Memory Calculation

Reference:

- OS, CompArch, Ad CompArch @ Cambridge
- CS 152: Computer Architecture and Engineering Final Exam @ UC Berkeley
- Cache: wiki page

By default, the unit used is 1 bit. Note that 1 Byte = 8 bits, i.e. 0x _ _ in HEX.

Given page size of 1MB (2^{20}), Frame Offset (FO) = Page O. (PO) = $\log_2(\text{page size}) = \log_2 1\text{MiB} = 20$.

Physical memory $P = \{1, ..., M\}$ (RAM) is divided into frames.

- If the physical address space is 32-bit, then the physical address space is 2^{32}.
- Physical address = Physical Frame No. (PFN) + Frame Offset (FO),

~ 1-level Page Table

addr (32)	PFN (12)	FO (20)
0x _ _ *4	pfn	o

~ 2-level Page Table

addr (48)	PFN1 (14)	PFN2 (14)	FO (20)
0x _ _ *4	pfn1	pfn2	o

Secondary storage memory $\{1, ..., K\}$, e.g. on disk.

Virtual memory $V = \{1, ..., N\}$, including physical and secondary storage memory, is divided into pages. Usually $N \gg M \gg K$.

- If the virtual address space is 48-bit, then the virtual address space is 2^{48}.
- Virtual address = Virtual Page No. (VPN) + Page Offset (PO),

addr (48)	VPN (28)	PO (20)
0x _ _ *6	vpn	o

Page Table maps the virtual pages to the physical frames $V \to P$, where the offset is the same in both cases, i.e. PO=FO.

- PTE is indexed by virtual page number (VPN).
- PTE (16) = PFN (12) + RWXI (4),

```
| PT index  | PFN (12) | Metadata (4) |
| --------- | -------- | ------------ |
|  VPN (28) | pfn      | 0 W 0 I      |
```

Translation Lookaside Buffer (TLB) $V \to P \cup (K \cup \emptyset)$ (hit or miss).

- fully associative cache with a small number of Page Table Entries (PTE) for faster access.
- TLB entry (44) = VPN (28) + PFN (12) + Metadata (4),

```
|   TLB    | VPN (28) | PFN (12) | Metadata (4) |
| -------- | -------  | -------- | ------------ |
| parallel | vpn      | pfn      | 0 W 0 I      |
```

Summary

Consider a system with the following specifications:

OS

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

Cache

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

Cache: assume all accesses are 4 bytes long and entirely complete before the next one starts.

Warning: distinguish between data address and value, e.g. load a byte (or word) from memory address 0x FFFF FFFF,

```
READBYTE 0x FFFF FFFF
READWORD 0x FFFF FFFC
```

Virtual addressing and caching

Depending on the index and tag addressing mode, there are VIPT, VIVT, PIPT, PIVT.

Two types of problems for virtual memory:

- synonyms (index aliasing): two VAes map to the same PA.

    - coherence problem, e.g., conflicting cache states, whether cached, are flushed.
    - solved by using non-overlapping physical memory for different virtual addressing.
    - or (part of) the cache must be flushed when the virtual-to-physical mapping changes.
- homonyms: same VA map to different PAes.
    - e.g. after context switch, virtual-to-physical mapping changes.
    - solved by adding process ID to the VA or VIPT.

	virtual addressing	**physical addressing**
hit (or index) time	fast, check within offset permission check TLB later	slow, wait TLB translation south bridge hw (address space)
permission check (TLB)	after fetch / copy permission info	before fetch
prefetch-able	✓	× (update without cache)
I/O update with PAes	× not directly	✓
after context switch homonyms	same VA for different PAes ✓ VIVT, × VIPT	distinct physical addresses ×
synonyms/aliasing	✓ (two VAes coherence)	×
others	homonyms unless flush	network package (last bit) not write mergeable (two core write)
example	VIVT	PIPT

Hybrid cache addressing modes, cache = PN(tag + index) + Offset, where

Virtually-Indexed Physically-Tagged (VIPT)

- use virtual index into the physical cache for the physical tag,
- parallel with TLB lookup VPN -> PFN, and check the tag, if
    - index bits taken within the part of not translated PO (rule).

```
|----VPN----|--------PO-------| virtual addressing
|------PFN------|-INDEX + BO--| cache addressing
```

    - limitation: cache size < page size * associativity (# ways).
- ✓ faster / lower latency (hit ✓; TLB miss ×).
- ✓ ~~homonyms~~, where the physical tag PPN + PO can help detecting.
- × synonyms exists iff. (rule) is not satisfied, since multiple VPNs are mapped to the same PFN,

```
|----VPN----|----PO----| virtual addressing
|--PFN--|---INDEX+BO---| cache addressing
```

    - solved by storing part of the VPN in the shared cache.

Physically-Indexed Virtually-Tagged (PIVT)

- × useless and mostly non-existent.

Cache address = Tag [+ Index] + Offset, where

- set index selects the cache line for non-fully associative caches,
- tag checks validity of the cache line (AND gate),
- offset selects the word within the cache line (MUX gate).

addr (48)	VPN (36)	P0 (12)	
cache addr	tag (48-i)	index (i)	offset (2)
0x _ _ *4	upper addr bits	index	word select

Set index is the number of bits needed to select a cache line.

- 16KiB cache with 16 B cache-lines.
- total number of cache lines = cache size / block size = 16KiB / 16B = 2^{10}.
- number of lines / way = number of cache lines / number of ways = 2^{10}.
- set index is $\log_2 (\text{number of lines} \ / \ \text{way})$ = 10 bits.

Hence,

$$\begin{aligned}\text{cache size} &= \#\text{lines} \times \text{block size (bytes} \ / \ \text{line)}. \\ &= \text{associativity (\# ways)} \times 2^{\text{index}}(\text{\# sets}) \times \text{block size (bytes} \ / \ \text{line)}. \\ &= \text{associativity (\# ways)} \times 2^{\text{index}+\text{block offset}}(\text{bytes} \ / \ \text{way})\end{aligned}$$

Cache line (or block) = Tag + MESI + Word 1 + ... + Word 4, which contains 4 words (16 bytes) in total, for spatial locality.

Block offset is the number of bits needed to select a word within a cache line.

- $\log_2 4 = 2$ bits, if we select a word (by default).
- $\log_2 16 = 4$ bits, if we select a byte.

Tag is the upper address bits, i.e. the remaining bits after index and offset.

Direct-mapped cache, also known as 1-way set associative.

Each block only maps to a single cache line.

addr (32)	tag (18)	index (10)	offset (2)
0x _ _ *4	upper addr bits	index	word select

For each cache line = tag + MSI + Word 1 + ... + Word 4,

cache	tag (20)	MSI	word 1	...	word 4
set index (10)	tag	S	0x _ _ *4	...	0x _ _ *4

Fully-associative cache, N*M-way set associative.

Each block can be placed any line in the cache and the hardware checks all cache lines in parallel, i.e. no index.

addr (32)	tag (30)	block offset (2)
0x _ _ *4	upper address bits	offset

For each cache line = tag + MSI + Word 1 + ... + Word 4,

cache	tag (30)	MSI	word 1	...	word 4
parallel	...	.	...	...	...
way i	tag	S	0x _ _ *4	...	0x _ _ *4
parallel	...	.	...	...	...

N-way set-associative cache

Each memory block maps to a set among M sets, each containing N lines, which is a compromise between the two above types of caches.

64KiB (4x) 4-way set-associative L1 cache with the same cache line setup.

- number of cache lines = 64KiB/16 = 4KiB, number of ways = 4,
- number of sets in each way = 4KiB/4 = 1KiB ~ 2^{10},
- set index = $\log_2 2^{10} = 10$ bits.
- i.e. exactly the same as the 64/4=16KiB direct-mapped cache above.

addr (32)	tag (20)	index (10)	offset (2)
0x _ _ *4	upper address bits	index	word select

With 4 ways, to be checked in parallel,

cache	way 1	...	way 4
set index (10)	tag + MESI + Word*4	...	tag + MESI + Word*4