Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Übung Rechnerarchitekturen AIN 2 SS2023

5. Cache

Die Abgabe erfolgt durch Hochladen der Lösung in Moodle. Zusätzlich wird die Lösung in der Übung nach dem Abgabetermin stichprobenartig kontrolliert.

Bearbeitung in Zweier-Teams

Team-Mitglied 1: Philippe Westenfelder

Team-Mitglied 2:

Hochschule Konstanz
Fakultät Informatik

Prof. Dr.-Ing. Michael Blaich
Robotik und Künstliche Intelligenz

Aufgabe 5.1 Direct Mapped Cache

Ein Prozessor verwendet eine Speicherhierarchie mit zwei Cache-Ebenen. Die erste Cache-Ebene besteht aus einem Cache mit 32 Blöcken zu je 16 Bytes. Die zweite Cache-Ebene aus 1024 Blöcken mit ebenfalls 16 Bytes. Bei beiden Caches handelt es sich um Direct-Mapped-Caches. Ein Auszug der beiden Caches ist in Tabelle 1 und Tabelle 2 dargestellt. Der Prozessor lädt ein Wort an der Speicheradresse 2404. Welcher Wert wird zurückgegeben?

Index	V	Tag	Speicherblock (Words)			
…						
19	N	14	11	4	7	13
20	Y	300	23	32	98	76
21	Y	22	98	23	67	98
22	Y	1	7	6	5	4
23	N	7	8	9	10	11
…						

Tabelle 1: Cache der ersten Ebene

Index	V	Tag	Speicherblock			
offset	littleendian		4	3	2	1
148	Y	80	123	132	198	176
149	Y	6	98	23	67	98
150	Y	0	70	60	50	40
151	N	2	8	9	10	11
152	N	14	0	0	0	0
…						

Tabelle 2: Cache der zweiten Ebene

Blocknr.: 2404 : 16 = 150,25 ≙ 150

Blockindex: 150 mod 32 = 22

Tag: 150 : 32 = 4,6875 ≙ 4 ≠ 1

offset: 2404 mod 16 = 4

T2:

Blockindex: 150 mod 1024 = 150

Tag: 150 : 1024 ≙ 0

70 wird zurückgegeben

Aufgabe 5.2 Direct-Mapped-Cache und Set-Associative Cache

Ein Prozessor verwendet getrennte Daten- und Instruktionscaches. Der Daten-Cache wird über eine Speicherhierarchie mit zwei Cache-Ebenen realisiert. Der First-Level-Cache besteht aus einem Direct-Mapped-Cache mit 64 Blöcken zu je 8 Bytes. Ein Auszug des Caches ist in Tabelle 3 gegeben.
Der Second-Level-Cache ist ein 2-Way-Set-Associative-Cache mit einem Gesamtspeicherplatz von 2048 Bytes und einer Blockgröße von 8 Bytes. Ein Auszug dieses Caches ist in Tabelle 2 und Tabelle 4 gegeben. Gehen Sie weiterhin davon aus, dass eine LRU-Ersetzungsstrategie verwendet wird und die „linken" Blöcke in der Tabelle jeweils zuletzt genutzt wurden.

Beide Caches verwenden eine Write-Back-Strategie, wobei beim Ersetzen der „Dirty"-Block nur auf die nächst tiefere Cache-Ebene geschrieben wird.

In Abbildung 1 ist der Assembler-Code einer Prozedur gegeben, die elementweise die Summe zweier Arrays A und B bestimmt und in einem dritten Array C abspeichert. Die Übergabeparameter der Funktion sind:

$a0: Adresse des Arrays A
$a1: Adresse des Arrays B
$a2: Adresse des Arrays C
$a3: Anzahl Elemente

Hinweis: Die Cache-Inhalte in den Tabellen sind in nicht vorzeichenbehafteten (unsigned) Worten (Word) dargestellt.

5.2.1 Bestimmen Sie für den ersten Schleifendurchlauf, wie sich die Werte im Cache verändern, wenn die Prozedur mit den Werten $a0=1000, $a1=3040, $a2=9196 und $a3=3 aufgerufen wird. Tragen Sie die Veränderungen der Cache-Inhalte in Tabelle 5 bzw. Tabelle 6 ein. Geben Sie auch die Codezeile an, die die Veränderung verursacht. Geben Sie auch die Codezeile an, die die Veränderung verursacht. Tragen Sie in der mit „D" überschriebenen Spalte ein, ob es sich um einen „Dirty" Block handelt.

5.2.2 Betrachten Sie nun die gesamte Schleife mit den gegebenen Argumenten.
a. Wie viele Misses treten jeweils in der ersten und zweiten Cache-Ebene auf?
b. Bestimmen Sie den CPI der Prozedur, wenn der Zugriff auf den First-Level-Cache einen Takt, der Zugriff auf den Second-Level-Cache 10 Takte und der Zugriff auf den Hauptspeicher 100 Takte dauert?

a0

L1:

BNr: 1000 : 8 = 125

Bi : 125 % 64 = 61

Tag: 125 : 64 = 1,9 ≙ 1

Offset: 1000 % 8 = 0

L2: Sets: 256 : 2 = 128

Bi: 125 %

Tabelle 3: Auszug des Inhalt des First-Level-Caches

Index	V	Tag	Speicherblock (in Worten, dezimal, Byte 0 rechts)	
0	Y	17	1	0
1	Y	17	3	2
2	Y	17	5	4
…				
60	Y	5	7	6 a1
61	Y	17	9	8
62	Y	17	11	10
63	Y	17	13	12
…				

Tabelle 4: Auszug des Inhalts des Second-Level-Caches

Set Index	V	Tag	Speicherblock 1 (in Worten, dezimal, Byte 0 rechts)		V	Tag	Offset 4? Speicherblock 2 Offset 0? (in Worten, dezimal, Byte 0 rechts)	
0	Y	8	1	0	Y	5	1	0
1	Y	8	3	2	Y	5	3	2
2	Y	8	5	4	Y	5	5	4
…	Y				Y			
123	Y	8	7	6	Y	2	7	6
124	Y	8	9	8	Y	5	9	8
125	Y	8	11	10	Y	0	11	10 a0
126	Y	8	13	12	Y	5	13	12
…								

Tabelle 5: Veränderungen im First-Level-Cache

Codezeile	Index	D	Tag	Speicherblock (in Worten, dezimal, Byte 0 rechts)	
2	61	1	1	11	10 a0
5	61	1	17	16 a2	0
6	61	1	1	11	10

Tabelle 6: Veränderungen im Second-Level-Cache

Codezeile	Set Index	D	Tag	Speicherblock 1 (in Worten, dezimal, Byte 0 rechts)		D	Tag	Speicherblock 2 (in Worten, dezimal, Byte 0 rechts)	
2	125	0	8	11	10	1	0	5	4
5	125	0	8	11	10	1	0	11	10
6	125	0	8	11	10	1	0	16	0

Abbildung 1: Hauptprogramm

```
1     ARRSUM: beq $a3,$zero,BACK
2             lw $s0,0($a0)
3             lw $s1,0($a1)
4             add $s2,$s0,$s1
5             sw $s2,0($a2)
6             addi $a0,$a0,4
7             addi $a1,$a1,4
8             addi $a2,$a2,4
9             addi $a3,$a3,-1
10            j ARRSUM
11       BACK: jr $ra
```