Memory Calculation

Consider a system with the following specifications:

OS

- 1MB page sizes (2^{20})
- 32-bit physical address space
- 48-bit virtual address space
- Assumption: 1-level page table
- PTE and TLB entries contain 4 bits of metadata [R W X I]

Cache

- 16KiB directed-mapped L1 cache with 16-byte cache-lines.
- 16KiB fully-associative L1 cache with 16-byte cache-lines.
- 64KiB 4-way set-associative L1 cache with 16-byte cache-lines.
- Assumption: All accesses are 4 bytes long and entirely complete before the next one starts.

Source: OS, Arch Course; CS 152: Computer Architecture and Engineering Final Exam

OS

By default, in unit of bit. Note 1B = 8 bits. (0x _ _ in HEX)

Given 1MB page sizes (2^{20}) Page size = Offset size Page/Frame Offset = $log_2(1MB)$ = 20

physical address (RAM)

address (32)	Frame No. (12)	Frame Offset (20)
0x _ _ *4	p	d

logical address (physical + secondary storage)

- Physical (Valid)
- Secondary Storage (Invalid)

address (48)	Page No. (28)	Page Offset (20)
0x _ _ *6	p'	d

Extension: 2-level Page Table

addr (48)	Frame No. (14)	Frame No. (14)	Frame Offset (20)
0x _ _ *4	p1	p2	d

page table

- PTE is indexed by page number
- PTE size (16) = frame number (12) | RWXI (4)

Page Table index	Frame No. (12)	Metadata (4)
Page No. p'	p	W I

Translation Look-aside Buffer

- TLB is hash table, no index as hardware process it in parallel
- TLB entry size (44) = page number (28) | frame number (12) | RWXI (4)

	Page No. (28)	Frame No. (12)	Metadata (4)
in parallel	p'	p	W I

Cache

16KiB L1 cache with 16-byte cache-lines.

All accesses are 4 bytes long (1 word) and entirely complete before the next one starts.

- Warning: distinguish between data address and value

```
READBYTE 0x FFFF FFFF
```

direct-mapped cache

16KiB directed-mapped L1 cache with 16-byte cache-lines. Also known as 1-Way Set Associative.

Total number of cache lines 16KiB/16B= 2^{10}

Cache index $log_2(2^{20}) = 10$ bits

Case 1: Byte Select

- Each Cache line has *16 bytes*
- So the byte select has $log_2(16) = 4$ bits

Address (32)	Tag (18)	Index (10)	Offset (4)
0x _ _ *4	upper address bits	index	byte select

- Direct-mapped cache is indexed by index
- Each cache line has 16 bytes

cache index	Tag (20)	MSI	Byte 1	Byte 2	...	Byte 16
row index (10)	tag	M	0x _ _	0x _ _		0x _ _

Case 2: Word Select

- Each Cache line has 16 bytes, i.e. *4 words*
- So the byte select has $log_2(4) = 2$ bits

address (32)	Tag (20)	Index (10)	Offset (2)
0x _ _ *4	upper address bits	index	word select

- Direct-mapped cache is indexed by index
- Each cache line has 4 words

cache Index	Tag (20)	MSI	Word 1	Word 2	Word 3	Byte 4
Index (10)	tag	S	0x _ _ *4	0x _ _ *4	0x _ _ *4	0x _ _ *4

fully-associative cache

16KiB fully-associative L1 cache with 16-byte cache-lines.

- Fully-associative cache is hash table, no index as hardware check it in parallel

address (32)	Tag (30)	Word Select (2)
0x _ _ *4	upper address bits	offset

cache	Tag (30)	MSI	Word 1	Word 2	Word 3	Byte 4
way 1 in parallel	tag	S	0x _ _ *4	0x _ _ *4	0x _ _ *4	0x _ _ *4

N-way set-associative cache

It lies between the above two cache.

64KiB 4-way set-associative L1 cache with 16-byte cache-lines.

- set sizes = # ways = 4
- number of cache lines = 64KiB/16 = 4KiB
- number of sets = 4KiB/4 = 1KiB ~ 2^{10} , as each set contains 4 ways
- set index = $log_2(2^{10}) = 10$ bits

address (32)	Tag (20)	Index (10)	Offset (2)
0x _ _ *4	upper address bits	index	word select

- One way to simply the calculation is to use 64/4=16KiB direct mapped cache address layout

- but with 4-way cache lines (to check ways in parallel) now for each index

cache index	Way 1	Way 2	Way 3	Way 4
set index (10)	tag + MSI + Word * 4	...	...	tag + MSI + Word * 4