

# CT Praktikum: Cache

## 1 Funktion

In diesem Praktikum lernen sie die Funktionsweise eines Direct-Mapped-Cache kennen. Abbildung 1 zeigt die schematische Darstellung eines solchen Cache. Anhand eines einfachen Programmes mit verschachtelter Schleife sollen die Speicherzugriffe auf den Cache untersucht und durch Verändern der Cache-Parameter optimiert werden.

## 2 Lernziele

- Sie verstehen wie ein Direct-Mapped-Cache aufgebaut ist.
- Sie können die Größen Tag, Index und Offset erklären.
- Sie können die Hit-Rate eines Cache optimieren.

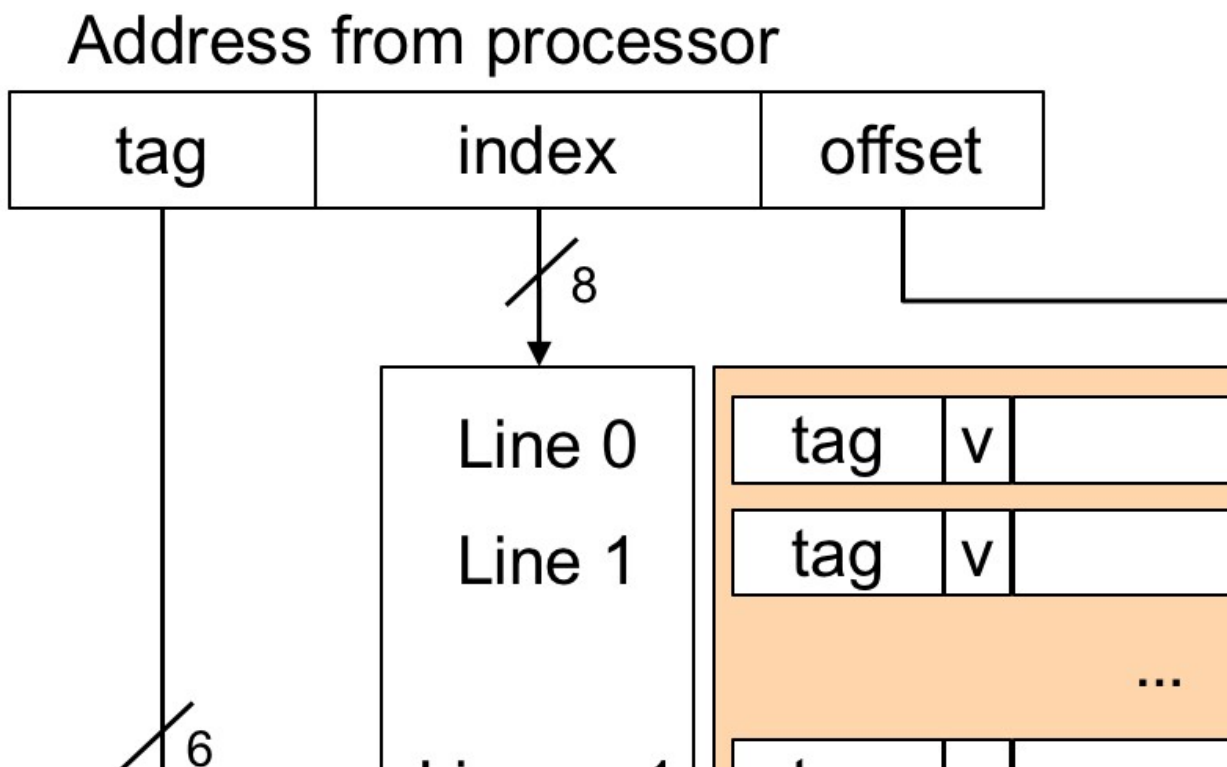


Abbildung 1: Schematische Darstellung eines Direct-Mapped-Cache

### 3 Aufgaben

Ein Direct-Mapped Cache soll am einfachen Beispiel einer Summation von zwei Arrays untersucht werden. Dazu wird der Cache auf dem CT-Board simuliert.

#### 3.1 Speicherzugriffe bei Arrays

Analysieren Sie folgenden Beispielcode:

```
uint8_t a[ARRAY_ROWS][ARRAY_COLUMNS];
uint8_t b[ARRAY_ROWS][ARRAY_COLUMNS];
uint8_t c[ARRAY_ROWS][ARRAY_COLUMNS];

/* Loop through columns */
for (int j = 0; j < ARRAY_COLUMNS; j++) {
    /* Loop through rows */
    for (int i = 0; i < ARRAY_ROWS; i++) {
        a[i][j] = b[i][j] + c[i][j]
    }
}
```

Für diese Aufgabe werden folgende Konfigurationen der Arrays verwendet:

Anzahl Zeilen	5
Anzahl Spalten	10
Grösse eines Elements	1 Byte
Startadresse des ersten Arrays (a)	0x0000'0000

##### 3.1.1 Adressen?

Die Arrays werden wie im Code oben hintereinander deklariert. Welches sind die Adressen der ersten Elemente als HEX-Zahlen?

Element	Adresse des ersten Elements
a[0][0]	0x0000'0000 -> 0
b[0][0]	0x0000'0032 -> 1*50d -> 0x32
c[0][0]	0x0000'0064 -> 2*50d -> 0x64

### 3.1.2 Dimensionen RAM / Cache

Für die Simulation werden folgende Parameter für den Cache verwendet:

Anzahl Bits für die Adressierung	11
Anzahl Bits für den Offset	2
Anzahl Bits für den Index	2
Anzahl Bits für den Tag	7

Bestimmen Sie folgende Dimensionen des RAM und Cache:

Grösse des RAM in Bytes (Memory Size)	$2^{11} * 1 \text{ Byte} = 2048 \text{ Bytes}$
Anzahl Zeilen im Caches	$2^2 \text{ (Offset)} = 4 \text{ Zeilen}$
Grösse einer Cache-Zeile in Bytes (nur Nutzdaten)	$2^2 \text{ (Index)} = 4 \text{ Bytes}$
Grösse des Cache in Bytes (Cache Size)	$4 \text{ Zeilen (Offset)} * 4 \text{ Bytes (Index)} = 16 \text{ Bytes}$

### 3.2 Simulation

In dieser Aufgabe werden die Speicherzugriffe der vorherigen Aufgabe auf dem CT-Board simuliert.

Die Funktionen des Cache sind im Header-File `cache.h` zu finden. Arbeiten Sie sich in die entsprechenden Funktionen ein. Analysieren Sie dann im File `cache.c` die Funktion `access_cache(uint32_t address)`.

Lesen Sie zusätzlich das Header-File `config.h`, um einen Überblick über die Konfiguration des Cache und der Arrays zu erhalten. Kontrollieren Sie ob die Konfiguration mit der vorherigen Aufgabe übereinstimmt.

Schauen Sie sich nun die Funktion `run_simulation(void)` im File `main.c` und vergleichen Sie die Simulation mit dem Beispielcode der vorherigen Aufgabe.

Wo unterscheidet sich der Code der Simulation mit dem des Beispiels?

`a_equals_b_plus_c(i, j);` anstatt `a[i][j] = b[i][j] + c[i][j]`

---

Warum unterscheidet sich der Code der Simulation mit dem des Beispiels?

Damit der Zugriff aufs Memory durch den Cache simuliert werden kann

---

**Hinweis:** Sie können die Simulation mit den Tasten T0 und T1 steuern (T1 -> Single Step / T0 -> fortlaufend).

Die Simulation ist erfolgreich durchgelaufen, wenn das LCD blau leuchtet die Anzahl der Hits und Misses angezeigt werden

Kompilieren Sie nun das Programm und laden Sie es auf das CT-Board. Lassen Sie zuerst das Programm schrittweise durch Drücken der Taste T1 laufen. Stimmen die Adressen mit der vorherigen Aufgabe 3.1.1 überein?

- Achten Sie auf die Hits und Misses
- Wie viele Hits / Misses gab es?
- Berechnen Sie die Hit- und Miss-Rate

Anzahl Hits	0
Anzahl Misses	150
Hit-Rate	$0/150 * 100 = 0\%$
Miss-Rate	$150/150 * 100 = 100\%$

### 3.2.1 Schleifen optimieren

Optimieren Sie nun die Funktion `run_simulation(void)`, so dass Sie eine höhere Hitrate erzielen.

Was haben Sie geändert?

Row und Column gewechselt und variable i und j

Welche Werte erhalten Sie nun und wann tritt der erste 'Hit' auf?

Erster Hit bei Adresse	0x0000'0065
Anzahl Hits	37
Anzahl Misses	113
Hit-Rate	$37/150 * 100 = 24.67\%$
Miss-Rate	$113/150 * 100 = 75.333\%$

Zur Veranschaulichung verwenden Sie nun den Direct-Mapped Cache Simulator im Internet unter <https://www3.ntu.edu.sg/home/smitha/ParaCache/Paracache/dmc.html>. Konfigurieren Sie diesen mit den gleichen Parametern für Cache Size, Memory Size und Offset Bits wie unter 3.1.2 angegeben. Geben Sie nun die ersten Memory Load-Zugriffe bis zum ersten 'Hit' ein, wie gerade auf dem CT-Board getestet. Verwenden Sie direkt die Hex-Zahlen, die Sie bei Single-Step auf dem CT-Board sehen. In welcher Zeile tritt der erste 'Hit' auf? Warum nicht früher?

### 3.3 Cache optimieren

Optimieren Sie die Cache-Parameter im File `config.h` für die vorherigen Aufgaben. Ändern Sie schrittweise die Parameter für Offset und Index, lassen Sie jedoch den Wert für `ADDRESS_SIZE` auf 11 stehen. (Achtung:  $\text{OFFSET} + \text{INDEX} < \text{ADDRESS\_SIZE}$ )

Mit welchen Werten erhalten Sie die besten Resultate?

Anzahl Bits für den Offset	8
Anzahl Bits für den Index	2

Was stellen Sie fest? Wie gross ist nun der Cache gegenüber dem gesamten Memory?

Umso höher der Offset Bit desto besser sind die Resultate -> 50Bytes \* 3 Arrays = 150 Bytes ->  $2^8$

Adressen können mit einerm Cache line alle Daten abbilden

### 3.4 Grosses Array

Stellen Sie nun wieder die gleichen Parameter wie in Aufgabe 3.1.2 für den Cache ein. Ändern Sie diesmal aber die Array Grösse auf die folgenden Werte:

Anzahl Zeilen	50
Anzahl Spalten	10

Welche Werte stellen Sie fest?

Anzahl Hits	1125
Anzahl Misses	375
Hit-Rate	$1125/1500 * 100 = 75\%$
Miss-Rate	$375/1500 * 100 = 25\%$

Vergleichen Sie die Performance mit Aufgabe 3.2.1. Machen Sie schliesslich Ihre Änderungen in `run_simulation(void)` im Abschnitt 3.2.1 wieder rückgängig und schauen Sie sich die Hit-Rate an.

### 3.5 Bewertung

Die lauffähigen Programme müssen präsentiert werden. Die einzelnen Studierenden müssen die Lösungen und den Quellcode verstanden haben und erklären können.

Bewertungskriterien	Gewichtung
Speicherzugriffe bei Arrays	1/4
Simulation	1/4
Cache optimieren	1/4
Grosses Array	1/4