variante des Caches „besser“ als die aus Aufgabenteil a)? Würden sich weitere Verbesserungen ergeben, wenn man einen 16-fach satzassoziativen Cache verwenden würde?*

Lösungsvorschlag:

Wenn ein Cache eine niedrige Assoziativität hat, kann es sein, dass einige Cache-Einträge ungenutzt bleiben (wenn nämlich keine der verwendeten Adressen die entsprechenden Index-Bits aufweist). Die Wahrscheinlichkeit für so einen Fall ist geringer, wenn es weniger Cache-Zeilen gibt. Somit verbessert sich die Ausnutzung des Caches mit höherer Assoziativität.

Andererseits steigt hierdurch auch der Aufwand, eine Seite im Cache zu suchen, da mehr Tag-Felder verglichen werden müssen. Bei schnellen Hardware-Caches wird hierbei ein paralleler Vergleich der Tags durchgeführt. Daher vergrößert sich der Hardware-Aufwand mit erhöhter Assoziativität, wodurch der Cache letztlich teurer wird.

Aufgabe 3: Direct-Mapped Cache

Betrachten Sie den folgenden Pseudo-Code:

```
int[][] dst = new int[2][2];
int[][] src = new int[2][2];

... // Hier wird src mit Werten gefuellt

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        dst[j][i] = src[i][j];
    }
}
```

Gehen Sie davon aus, dass die Elemente von `dst` *und* `src` *der Reihe nach hintereinander im Speicher abgelegt sind und jeder* `int`*-Wert 4 Bytes benötigt, wobei* `src` *an Speicheradresse* 0 *startet und* `dst` *direkt dahinter an Speicheradresse* 16 (= 10000_2).

Die Reihenfolge eines Arrays `a[2][2]` *im Speicher sei:* `a[0][0], a[0][1], a[1][0], a[1][1]`

Der Zugriff auf die beiden Arrays erfolgt über einen direkt abbildenden, byte-adressierten Cache mit Write-Back-Strategie. Die Blockgröße beträgt 8 Byte, die Gesamtgröße 16 Byte.

a) *Skizzieren Sie die Aufteilung einer 32-Bit-Adresse in Tag, Index und Blockoffset.*

Lösungsvorschlag:

Wegen der Blockgröße von 8 Byte werden 3 Bits für das Blockoffset benötigt. Der Cache hat 2 Einträge, daher ist Index 1 Bit breit. Die restlichen $32 - 3 - 1 = 28$ Bit werden für das Tag-Feld verwendet.

b) *Tragen Sie in den folgenden Tabellen ein, welche Arrayzugriffe bei Ausführung des o.a. Codestücks Hits (h) und welche Misses (m) sind.*

Lösungsvorschlag:

In der Schleife wird auf jedes Element genau einmal zugegriffen. So ist der erste Zugriff auf `src[0][0]` mit der Speicheradresse 0...0 xx (insgesamt 4 Bytes), die folgendermassen aufgeteilt ist:

Tag	Index	BlockOfs
0...0 0	0	0xx

Der erste Zugriff ist auf jeden Fall ein Miss, da der Cache leer ist. Die erste Zeile von `src` wird in den Cache-Eintrag mit Index 0 geladen.

Der zweite Zugriff ist auf `dst[0][0]` mit der Adresse 0...0 100xx:

Tag	Index	BlockOfs
0...0 1	0	0xx

Wieder wird der Cache-Eintrag 0 genutzt, aber da der Tag-Wert nicht übereinstimmt, ist es auch hier ein Miss.

Dritter Zugriff: `src[0][1]` → 0...0 001xx:

Tag	Index	BlockOfs
0...0 0	0	1xx

Das ist wieder ein Miss usw.

Insgesamt ergibt sich folgende Tabelle:

(a) `src`-Array

	Col 0	Col 1
Row 0	m	m
Row 1	m	h

(b) `dst`-Array

	Col 0	Col 1
Row 0	m	m
Row 1	m	m

Tabelle 3.2: 16-Bit-Cache

c) *Betrachten Sie jetzt dieselbe Aufgabenstellung, jedoch mit einem 32-Byte-Cache. Wie ist die Aufteilung von Speicheradressen, wie sehen die Tabellen aus?*

Lösungsvorschlag:

(a) `src`-Array

	Col 0	Col 1
Row 0	m	h
Row 1	m	h

(b) `dst`-Array

	Col 0	Col 1
Row 0	m	h
Row 1	m	h

Tabelle 3.3: 32-Bit-Cache

Hier ist zu beachten, dass der Cache anfangs wieder leer ist. Die Aufteilung der Adressen ist dieses mal etwas günstiger, und der Cache gross genug, dass jedes Element nur einmal in den Cache geladen wird.

Aufgabe 4: Vollassoziativer Cache

a) *Gegeben sei ein Cache mit folgenden Eigenschaften:*

- *Vollassoziativ*
- *Blockgröße 64 Kilobyte*
- *Gesamtgröße (ohne Verwaltungsinformationen) 256 Kilobyte*
- *FIFO*[1] *Ersetzungsstrategie*

Weiterhin sei eine Zugriffsfolge in nachfolgender Tabelle gegeben, bei der byteweise lesend auf die Daten im Hauptspeicher zugegriffen wird.

Notieren Sie zu jedem Zugriff, ob es sich um einen Hit (h) oder einen Miss (m) handelt sowie den Zustand der Tag-Felder des Datencaches. Tragen Sie die Werte für die Tag-Felder in Hexadezimalschreibweise ein. Leere Felder versehen Sie bitte mit einem Strich. Gehen Sie davon aus, dass der Datencache zu Beginn leer ist.

Benutzen Sie die folgende Tabelle:

[1] First In First Out

Lösungsvorschlag:

Zugriff (Byteadresse)	Hit/Miss	Tag-Felder des Caches			
		Eintrag 1	Eintrag 2	Eintrag 3	Eintrag 4
0x9A44 D124	m	*0x9A44*	-	-	-
0xA4B3 7486	m	0x9A44	*0xA4B3*	-	-
0x21CF EA04	m	0x9A44	0xA4B3	*0x21CF*	-
0xA4B3 7976	h	0x9A44	0xA4B3	0x21CF	-
0xFE67 D7E3	m	0x9A44	0xA4B3	0x21CF	*0xFE67*
0x9A44 8B23	h	0x9A44	0xA4B3	0x21CF	0xFE67
0x315A 16AD	m	*0x315A*	0xA4B3	0x21CF	0xFE67
0x9A44 D0A0	m	0x315A	*0x9A44*	0x21CF	0xFE67
0xA4B3 2497	m	0x315A	0x9A44	*0xA4B3*	0xFE67
0xFE67 F10C	h	0x315A	0x9A44	0xA4B3	0xFE67
0x21CF 1234	m	0x315A	0x9A44	0xA4B3	*0x21CF*

b) *Gegeben sei derselbe Cache wie in der vorherigen Teilaufgabe, jedoch soll in dieser Aufgabe eine LRU-Ersetzungsstrategie verwendet werden. Das heisst, es wird bei einem Cache-Miss derjenige Eintrag ersetzt, auf den am längsten nicht mehr zugegriffen wurde.*

Tragen Sie in die nachfolgende Tabelle wieder für jeden Zugriff ein, ob er ein Hit (h) oder ein Miss (m) ist und notieren Sie die Cachebelegung, indem Sie die Tagfelder eintragen:

Lösungsvorschlag:

Zugriff (Byteadresse)	Hit/Miss	Tag-Felder des Caches			
		Eintrag 1	Eintrag 2	Eintrag 3	Eintrag 4
0x9A44 D124	m	*0x9A44*	-	-	-
0xA4B3 7486	m	0x9A44	*0xA4B3*	-	-
0x21CF EA04	m	0x9A44	0xA4B3	*0x21CF*	-
0xA4B3 7976	h	0x9A44	0xA4B3	0x21CF	-
0xFE67 D7E3	m	0x9A44	0xA4B3	0x21CF	*0xFE67*
0x9A44 8B23	h	0x9A44	0xA4B3	0x21CF	0xFE67
0x315A 16AD	m	0x9A44	0xA4B3	*0x315A*	0xFE67
0x9A44 D0A0	h	0x9A44	0xA4B3	0x315A	0xFE67
0xA4B3 2497	h	0x9A44	0xA4B3	0x315A	0xFE67
0xFE67 F10C	h	0x9A44	0xA4B3	0x315A	0xFE67
0x21CF 1234	m	0x9A44	0xA4B3	*0x21CF*	0xFE67

Hausaufgabe 1: Vollassoziativer Cache (6 Punkte)

Gegeben sei ein Datencache mit folgenden Eigenschaften:

- *Vollassoziativ*
- *Blockgröße 4 KByte, byte-adressierbar*
- *4 Einträge*
- *LRU Ersetzungsstrategie*

a) *Wie viele Nutzdaten (d. h. keine Tag- und Verwaltungsbits) kann der Cache aufnehmen? Wie viele Bits besitzt ein Tag-Feld des Caches?*

Lösungsvorschlag:

Mit 4 KByte großen Blöcken und 4 Einträgen kann der Cache $4 \cdot 4 = 16$ KByte Daten aufnehmen. Die Blockgröße von 4 KByte $= 2^{12}$ Byte führt auf 12 Bits Blockoffset. Die übrigen Bits werden für das Tag benötigt. Bei einem 32 Bit System wären dies $32 - 12 = 20$ Bit für das Tag.

b) *Ergänzen Sie die folgende Tabelle. Gehen Sie davon aus, dass der Datencache zu Beginn leer ist.*

Lösungsvorschlag:

Zugriff		Tag-Felder des Caches			
—- (Byteadresse)	Hit/Miss	Eintrag 1	Eintrag 2	Eintrag 3	Eintrag 4
0x1234 0001	Miss	0x12340			
0x1A13 0002	Miss	0x12340	0x1A130		
0x1A13 1006	Miss	0x12340	0x1A130	0x1A131	
0x1A13 1876	Hit	0x12340	0x1A130	0x1A131	
0x17D8 0A02	Miss	0x12340	0x1A130	0x1A131	0x17D80
0x1234 0001	Hit	0x12340	0x1A130	0x1A131	0x17D80
0x1A13 0002	Hit	0x12340	0x1A130	0x1A131	0x17D80
0x2478 2174	Miss	0x12340	0x1A130	0x24782	0x17D80
0x1998 4446	Miss	0x12340	0x1A130	0x24782	0x19984
0xFA12 060E	Miss	0xFA120	0x1A130	0x24782	0x19984
0xD728 8802	Miss	0xFA120	0xD7288	0x24782	0x19984
0x213F 2A0B	Miss	0xFA120	0xD7288	0x213F2	0x19984

Hausaufgabe 2: Bus-Arbitration (4 Punkte)

Mehrere Subsysteme teilen sich einen gemeinsamen Datenbus. Es soll anhand eines einfachen Beispiels die Bus-Arbitration, d. h. die Zuteilung des Busses an die Subsysteme, untersucht werden. Die Bus-Arbitration soll gemäß festen Prioritäten (beispielsweise in Form einer „Daisy-Chain“) erfolgen. Subsystem 1 hat dabei die höchste, Subsystem 3 die niedrigste Priorität. Gehen Sie dabei zunächst von drei Subsystemen mit folgenden Buszugriffsanforderungen aus:

- *Subsystem 1 fordert, beginnend in Takt 0, alle vier Takte Zugriff auf den Bus an. Wenn die Zuteilung erfolgt wird der Bus jeweils einen Takt lang belegt.*
- *Subsystem 2 fordert, beginnend in Takt 1, alle sechs Takte Zugriff auf den Bus an. Wenn die Zuteilung erfolgt wird der Bus jeweils drei Takte lang belegt.*
- *Subsystem 3 fordert, beginnend in Takt 2, alle acht Takte Zugriff auf den Bus an. Wenn die Zuteilung erfolgt wird der Bus jeweils zwei Takte lang belegt.*

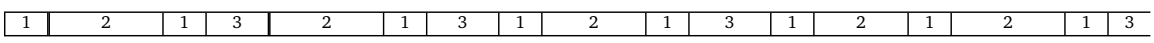
a) *Wie würde in diesem Fall die Bus-Zuteilung aussehen? Geben Sie in einem Diagramm die Anforderungszeitpunkte so wie die jeweilige Buszuteilung für die ersten 30 Takte an. Den Zeitaufwand für die Bus-Arbitration können Sie dabei vernachlässigen.*

Lösungsvorschlag:

Busanforderungen:

1 2 3 1 2 1 3 1 2 1 3 2 1 1 2 3 1

Buszuteilung:



b) *Angenommen es wird ein Subsystem 4 hinzugefügt welches eine niedrigere Priorität als Subsystem 3 haben soll. Welche Probleme treten auf wenn zusätzlich zur obigen Anforderungsreihenfolge noch Anforderungen eines vierten Subsystems hinzukommen würden?*

Lösungsvorschlag:

Der Bus ist bereits vollständig ausgelastet. Daher würden weitere Anforderungen zwangsläufig dazu führen, dass nicht mehr alle Anforderungen erfüllt werden können. Aufgrund der festen Prioritäten würde das neue Subsystem 4 nie eine Buszuteilung erhalten; seine Anforderungen würden „verhungern“.

c) *Welches der auftretenden Probleme kann durch eine andersartige Bus-Arbitration verhindert werden? Was müsste hierzu an der Bus-Arbitration geändert werden?*

Lösungsvorschlag:

Das Problem der Überlastung des Busses kann nicht durch die Bus-Arbitration gelöst werden. Lediglich durch einen leistungsfähigeren Datenbus könnte sichergestellt werden, dass jede Anforderung erfüllt wird. Allerdings kann verhindert werden, dass ein Subsystem niemals die Zuteilung erhält, indem eine faire Bus-Arbitration verwendet wird, bei der keines der Subsysteme „verhungern“ kann.