Exercise sheet 10 – Associative memory

Goals:

- Associative memory (Cache, TLB)

Exercise 10.1: Cache (theoretical)

Consider an architecture with the following details:

- 32 bit architecture
- Memory address: 0x00014492
- Data (`uint16_t`): 0xAFFE

(a) State the cache entry, considering a cache line size of *2 bytes*.

Proposal for solution:

Key (real adr.) **Value (data: byte 0 to 3)**

	#0	#1
0x00014492	0xAF	0xFE

*This is the view for a BE (big endian) architecture.

(b) State the cache entry, considering a cache line size of *256 bytes*.

Proposal for solution:

Key (real adr.) **Value (data: byte 0 to 3)**

	#0	...	#92	#93	...	#FF
0x000144	?	...	0xAF	0xFE	...	?

*This is the view for a BE (big endian) architecture.

(c) Exactly state the position of the value (0xAFFE) inside the 256 byte data entry.

Proposal for solution: Check out the solutions for the previous part.
0xAFFE is 2 bytes, which is the size of a word for the given 16 bit architecture.
For a block size of 256 bytes, 0xAF is at position #94, and 0xFE on #95, respectively (for those who wants to know it in detail: for a BE architecture!).

(d) What is happening if the cache is empty and we try to access the cache line?

Proposal for solution:
First: Load the data for the whole cache line from the memory into the cache.
Then: The data can be accessed through the cache, e.g. loaded into a register.

Exercise 10.2: Cache architecture

Hint: Use the „ARMv7-M Processor Architecture Reference Manual“ to answer that question: `CA_exercises/sheet_10_memory_cache_tlb/DDI0403E_d_armv7m_arm.pdf`

(a) How many hierarchy levels has the ARMv7 memory system? *You may have a look on chapter A3.8.2.*

Proposal for solution: As shown in figure A3-9 on page A3-97, the ARMv7 has four levels of memory. Two of them are explicitly referred as cache *(one instruction cache and one data cache as minimum)*. The architecture itself may support up to seven levels of cache.

(b) Caches are largely invisible to application programmers. State a situation where a breakdown in cache coherency might occur. *You may have a look on chapter A3.8.3.*

Proposal for solution: Referred to chapter A3.8.3, such a breakdown might occur if the processor is writing new data into the cache, while the DMA *(direct memory access)* is reading the old data. But of course there are more possibilities to break coherency in cache.

(c) How would you get some information such as the *type of a cache* or its *level of coherency* or the *line size*, considering your software has privilege access? You may find chapter B4.8 interesting for answering that question.

Proposal for solution: As stated in chapter B4.8, we have some special registers for getting cache information, summarized as the *Cache Control Identification Registers*.

(d) Looking at the Cache Size ID Register, state a formula to calculate the cache size and explain each operand.

Proposal for solution: $cachesize = NumSets * Associativity * LineSize$
NumSets: Number of sets in cache
Associativity: Number of possible places, where an entry can be stored.
LineSize: Number of words in each cache line. To calculate a solution in bytes, convert this value to bytes.

Exercise 10.3: TLB 1

Consider an architecture for page addressing similar to the 1 level page table in the lecture about the VM/MMU, but with a 32 bit architecture.

- The TLB *(translation lookaside buffer)* contains the following entry:

Key	Value
0x00009	0x00002

- The page table contains the following entries:

Page table offset	Value
...	
0x0000A	0x00003
0x00009	0x00002
0x00008	0x00001
...	

(a) Is the page table and the TLB consistent?

Proposal for solution: TLB-Key: 0x00009
TLB-Value: 0x00002
Page-Table-Entry 0x00009 contains 2
Yes they are consistent.

(b) State the virtual address, which is mapped to the 32 bit real address 0x00002AAA.

Proposal for solution: The *value* in the TLB is the real address, the corresponding *key* is the virtual address, the offset can be extracted from the given real address.
Offset: 0xAAA
Frame number: 0x00002
Page number: 0x00009
Virtual address: 0x00009AAA

Exercise 10.4: TLB 2

Consider an architecture for page addressing similar to the 2 level page table in the lecture for a 32 bit architecture with a 4 KiB page/frame size.
Given:

- Virtual address: 0x1202F494
- Real address: 0x00014494

(a) State the entry in the TLB for the given addresses.

Proposal for solution:

Page base address	Frame base address
0x1202F	0x00014

(b) What is happening if the TLB is empty and we try to access the page?

Proposal for solution:

- Load page table(s)
- Lookup inside page table(s)
- Address translation
- Store address into TLB