Virtual memory & MMU II

CompSys304 – Computer Architecture

A/Prof Oliver Sinnen

Department of Electrical, Computer, and Software Engineering

Content

Performance problem?

Isn't there a performance problem???

Performance problem?

Isn't there a performance problem???

- Translating every address (by looking into a table in memory) before it is accessed
  - I.e read/write would require two memory accesses, plus translation
    - 1st page table, 2nd data at address
  - More potential for cache misses

Performance problem?

Isn't there a performance problem???

- Translating every address (by looking into a table in memory) before it is accessed
  - I.e read/write would require two memory accesses, plus translation
    - 1st page table, 2nd data at address
  - More potential for cache misses

Solution:

TLB – translation lookaside buffer

- Page table cache (yes, caches again ☺)
- In MMU of every modern CPU

Performance problem?

Isn't there a performance problem???

- Translating every address (by looking into a table in memory) before it is accessed
  - I.e read/write would require two memory accesses, plus translation
    - 1st page table, 2nd data at address
  - More potential for cache misses

Solution:

TLB – translation lookaside buffer

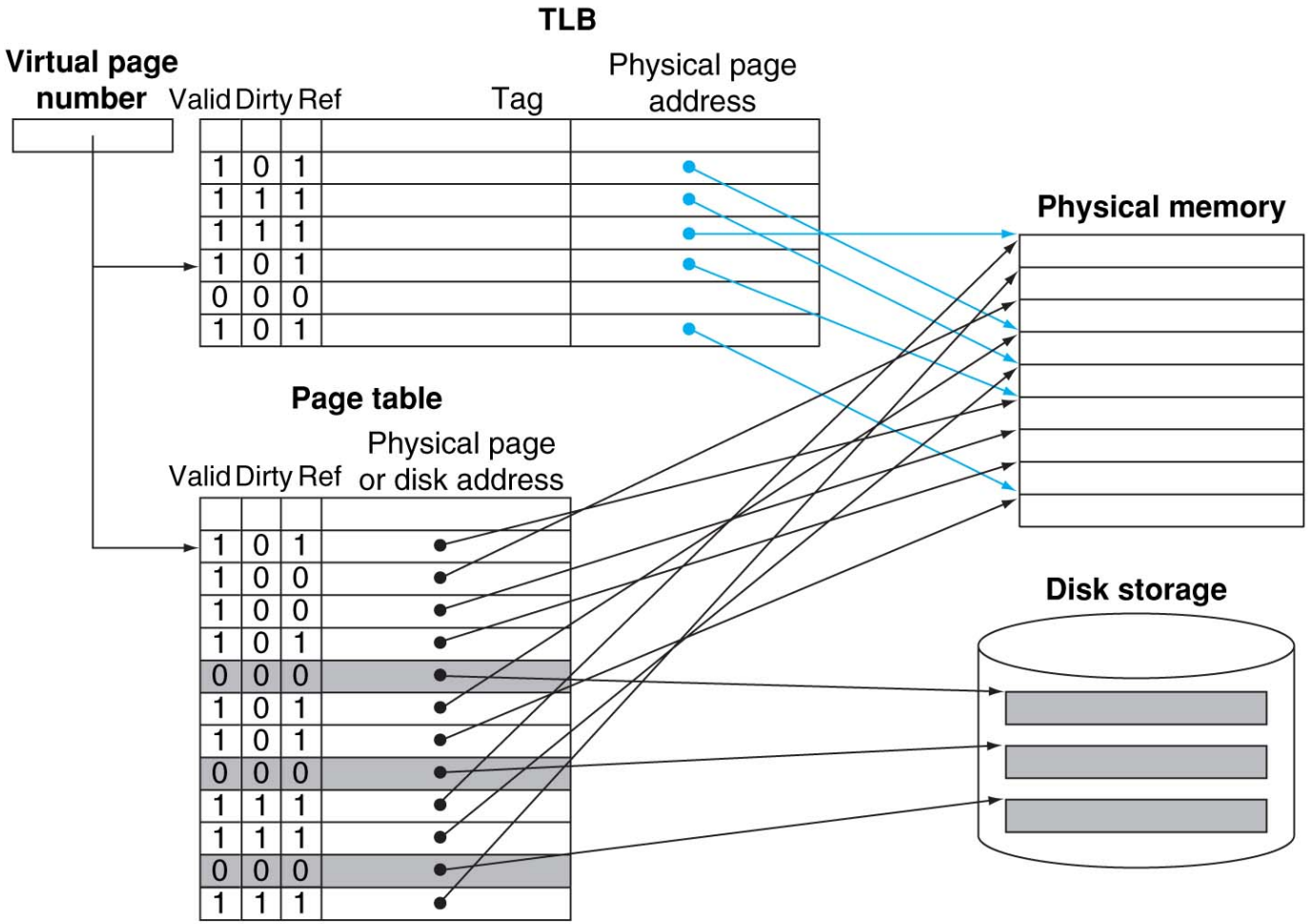
- Page table cache (yes, caches again ☺)
- In MMU of every modern CPU

When is address translation done?

- Before going to L1 cache

$\Rightarrow$ Caches use physical addresses (usually)

TLB principle



TLB metrics

Metrics – typical

- Size: typical 16-512 entries
- Block size: 1-2 page table entries (4-8 bytes each)
- Hit time: 0.5-1 cycle
- Miss penalty: 10-100 cycles (page hit)
- Miss rate: 0.01%-1%
- Levels: 1-2
- Split: data and instruction split on first level or both

TLB metrics

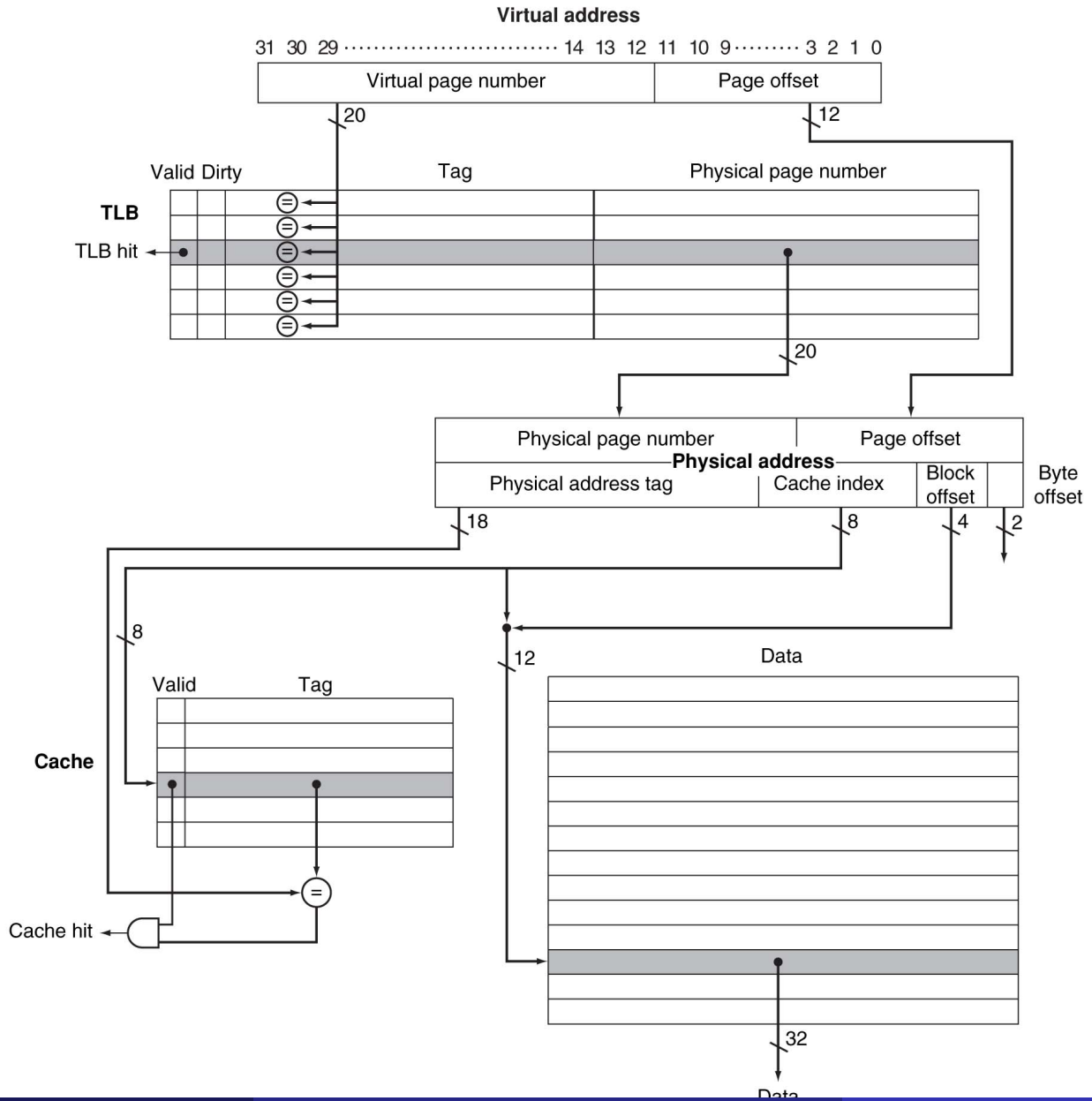
Metrics – typical

- Size: typical 16-512 entries
- Block size: 1-2 page table entries (4-8 bytes each)
- Hit time: 0.5-1 cycle
- Miss penalty: 10-100 cycles (page hit)
- Miss rate: 0.01%-1%
- Levels: 1-2
- Split: data and instruction split on first level or both

Associativity

- TLB's are small
- Need to be fast
- Fully associative OK for small caches
  - But (proper) LRU too expensive
- Rarely low associativity
  - Miss expensive $\Rightarrow$ aim for low miss rate

TLB and cache



TLB examples

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
Virtual address	48 bits	48 bits
Physical address	44 bits	48 bits
Page size	4 KB, 2/4 MB	4 KB, 2/4 MB
TLB organization	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement The L2 TLB is four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB has 512 entries TLB misses handled in hardware	1 L1 TLB for instructions and 1 L1 TLB for data per core Both L1 TLBs fully associative, LRU replacement 1 L2 TLB for instructions and 1 L2 TLB for data per core Both L2 TLBs are four-way set associative, round-robin Both L1 TLBs have 48 entries Both L2 TLBs have 512 entries TLB misses handled in hardware

TLB examples 2015-2017

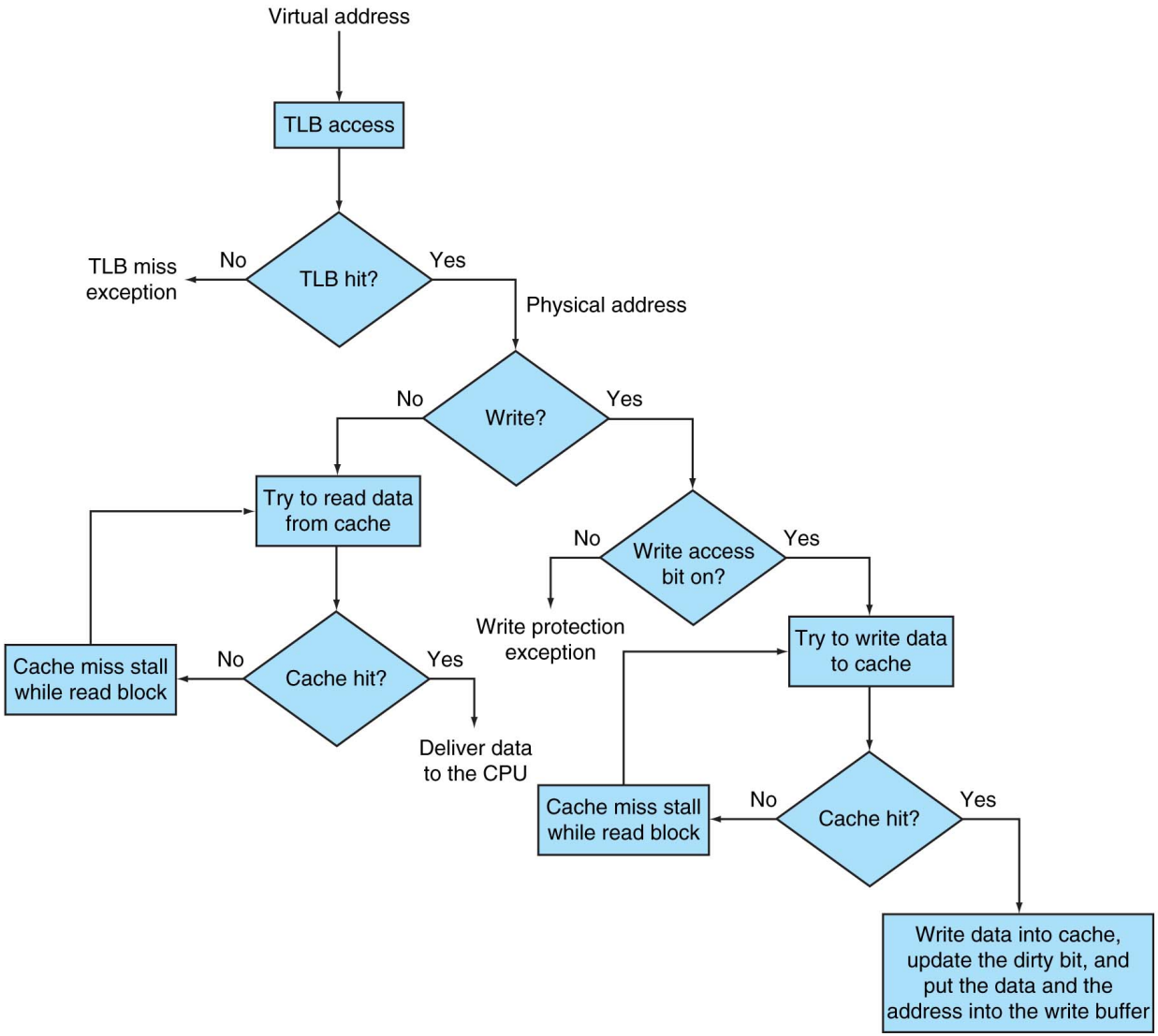
	Intel Skylake (architecture) (*2015*/16)	AMD Zen (architecture) (*2017*)
Virtual address	48 bits (?)	?
Pyhsical address	- (depends on model)	- (depends on model)
Page size	4 KB, 2/4 MB, 1 GB	4 KB, 2/4 MB, 1 GB
TLB organization		
TLB L1 instruction	128 (4 KB pages, 8-way), 8 per thread (2/4 MB)	**L0:8**(all sizes); 64 (all sizes)
TLB L1 data	64 (4 KB pages), 32 (2/4 MB), 4 (1 GB); 4-way	64 (all sizes)
TLB L2	1536 (4 KB, 2/4 MB; 12-way), 16 (1G; 4-way)	**I**:512 (no 1G); **D**: 1536 (no 1G): 6-way

TLB misses

TLB miss

- Not the same as page fault!
- Page hit
  - Can be handled by hardware or software
- Page fault
  - Handled by software
- Software is invoked via exceptions

Memory access procedure



Performance impact

- Working set > TLB covered address space
  - High TLB miss rate

Performance impact

- Working set > TLB covered address space
  - High TLB miss rate

Possible solution

- Larger pages
  - E.g. 2/4 MByte pages on x86

Performance impact

- Working set > TLB covered address space
  - High TLB miss rate

Possible solution

- Larger pages
  - E.g. 2/4 MByte pages on x86
- But, access of one byte pulls in 4 MB !!
  - Very bad for random access of large working set

Performance impact

- Working set > TLB covered address space
  - High TLB miss rate

Possible solution

- Larger pages
  - E.g. 2/4 MByte pages on x86
- But, access of one byte pulls in 4 MB !!
  - Very bad for random access of large working set

Example:

- Radix Sort has more TLB misses than Quicksort
  - Slower despite the theoretical advantage

TLB performance example

Exercise

Determine TLB miss rate for code that goes one by one through a large array of `double` (8 bytes).

- Page size: 4KByte

TLB performance example

Exercise

Determine TLB miss rate for code that goes one by one through a large array of `double` (8 bytes).

- Page size: 4KByte

Solution

- 4KByte page $\Rightarrow$ 512 array elements per page
- Access to first element: TLB miss
- Next 511 elements: TLB hit
- TLB miss rate $= \frac{1}{512} = 0.195\%$

Content

Role of kernel/OS

Only kernel/OS

- Can write page tables
  - Sets the mapping between virtual and physical memory
- Sets page table register
  - Changed during context switch
- Can only be done in supervisor/kernel mode
  - Enter with system call or exception

Role of kernel/OS

Only kernel/OS

- Can write page tables
  - Sets the mapping between virtual and physical memory
- Sets page table register
  - Changed during context switch
- Can only be done in supervisor/kernel mode
  - Enter with system call or exception
- Context switch
  - Change of execution program (process)
  - TLBs and caches are cleared
    - *Can* make switching expensive
- Part of memory space must always be in physical memory
  - Exception handler
  - OS page table

Page fault handler

- Part of kernel/OS
- Entered through exception

Page fault handler

- Part of kernel/OS
- Entered through exception

1. Find usable physical page
   - LRU algorithm
2. Write it back to disc (swap file/partition) (if dirty bit set)

Page fault handler

- Part of kernel/OS
- Entered through exception

1. Find usable physical page
   - LRU algorithm
2. Write it back to disc (swap file/partition) (if dirty bit set)
3. Read requested page from disc

Page fault handler

- Part of kernel/OS
- Entered through exception

1. Find usable physical page
   - LRU algorithm
2. Write it back to disc (swap file/partition) (if dirty bit set)
3. Read requested page from disc
4. Adjust page table entries

Page fault handler

- Part of kernel/OS
- Entered through exception

1. Find usable physical page
   - LRU algorithm
2. Write it back to disc (swap file/partition) (if dirty bit set)
3. Read requested page from disc
4. Adjust page table entries
5. Memory access retried
   - Cache miss ...

Protection

Virtual memory is used for protection

- Each program (process) has own page table
  - $\Rightarrow$ Cannot read memory of other process
    - Simply not in their page table
    - Not allowed to change page table
- Sharing of memory must be explicitly enabled by OS
  - Different virtual addressed pointing to the same physical address

Protection

Virtual memory is used for protection

- Each program (process) has own page table
  - $\Rightarrow$ Cannot read memory of other process
    - Simply not in their page table
    - Not allowed to change page table
- Sharing of memory must be explicitly enabled by OS
  - Different virtual addressed pointing to the same physical address
- Protection bit (read only)
  - Exception when written to
  - Ideal to share libraries
- Execution bit
  - Do not allow execution of data areas $\Rightarrow$ security
- Exception handlers are very flexible and versatile
  - Can be used to implement transparent memory sharing over network!