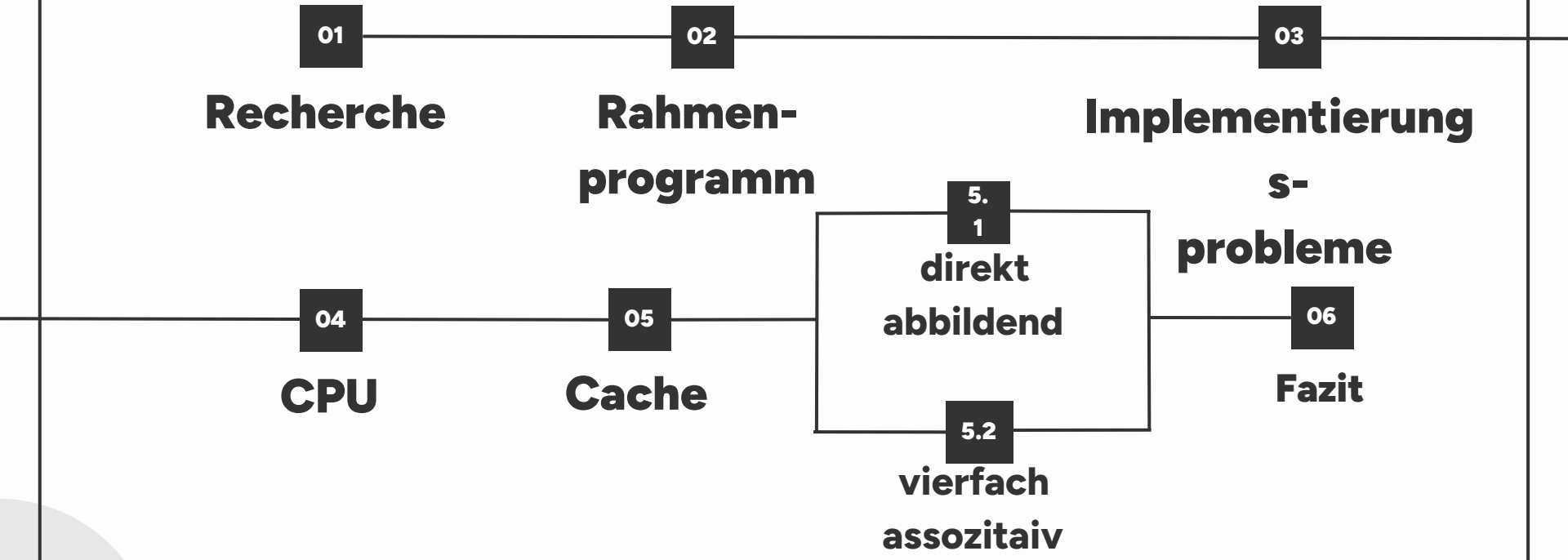
Grundlagenpraktikum: Rechnerarchitektur:

Cache Simulation und Analyse (A11)

Projektaufgabe – Aufgabenbereich Cacheassoziativität

Tim Albrecht, Alptuğ Ecer, Ömer Baha Namal

Inhalt


01
Recherche
02
Rahmen-
programm
03
Implementierung
s-
probleme
04
CPU
05
Cache
5.
1
direkt
abbildend
5.2
vierfach
assozitaiv
06
Fazit

01

Recherche

Problemstellung

- Cachesimulation in SystemC
  - Direkt abbildender Cache
  - Vierfach assoziativer Cache
- Unterschiede / Vor- und Nachteile
- Verhalten der Caches an konkreten Beispielen testen
- Ergebnisse mit der Recherche vergleichen

Latency

Cache / Latency	Size	Cycles	Nanoseconds
L1	32KB / core	4	1.5072
L2	1024KB / core	14	5.2752
L3	1.375MB / core (33MB / socket)	50 - 70	18.84 - 26.37

Unterschiede / Vor- und Nachteile

Kriterien	**Direkt abbildender Cache**	**Vierfach assoziativer Cache**
Implementierung	einfach	kompliziert
Geschwindigkeit / Zugriff	schnell	langsam
Energieverbrauch	niedrig	hoch
# Conflict-Misses	viele	wenig
Effizienz	niedrig	hoch
Geschwindigkeit insgesamt	**langsam**	**schnell**

02

Rahmenprogramm

Options:

```
"Positional arguments:\n"
"  inputFile   The file to get operations. Must be a .csv file.\n"
"\n"
"Optional arguments:                (Default: 32KB 4-way associative L1 cache)\n"
"  -c, --cycles <number>            Number of cycles to simulate (Default: 1000000)\n"
"      --directmapped               Simulate a direct-mapped cache. Can't set with --fourway option simultaneously (Default: directmapped)\n"
"      --fourway                    Simulate a four-way associative cache. Can't set with --directmapped option simultaneously (Default: directmapped)\n"
"      --cacheline-size <number>    Size of a cache line in bytes (Default: 64)\n"
"      --cachelines <number>        Number of cache lines (Default: 512)\n"
"      --cache-latency <number>     Cache latency in cycles (Default: 1)\n"
"      --memory-latency <number>    Memory latency in cycles (Default: 200)\n"
"      --tf=<filename>              Output trace file with all signals\n"
"  -h, --help                       Print this help message and exit\n";
```

Eingabedatei

- In der Form von CSV (Comma Separated Values) mit der Dateierweiterung “.csv”. Jede Zeile beschreibt eine “Request” von der CPU

- 1. Spalte definiert die Art des Befehls (Read / Write)
- 2. Spalte definiert die Adresse (als Dezimal / Hexadezimal)
- 3. Spalte definiert die zu schreibenden Daten (Dezimal / Hexadezimal)
    - nur bei Write-Befehlen gegeben

Programmverhalten

- Die von der Eingabedatei gelesenen Werte werden im Programm als 32-bit Integer behandelt
- Ein Cache kann nicht gleichzeitig direkt abbildend und vierfach assoziativ sein
- Wenn eine Request ein oder mehrere nicht-valide chars oder whitespaces beinhaltet, so wird die ganze Request als “invalid” behandelt.

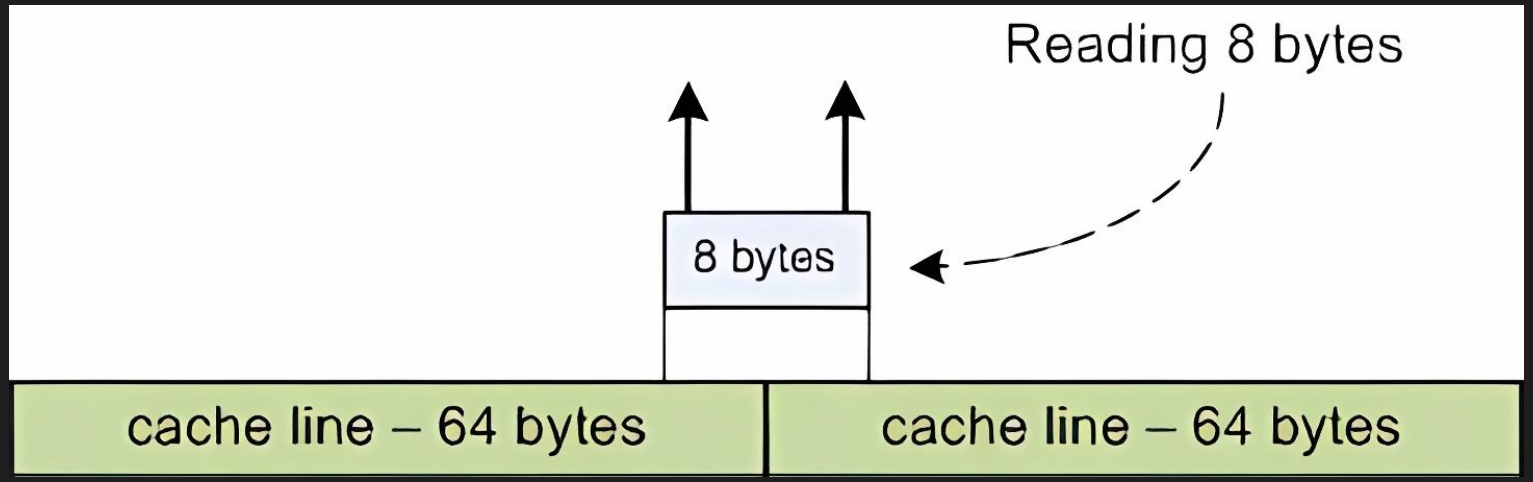
03

Implementierungs-probleme

WIE...

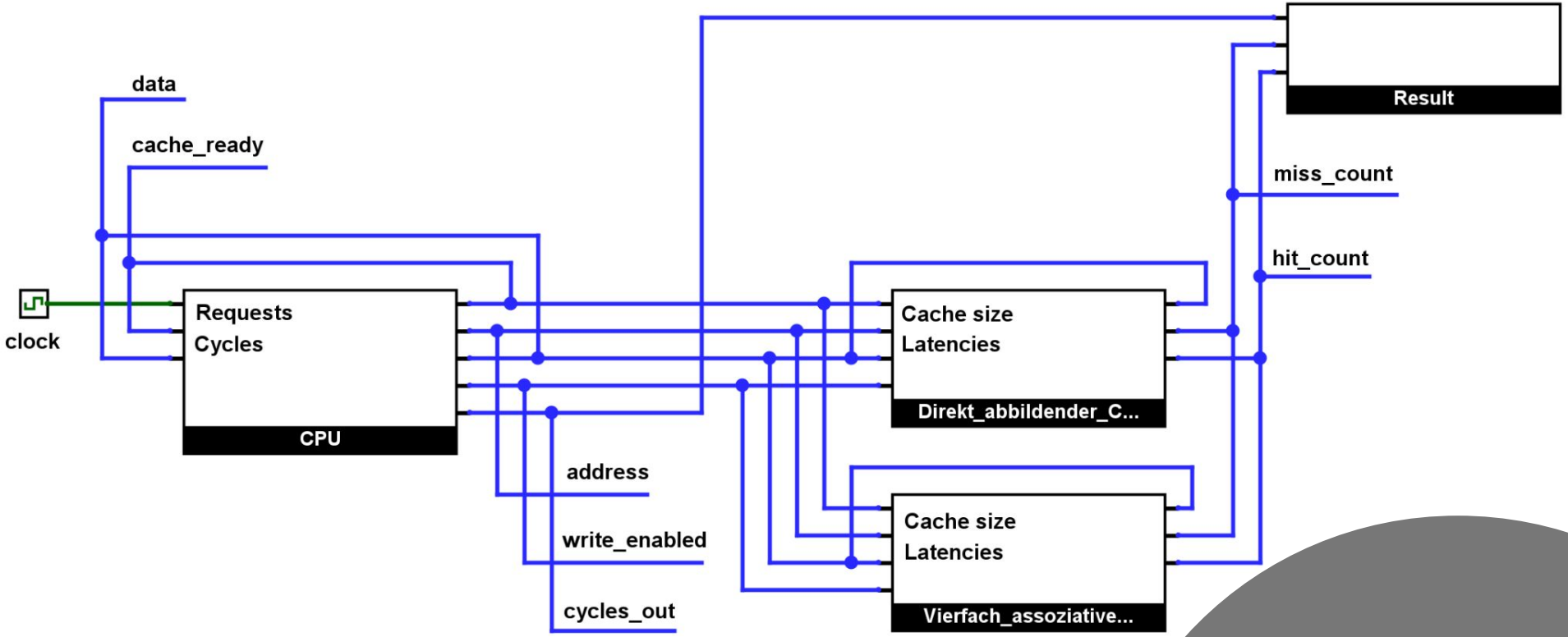
- werden Hauptspeicher und Cache im Code dargestellt
- bekommen / geben wir die Instruktionen / die Ergebnisse aus
- werden die Zyklen berechnet
- wird die korrekte Reihenfolge der Requests eingehalten
- wählen wir den Ersetzungsalgorithmus

Unaligned Cache Zugriffe



04

CPU



Darstellung der Bestandteile als Schaltbild

CPU

- wird hauptsächlich zum Senden der Requests an die Caches benutzt

- erhöht die Abstraktion
- stellt sicher, dass beide Caches mit den gleichen Signalen funktionieren

- aktualisiert read-Requests

- bricht die Simulation ab wenn die Anzahl an gegebenen Cycles erreicht wurde oder alle Requests bearbeitet wurden

CPU Pseudocode

```
void run() {
    while(true) {
        cycles++;
        if(cache_ready()) {
            send_request_to_cache();
        }

        wait();

        if(cache_ready()) {
            update_data_of_request();

            if(all_requests_done()) {
                break;
            }
        }

        if(max_cycles_reached) {
            break;
        }
    }

    if(!all_requests_done() && !cache_ready()) {
        cycles->write(SIZE_MAX);
    }
    sc_stop();
}
```

05

Cache

Gemeinsamkeiten

- Initialisierung:

```
SC_CTOR(CACHE);
CACHE(sc_module_name name, unsigned cacheLines, unsigned cacheLinesSize,
unsigned cacheLatency, unsigned memoryLatency,
unsigned offsetBitsCount, unsigned offsetBitsMask,
unsigned indexBitsCount, unsigned indexBitsMask) :
```

- Darstellung von Main Memory:

```
std::map<uint32_t, uint8_t> mainMemory;
```

... Cachezeile:

```
struct CacheLine {
    unsigned tag = 0;
    bool valid = false;
    std::map<uint32_t, uint8_t> data;
};
```

Direkt abbildender Cache

- Darstellung:

```
std::map<uint32_t, CacheLine> cache;
```

Direkt abbildender Cache Pseudocode

```
void processRequest() {
    read_request();
    while(true) {
        if(we) { // write
            for(int i = 0; i < 4; ++i) {
                split_address();
                if(is_miss(index, offset)){
                    load_Cacheline_from_main_memory(address);
                    wait(memoryLatency);
                }
                write_into_cache(addr, data);
                write_into_main_memory(addr, data);
            }
            wait(cacheLatency);
            update_result();

        } else { // read
            for(int i = 0; i < 4; ++i) {
                split_address();
                if(is_miss(index, offset)){
                    load_Cacheline_from_main_memory();
                    wait(memoryLatency);
                }
                read_data_from_cache(tempData);
            }
            wait(cacheLatency);
            update_data(tempData);
            update_result();
        }
        wait();
    }
}
```

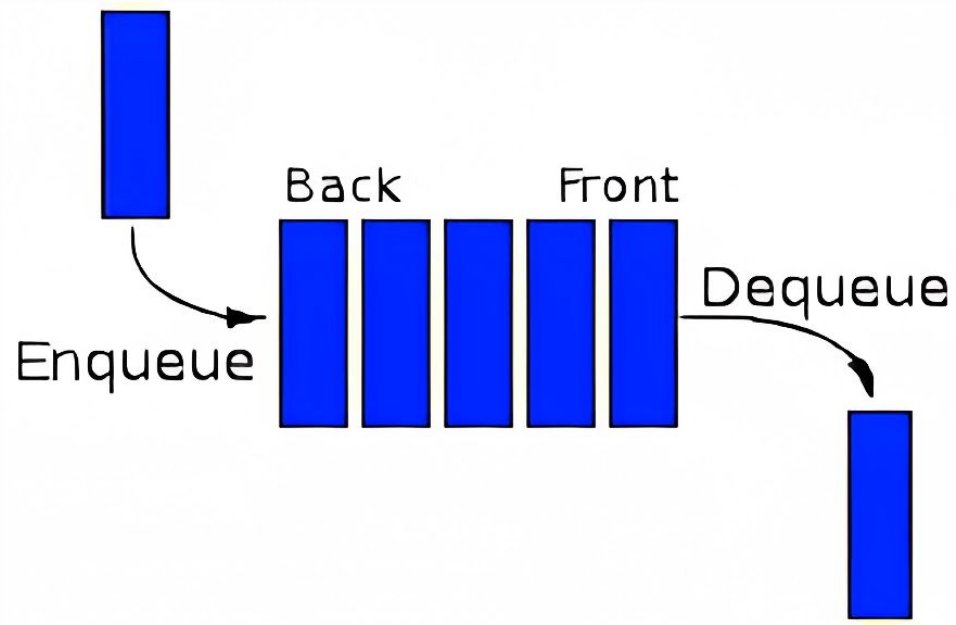
Vierfach assoziativer Cache

- Darstellung:

```
std::map<uint32_t, std::deque<CacheLine>> cache;
```

FIFO als Cache-Algorithm

Andere Möglichkeiten:
LRU LFU LIFO



Gatterberechnung: Direct-Mapped Logik

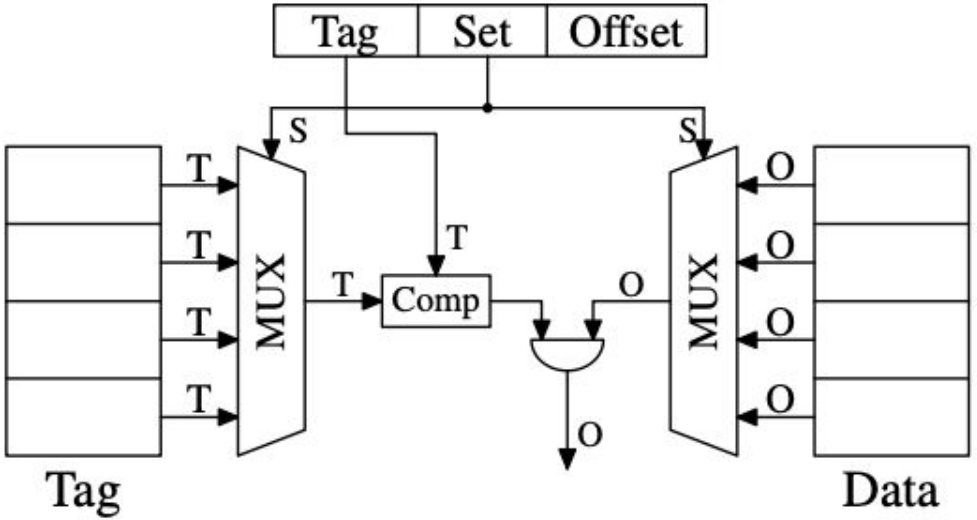
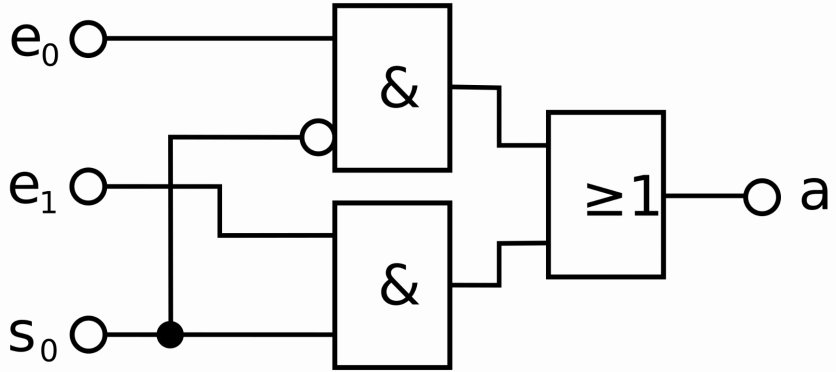
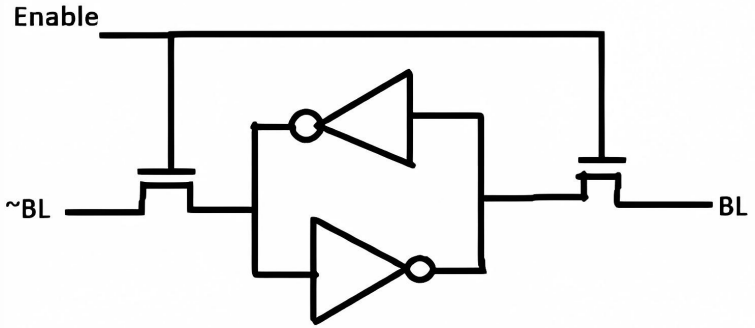


Figure 3.6: Direct-Mapped Cache Schematics

MUX



SRAM



Gatterberechnung: Assoziativ Logik

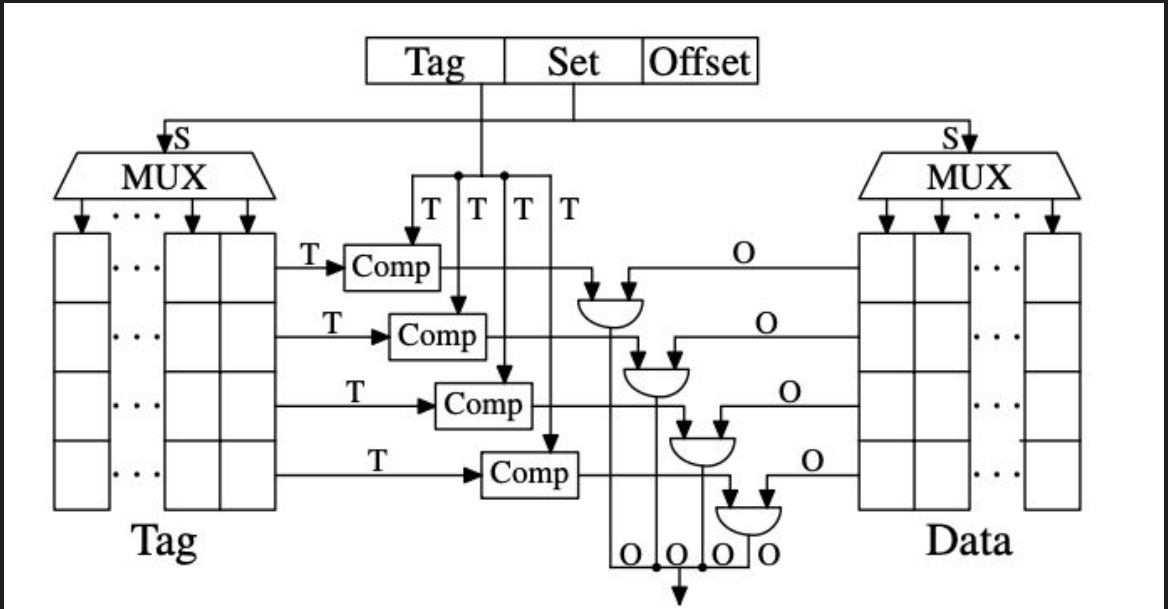
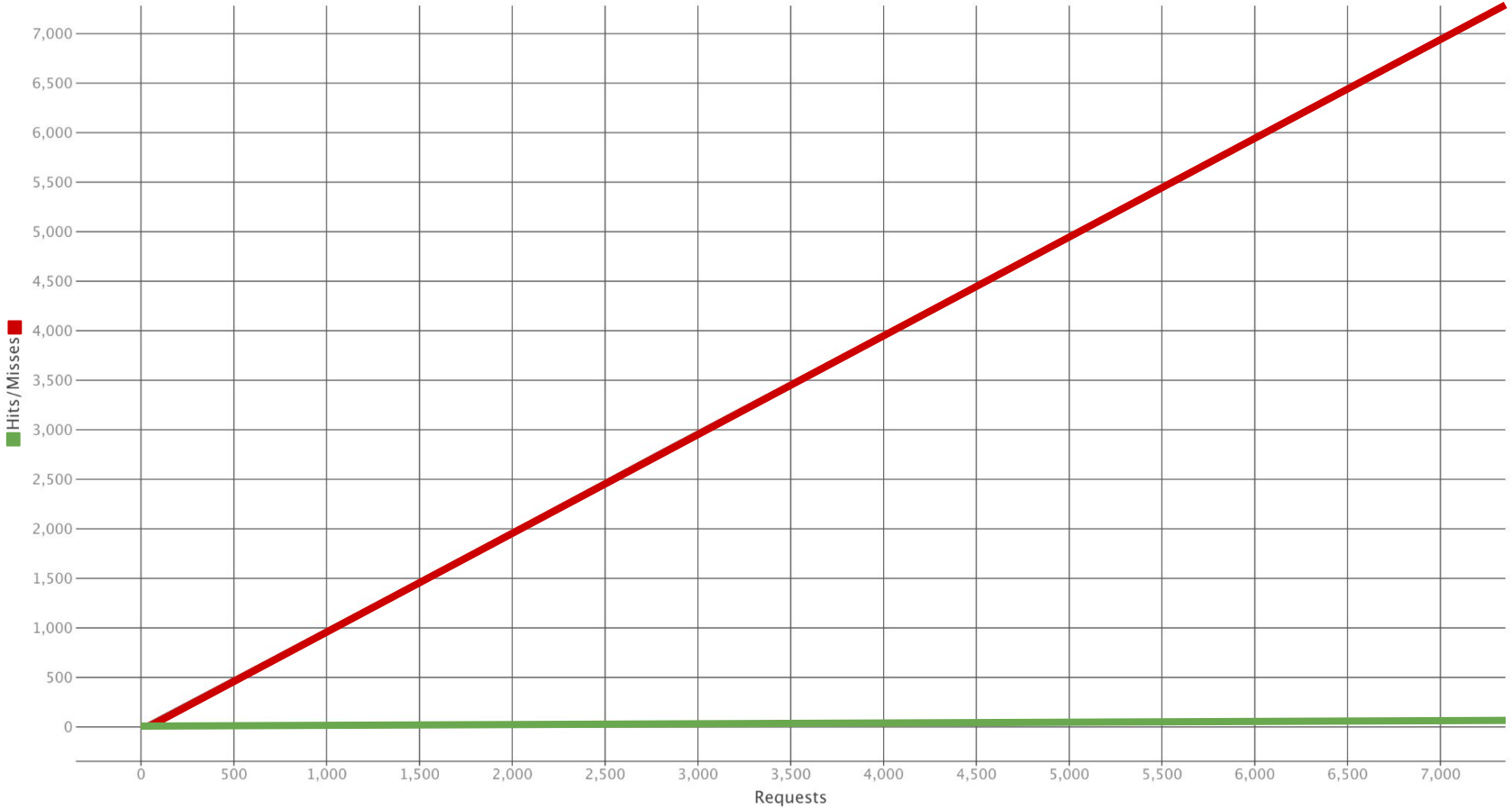


Figure 3.7: Set-Associative Cache Schematics

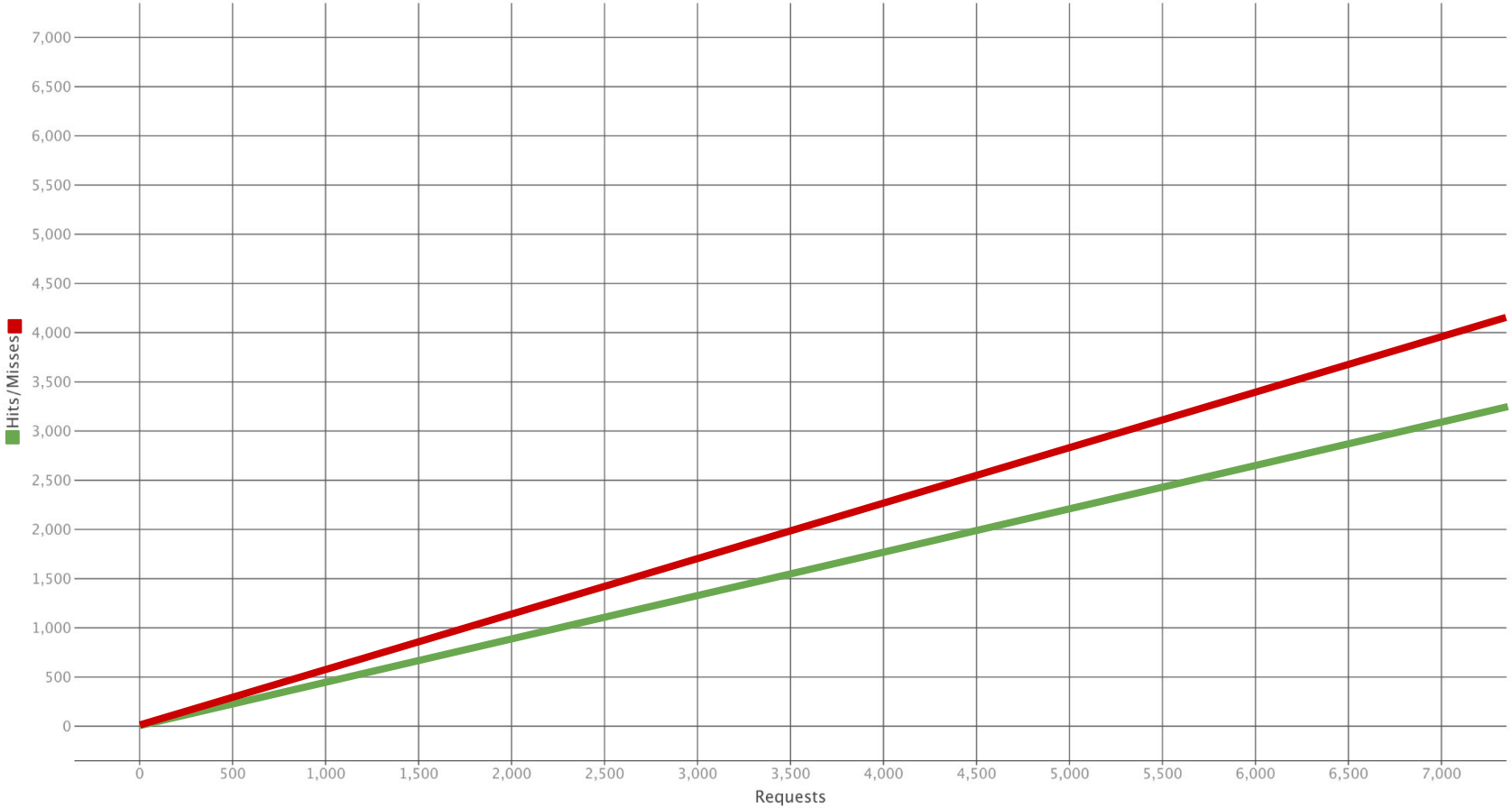
06

Fazit

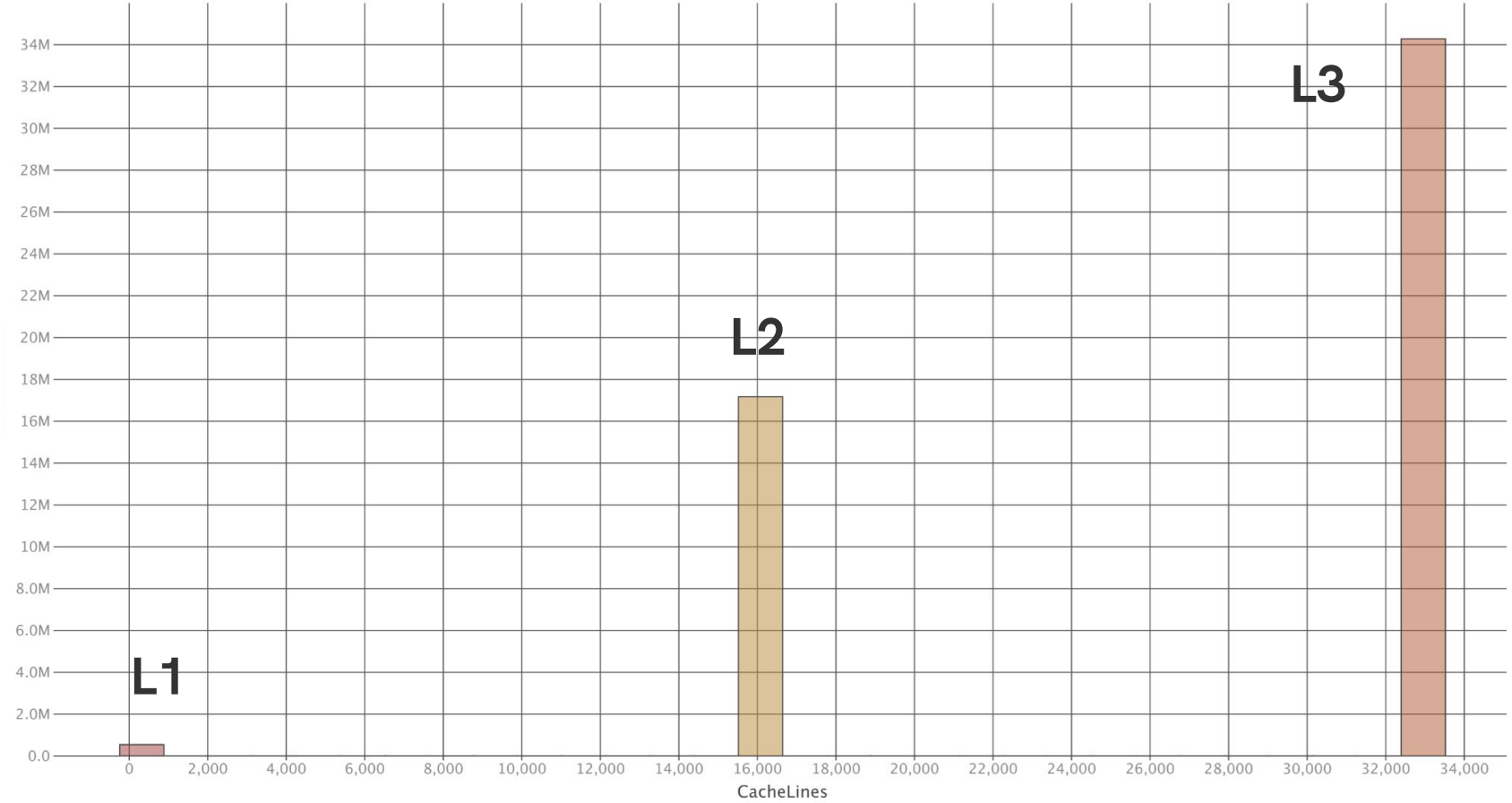
Hit/Miss Verhältnis beim Testen des direkt abbildenden Caches



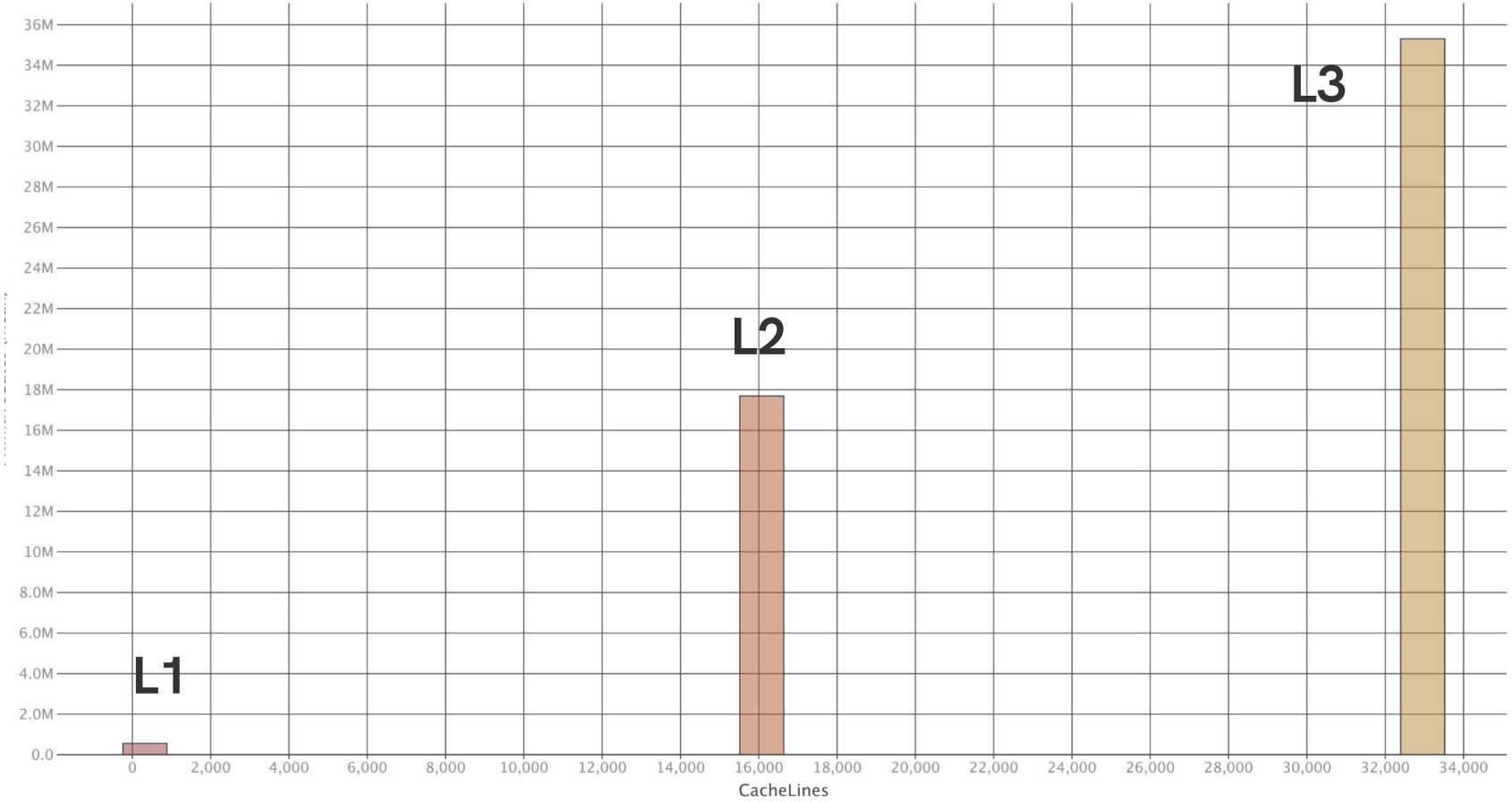
Hit/Miss Verhältnis beim Testen des vierfach assoziativen Caches



Wachstum der primitiven Gatter mit der Anzahl der Cachezeilen im direkt abbildenden Cache



Wachstum der primitiven Gatter mit der Anzahl der Cachezeilen im vierfach assoziativen Cache



Fazit

- Rechercheergebnisse wurden bestätigt
  - Vierfach assoziativer Cache tatsächlich schneller bei sinnvollen Eingaben
  - Direkt abbildender Cache kann aber auch schneller sein (z.B. zufällige Adressen)

- Unterschied von Anzahl der Gatter kleiner als erwartet

→ Vierfach assoziativer Cache quasi keine Nachteile und kann bedenkenlos benutzt werden

Unterschiede / Vor- und Nachteile

Kriterien	**Direkt abbildender Cache**	**Vierfach assoziativer Cache**
Implementierung	einfach	kompliziert
Geschwindigkeit / Zugriff	schnell	langsam
Energieverbrauch	niedrig	hoch
# Conflict-Misses	viele	wenig
Effizienz	niedrig	hoch
Geschwindigkeit insgesamt	**langsam**	**schnell**

Quellen

https://www.7-cpu.com
https://www.cpubenchmark.net
https://www.intel.com/content/www/us/en/developer/articles/technical/memory-performance-in-a-nutshell.html
https://www.amd.com/en/products/specifications/processors.html
What Every Programmer Should Know About Memory by Ulrich Drepper; Red Hat, Inc published November 21, 2007; https://www.semanticscholar.org/paper/What-Every-Programmer-Should-Know-About-Memory-Drepper/aea8f512bad854c509e6d4f9c093a6224b4045f3
https://www.instructables.com/DIY-SRAM-The-RAM-in-your-Microcontroller/
https://dev.to/satrobit/cache-replacement-algorithms-how-to-efficiently-manage-the-cache-storage-2ne1
https://upload.wikimedia.org/wikiversity/de/c/c7/2_zu_1_MUX.pdf

Vielen Dank für Ihre Aufmerksamkeit!