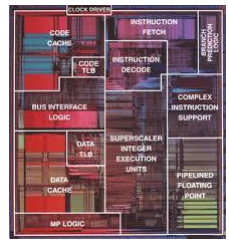
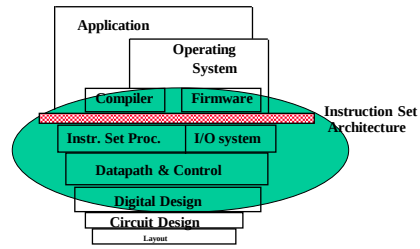
CS/SE 3340
Computer Architecture



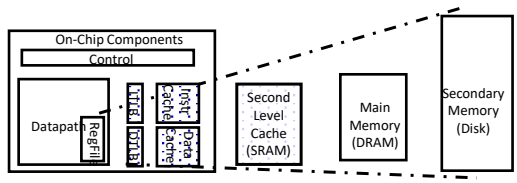


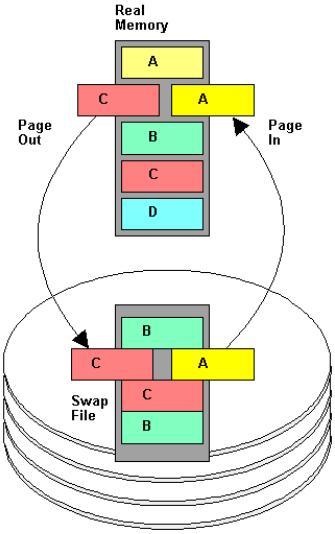
Memory Hierarchy – Virtual Memory

Adapted from slides by Profs. D. Patterson and J. Hennessey

Questions

1. What are the three sources of cache misses?
2. What/Why of virtual memory?
3. What/How of address translation?
4. How to map pages to storage?
5. What/Why/How of TLB/fast translation?





Least Recently Used (LRU) Algorithm

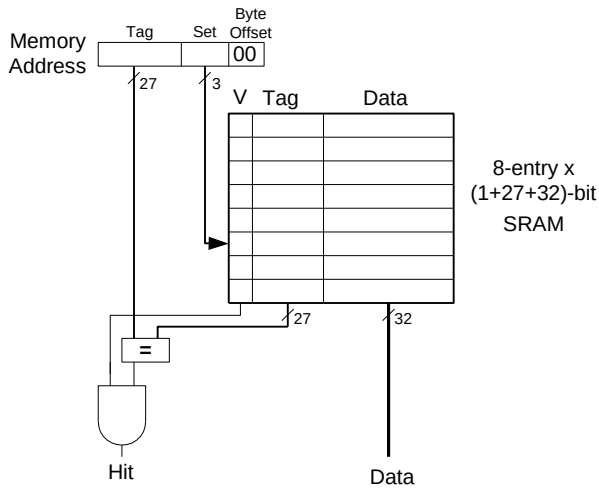
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

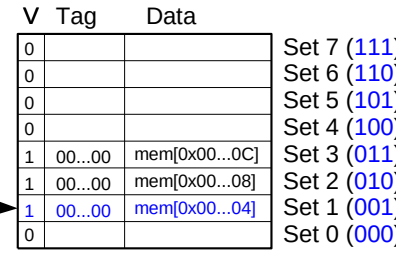
3

Recap: Direct-Mapped



4

Recap: Direct-Mapped



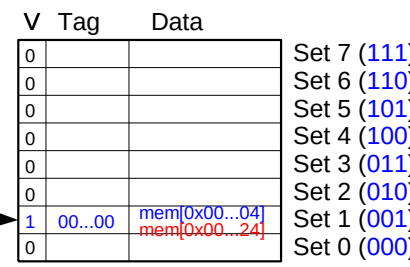

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 3/15

= 20%

Compulsory Misses

Recap: Direct-Mapped



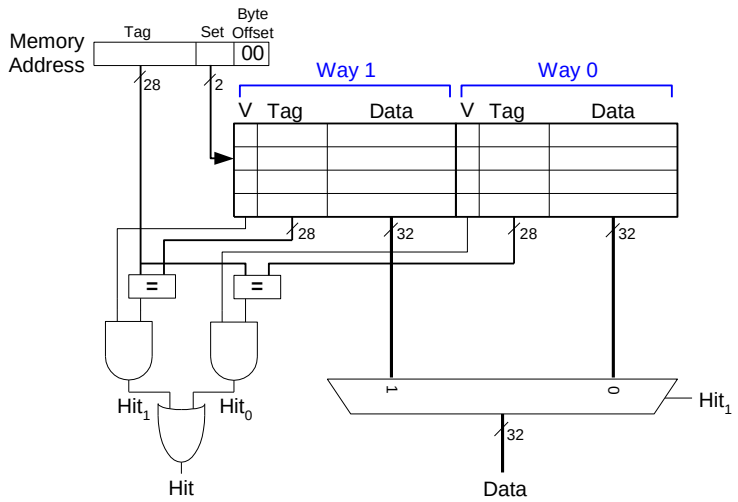

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 10/10

= 100%

Conflict Misses

Recap: N-way Set Associative



Recap: N-way Set Associative

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 2/10
= 20%

Associativity reduces conflict misses

	Way 1			Way 0		
	V	Tag	Data	V	Tag	Data
Set 3	0			0		
Set 2	0			0		
Set 1	1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]
Set 0	0			0		

Sources of Cache Misses

- *Compulsory misses (aka cold start misses)*
  - First access to a cache block
- *Conflict misses (aka collision misses)*
  - Due to competition for entries in a set
  - In a non-fully associative cache only, would not occur in a fully associative cache of the same total size
- *Capacity misses*
  - Due to finite cache capacity
  - A replaced cache block is later accessed again

9

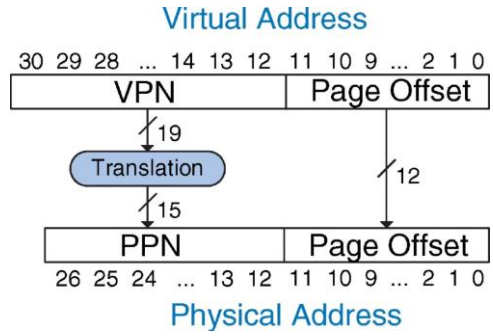
Virtual Memory

- Use *main memory* as a "*cache*" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a *private* virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate *virtual addresses* to *physical addresses*
  - VM "block" is called a **page**
  - VM translation "miss" is called a ***page fault***

10

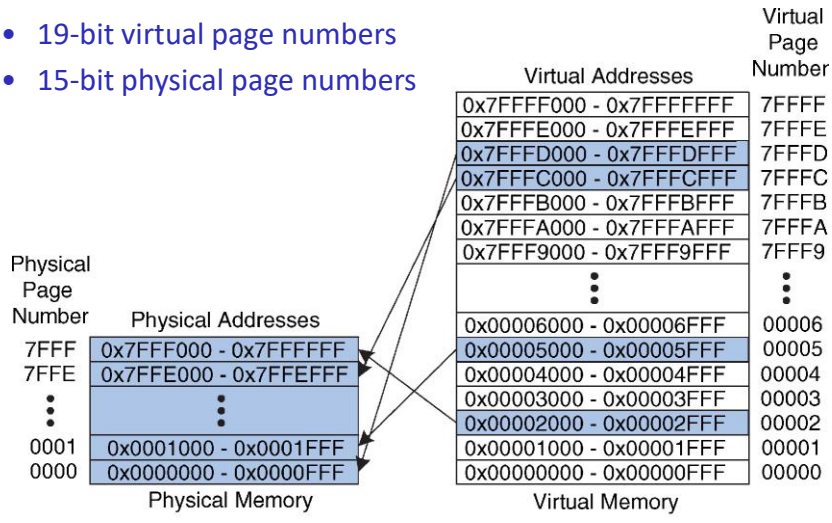
Virtual Memory Example

- Virtual memory size: 2 GB = 2^{31} bytes
- Physical memory size: 128 MB = 2^{27} bytes
- Page size: 4 KB = 2^{12} bytes

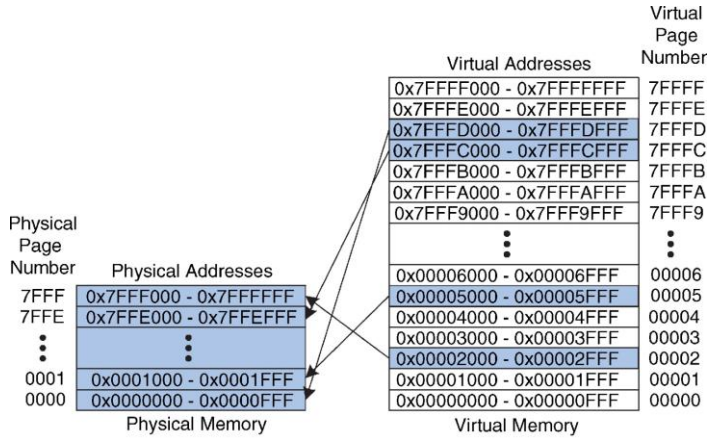


Virtual Memory Example

- 19-bit virtual page numbers
- 15-bit physical page numbers



Address Translation Example

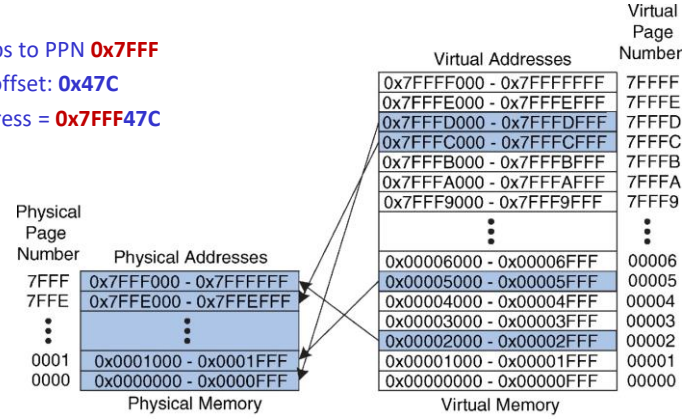


What is the physical address of virtual address **0x247C**?

Address Translation Example

What is the physical address of virtual address **0x247C?**

- VPN = **0x2**
- VPN 0x2 maps to PPN **0x7FFF**
- 12-bit page offset: **0x47C**
- Physical address = **0x7FFF47C**

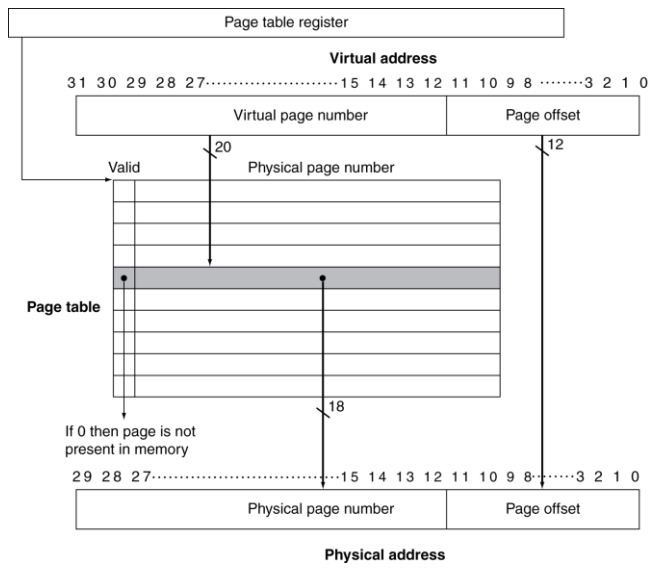


Page Tables

- Stores placement information
  - Array of *page table entries (PTE),* indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

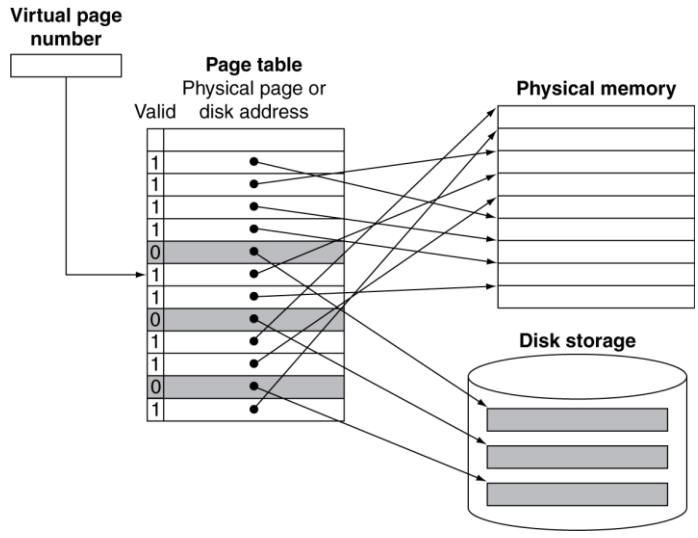
15

Translation Using a Page Table



16

Mapping Pages to Storage



17

Page Fault Penalty

- On *page fault*, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to *minimize page fault rate*
  - Fully associative placement
  - Smart replacement algorithms

18

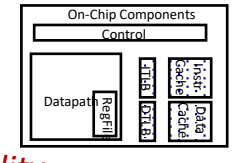
Replacement and Writes

- To reduce page fault rate, prefer *least-recently used (LRU)* replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

19

Fast Translation Using a TLB

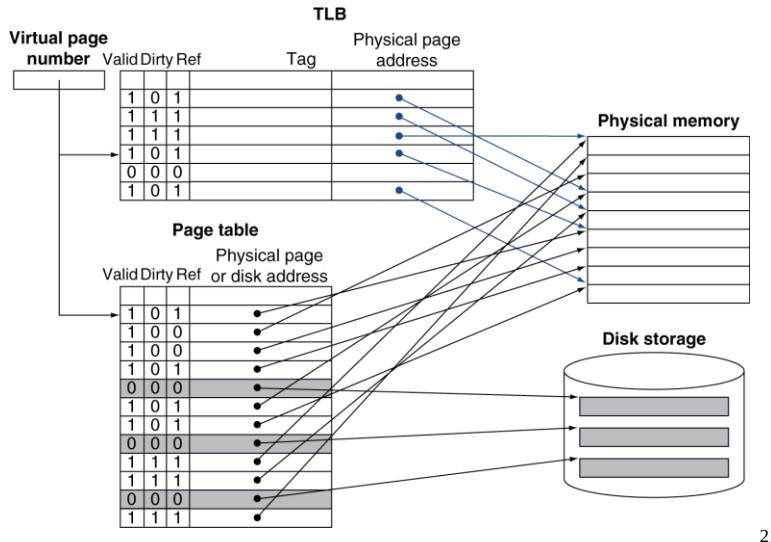
- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access



- But access to page tables has good *locality*
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

20

Fast Translation Using a TLB



21

The Memory Hierarchy

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of ***caching***
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

22

Block Placement

- Determined by set associativity
  - *direct mapped*
    - 1-way associative (# of sets = # of cache blocks)
    - 1 block associated with 1 set
  - *n-way* set associative
    - n choices within a set
    - # of sets = # of cache blocks / n
  - *fully* associative
    - All cache blocks are associated with one set
- Higher set associativity reduces miss rate
  - Increases complexity, cost, and access time

23

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

24

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

25

Write Policy

- *Write-through*
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- *Write-back*
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

26

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

27

Summary

- Virtual memory increases address space of a computer system by using secondary storage (e.g. hard disk)
  - Main memory used as 'cache' for secondary storage!
  - Page-in and page-out
  - Page fault
- Page fault causes performance hit
  - Data needs to be read from slow secondary storage to main memory
  - Pages in main memory are replaced
- Page replacement strategies
  - LRU – Least Recently Used
  - Random

28