Ve370 Final Big RC

Chapter 13

Main difference between interrupt and exception: Cause (external/internal)

Concept of parallelism and asynchronous.

Interrupt (Trap)

- Basic Part: an “unplanned” function call to a “third-party” routine; and later return control back to point of interruption
- An event that needs to be processed by another (system) program. The event is usually unexpected or rare from program’s point of view.

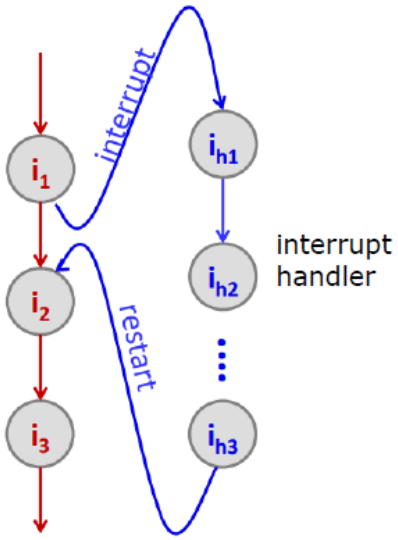


Image: http://users.ece.cmu.edu/~jhoe/course/ece447/

Exceptions

- Caused by the execution of an instruction
- The instruction cannot be completed
  - undefined opcode, privileged instructions
  - arithmetic overflow, FPU exception
  - misaligned memory access
  - virtual memory exceptions: page faults, TLB misses, protection violations

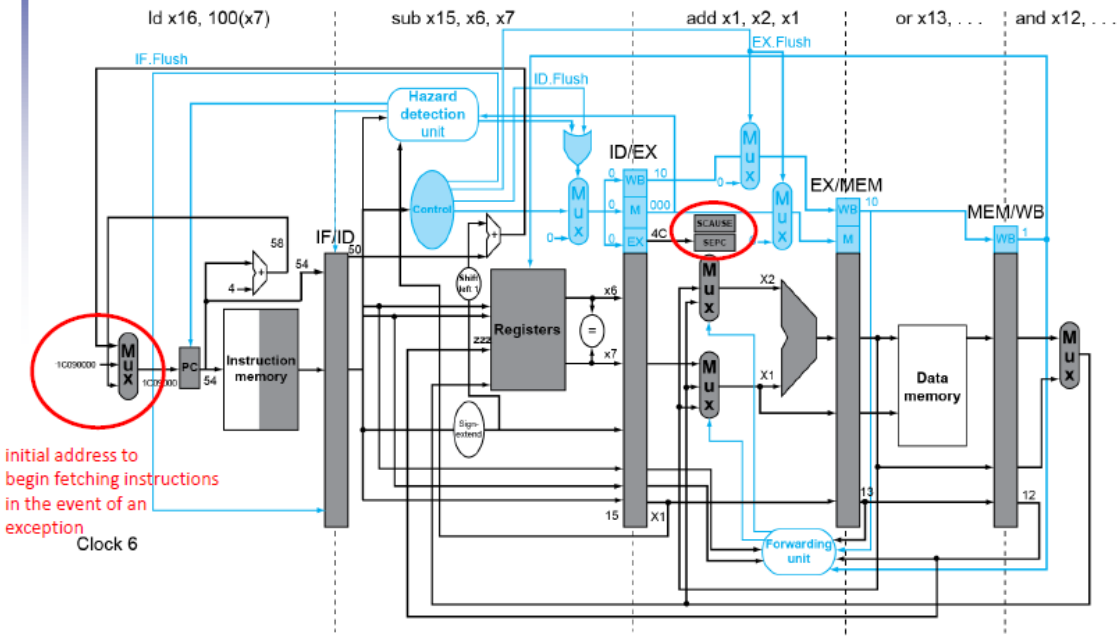
Handling Exceptions

- Save PC of offending (or interrupted) instruction
  - In RISC-V: Supervisor Exception Program Counter (**SEPC**)
- Save indication of the problem
  - In RISC-V: Supervisor Exception Cause Register (**SCAUSE**)
  - 64 bits, but most bits unused
    - Exception code field: 2 for undefined opcode, 12 for hardware malfunction, …
- Jump to handler
  - Assume at $0000\ 0000\ 1C09\ 0000_{hex}$

An Alternate Mechanism

- **Vectored Interrupts**
  - Handler address determined by the cause
- Exception vector address to be added to a vector table base register:
  - Undefined opcode 00 0100 0000$_{two}$
  - Hardware malfunction: 01 1000 0000$_{two}$
  - Entry points separated by limited number of instructions
    - 32 bytes in above example
  - …: …
- Instructions either
  - Deal with the interrupt, or
  - Jump to real handler

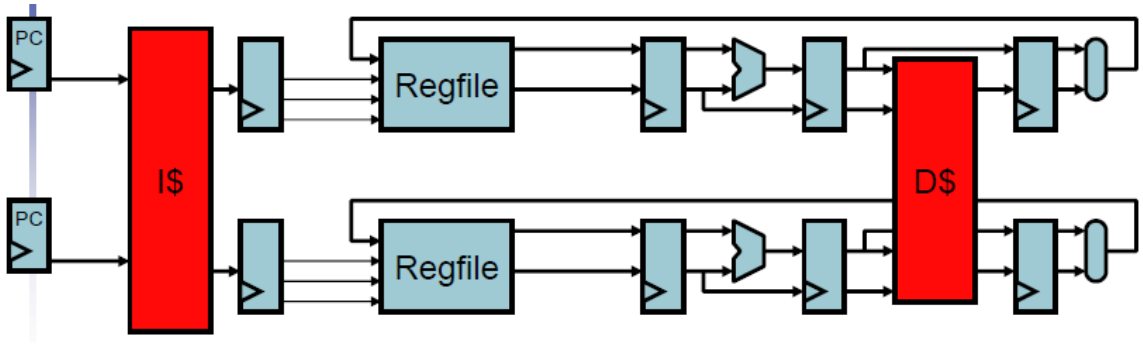
Exception Example



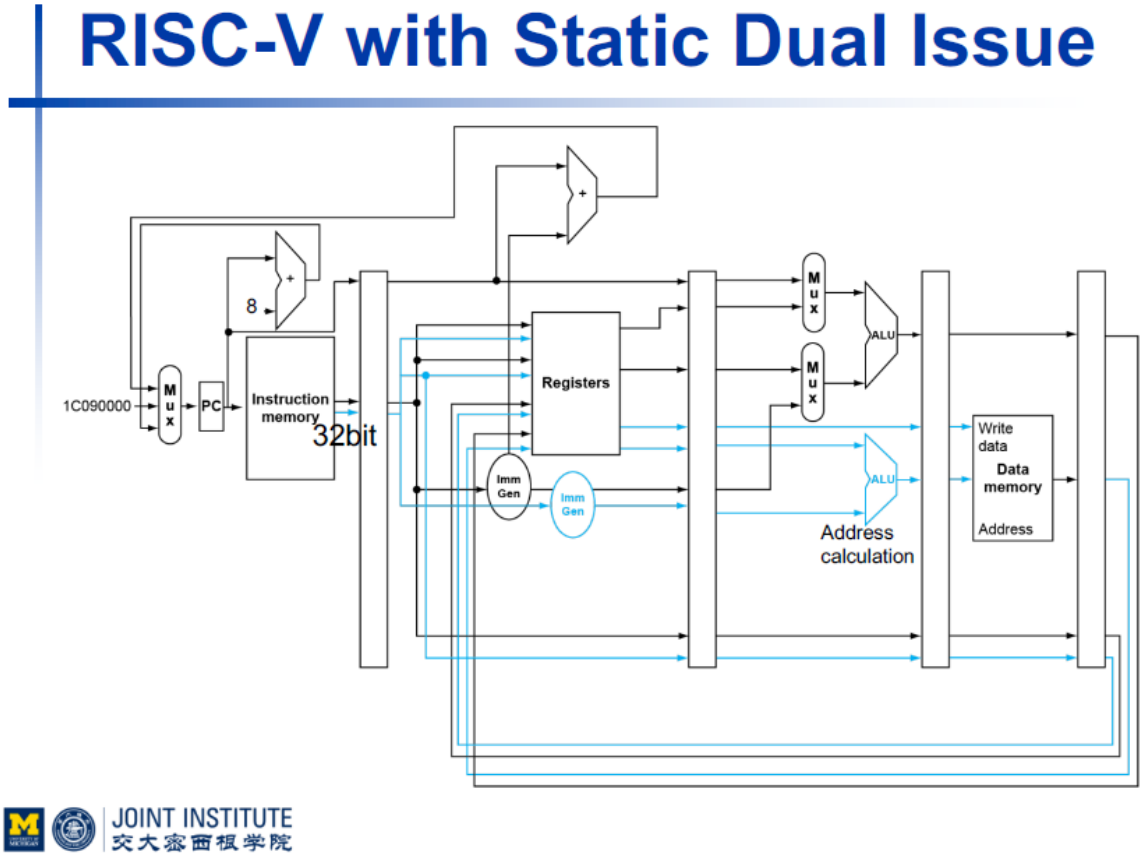
Multiple exceptions/interrupts at the same time: Priority.

High prioritized interrupts can interrupt low prioritized interrupt.

Chapter 14



Flynn Bottleneck: CPI = 1



"Hazards" caused by parallelism:

1. Data dependency (load use, type 1, type 2...)
2. Scramble for resource usage (memory, Reg file...)
3. Branch & Jump (maintain the overall instruction order), use out-of-order to maintain the order

Benefits:

Performance

Threats:

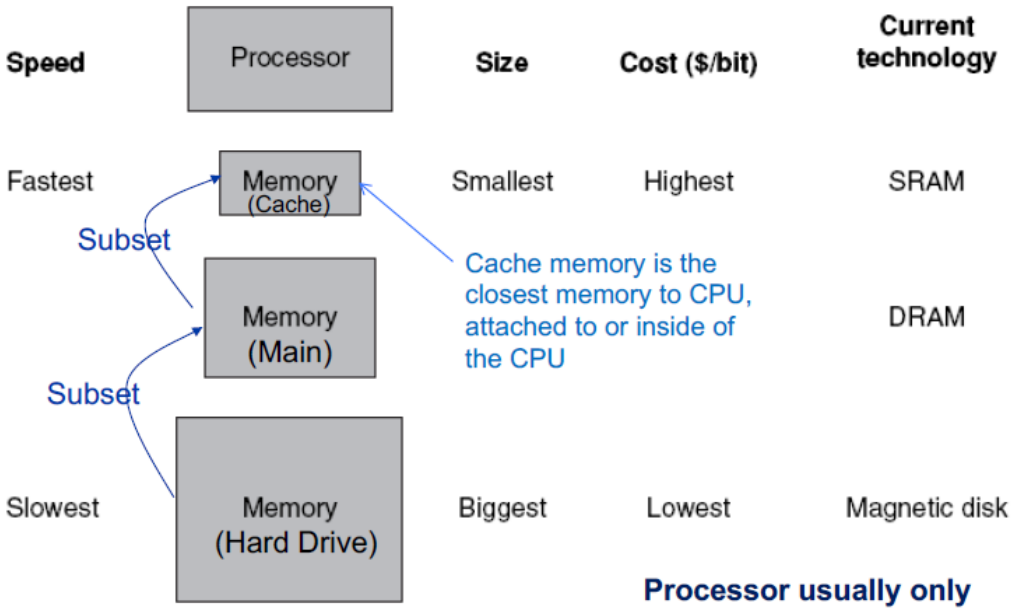
Power consumption

Chapter 15-18

What is Cache: Another memory

Why need cache: Performance

Memory Hierarchy



Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 7

Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

JOINT INSTITUTE 交大密西根学院

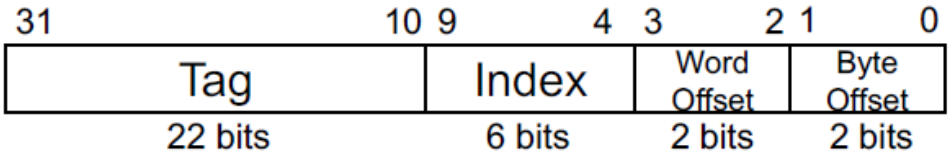
Terminologys:

1. Block: unit of copying
2. Word address/Byte address
3. Word offset/Byte offset (Word address & Byte offset = Byte address)
4. Hit/Miss ratio
5. Hit time/Miss penalty

Direct Mapped Cache

- Direct mapped cache: each memory location corresponds to **one** choice in cache
- How to map?
- Location of a block in cache is determined by address of the requested word
  - Given word address, we can get block address (block number)
  - *Cache location (or cache block index) =*
    (Block address in memory) % (Number of blocks in cache) =
    lower $\log_2$(Number of blocks in cache) bits of block address
- **Multiple** memory locations correspond to **one** block in cache – n-to-1 mapping

Tag + Cache index = block address



Cache Size (Total # bits)

- Assume
  - 64-bit addresses
  - A direct-mapped cache
  - The cache size is 2^n blocks, so n bits are used for the index
  - The block size is 2^m words (2^{m+2} bytes), so m bits are used for the word within the block, and two bits are used for the byte part of the address
- The size of the tag field is

  $64 - (n + m + 2)$
- The total number of bits in a direct-mapped cache is

  2^n x (block size + tag size + value field size)=

  $$2^n \times (2^m \times 32 + (64 - n - m - 2) + 1) = 2^n \times (2^m \times 32 + 63 - n - m)$$

Usually the cache naming only indicates the size of the data!

JOINT INSTITUTE 交大密西根学院

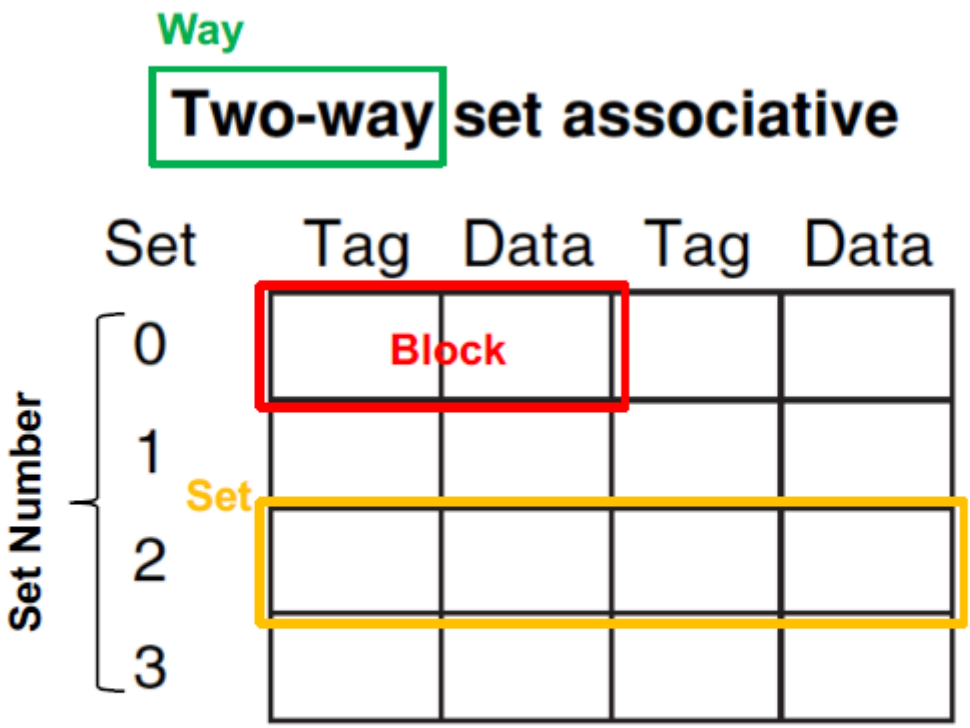
How to write for memory consistency:

1. Write through: when a write is needed, just write to memory. Pros: Easy to implement, less problems; Cons: worse performance
2. Write back: write to memory when the block is replaced. Use a dirty bit to indicate if replacement needs to write to memory. Pros: better performance; Cons: Complexity

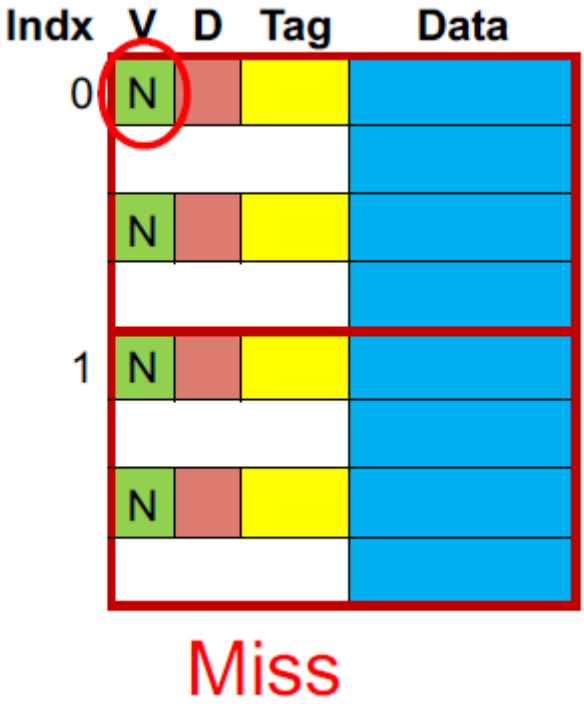
Associativity:

Improve Performance – Associative Caches

- *n*-way **set associative** cache
  - Each set contains ***n* blocks**
  - Each address maps to a unique set
    - Set index = (Block address) % (number of sets in cache)
  - A mem block maps to any block within the corresponding set
  - However, to locate a block in a set, need to compare *n* times
    - all *n* tags in a set must be checked and compared
    - *n* comparators (more effective - faster)
- Fully associative – opposite extreme of direct mapped
  - Allow a given block to go in any cache entry
  - Must search all entries to find a hit
  - One comparator each block (expensive)
    - # of comparator = cache size (block number)



Each block can have multiple words.



- Size of index field
  - Increasing degree of associativity decreases the number of sets, decreases number of bits for index, increases tag field
    - Doubling # of blocks by 2 halves # of set by 2
    - Reduce index bits by 1
    - Increase tag bits by 1

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns (L-1 miss penalty)
  - Miss rate for L-2 = 25% of L1 misses (have to access main memory)
    - L-1 cache miss have a miss on L-2
- Primary (L-1) cache miss with L-2 hit
  - Miss penalty = 5ns/0.25ns = 20 cycles
- Primary cache miss with L-2 miss main memory hit
  - Extra penalty = 400 cycles
- CPI = base CPI + L-1 miss L-2 hit (cycles per instruction) + L-1 miss L-2 miss (cycles per instruction)
  - CPI = 1 + 0.02 × 75% × 20 + 0.02 × 25% × (20+400) = 3.4
- Performance ratio = 9/3.4 = 2.6

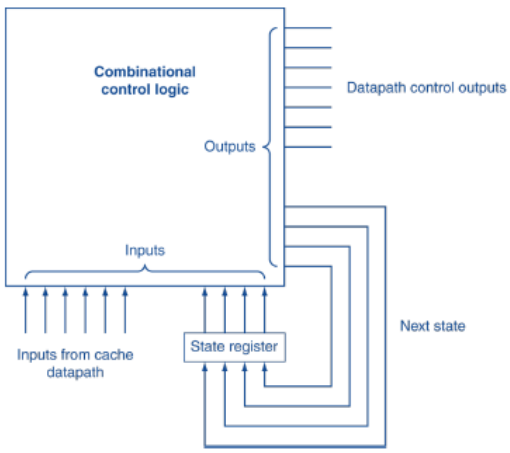
3 Cs Model of Misses

Miss Rate: ABC

- Associativity
  - Decrease conflict misses
  - Increase hit latency
- Block size
  - Increase conflict misses (fewer entries)
  - Decrease capacity misses (spatial locality)
  - Decrease Compulsory (cold) misses
- Capacity
  - Decrease capacity misses
  - Increase hit latency

Finite State Machines

- Use an FSM for sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state = f_n (current state, current inputs)
- Control output signals = f_o (current state)



Hamming Code:

[说人话，人话！！ 汉明码（海明码、hamming code）通俗易懂的解释，说人话！！！！ Yonggie的博客-CSDN博客汉明码]

Cache Coherence

- Basic question: If multiple processors cache the same block, how do they ensure they all see a consistent state?

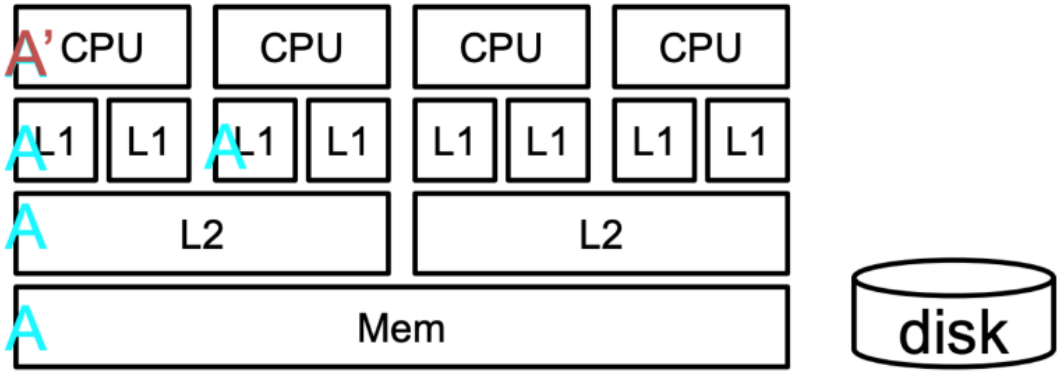
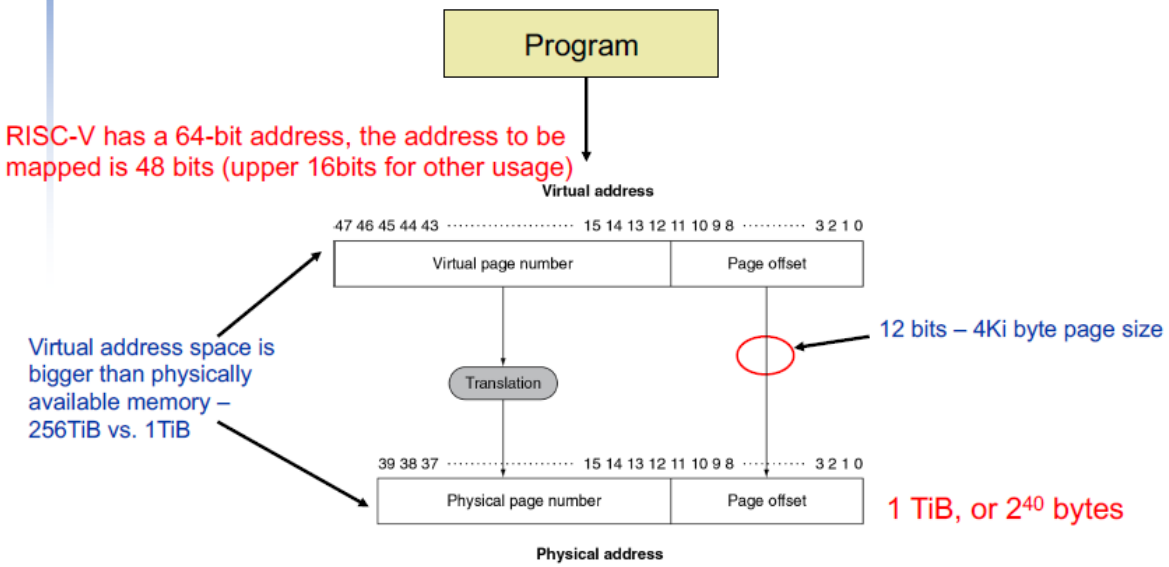


Image: cs.cornell.edu/courses/cs3410

Chapter 19

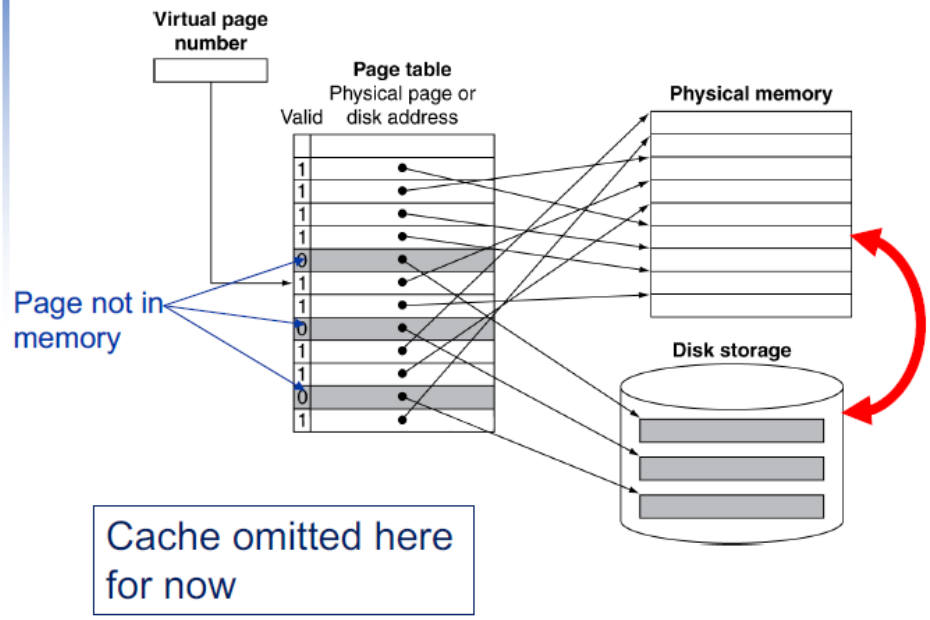
Address Translation

- Assuming fixed-size pages (e.g., 4K Bytes)

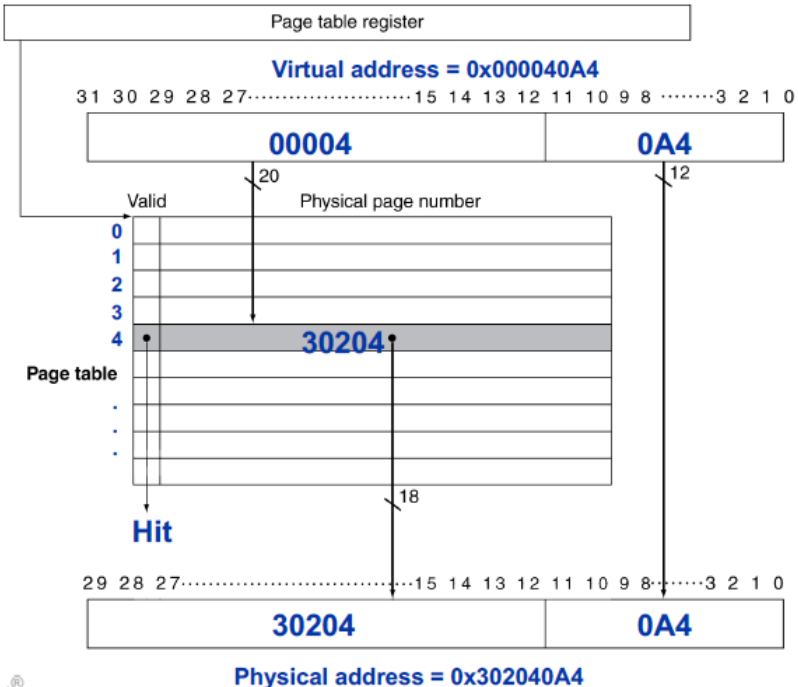


Address Mapping

Mapping Pages to Storage



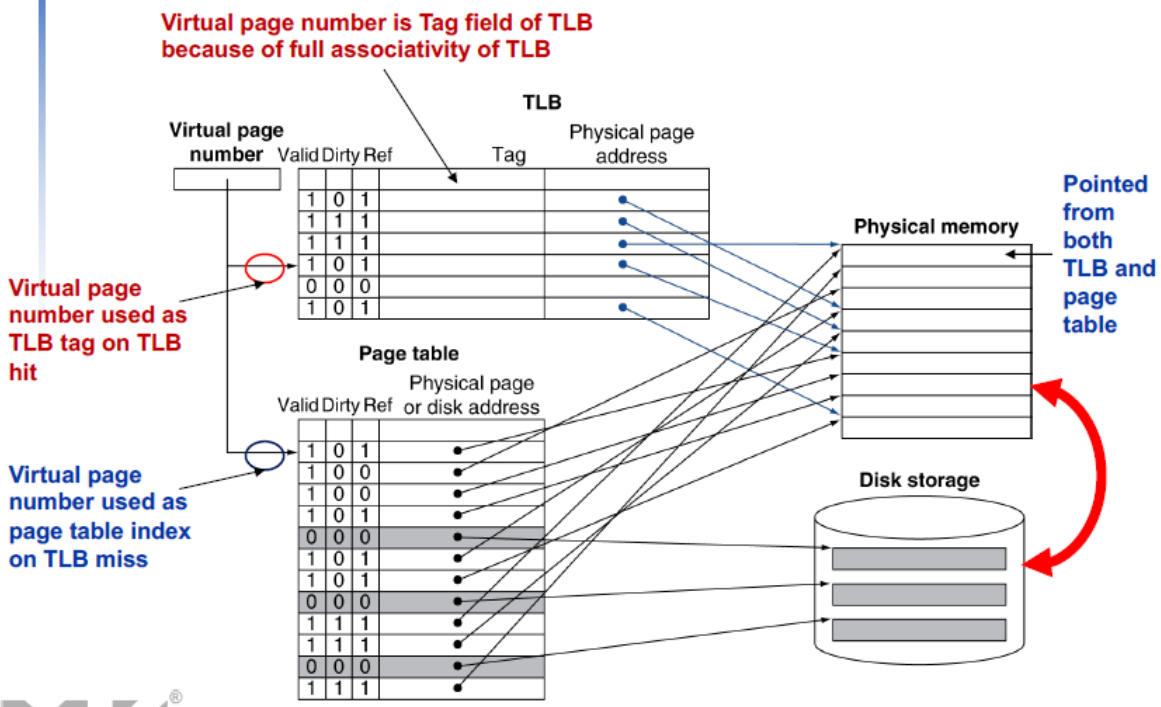
Example



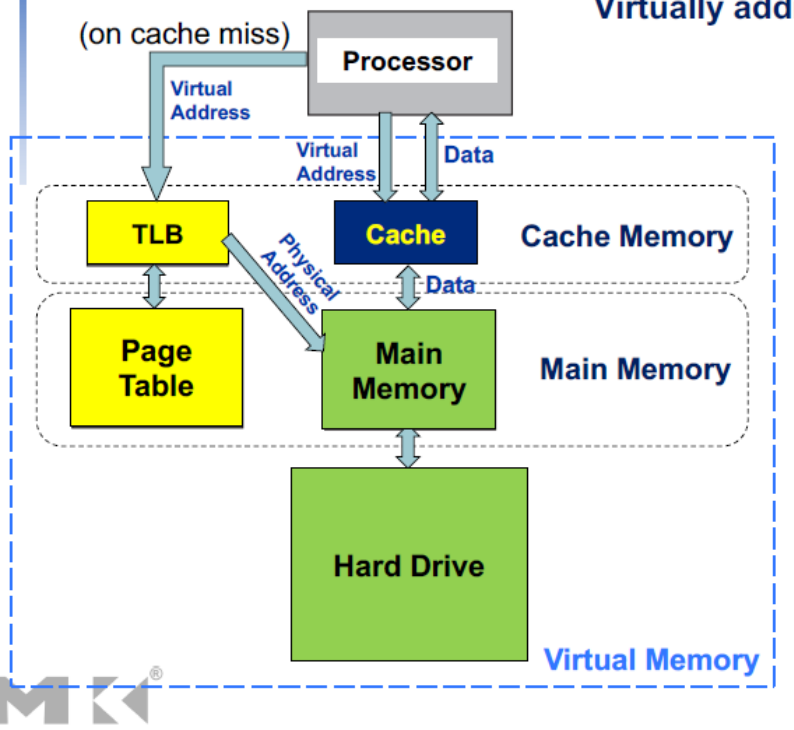
Page Table Size

- Example:
  - Page size: 4KB
  - 48-bit virtual byte address (256 TiB)
  - 5 bytes per page table entry
- Number of page table entries = $2^{(48-12)} = 2^{36}$
  - Up to, might be customized for each program and usually less than that
- Size of page table = number of page table entries x bytes/page table entry
  - Page table size = 2^{36} x 5!

Fast Translation Using a TLB

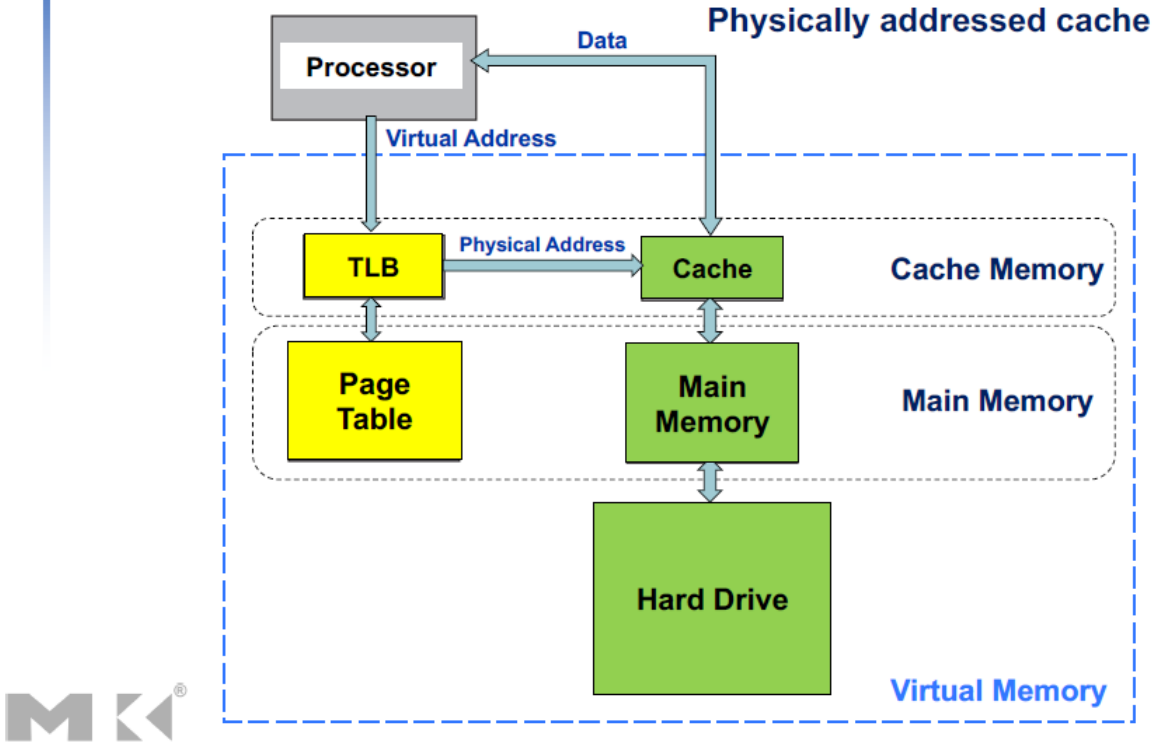


TLB vs. Cache



- No need for translation
- Translate when cache miss
- Complications due to aliasing
  - When different virtual spaces map to the same physical main memory location

TLB vs. Cache



Concepts in Chapter 20-23

1. Storage Types
2. RAM,DIMM, ROM, Flash
3. Volatility
4. Accessibility
5. Mutability
6. Addressability
7. I/O
8. OS
9. Bus
10. MMIO (memory mapped I/O)
11. Polling
12. Interrupt
13. DMA
14. Amdahl's Law
15. Flynn's Classification
16. Thread & SMT
17. GPU v.s CPU