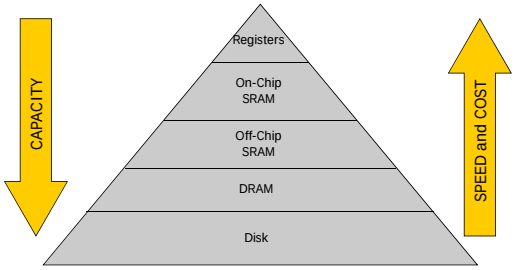
ECE/CS 552: Memory Hierarchy

Instructor: Mikko H Lipasti

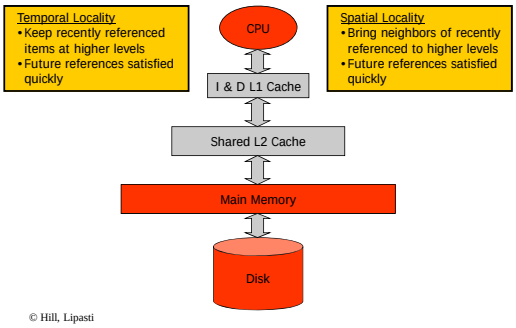
Fall 2010
University of Wisconsin-Madison

Lecture notes based on notes by Mark Hill
Updated by Mikko Lipasti

Memory Hierarchy



Memory Hierarchy



Four Burning Questions

- These are:
  - Placement
    - Where can a block of memory go?
  - Identification
    - How do I find a block of memory?
  - Replacement
    - How do I make space for new blocks?
  - Write Policy
    - How do I propagate changes?
- Consider these for registers and main memory
  - Main memory usually DRAM

Placement

Memory Type	Placement	Comments
Registers	Anywhere; Int, FP, SPR	Compiler/programmer manages
Cache (SRAM)	Fixed in H/W	*Direct-mapped*, *set-associative*, *fully-associative*
DRAM	Anywhere	O/S manages
Disk	Anywhere	O/S manages

Register File

- Registers managed by programmer/compiler
  - Assign variables, temporaries to registers
  - Limited name space matches available storage
  - Learn more in CS536, CS701

Placement	Flexible (subject to data type)
Identification	Implicit (name == location)
Replacement	Spill code (store to stack frame)
Write policy	Write-back (store on replacement)

Main Memory and Virtual Memory

- Use of virtual memory
  - Main memory becomes another level in the memory hierarchy
  - Enables programs with address space or working set that exceed physically available memory
    - No need for programmer to manage overlays, etc.
    - Sparse use of large address space is OK
  - Allows multiple users or programs to timeshare limited amount of physical memory space and address space
- Bottom line: efficient use of expensive resource, and ease of programming

Virtual Memory

- Enables
  - Use more memory than system has
  - Program can think it is the only one running
    - Don't have to manage address space usage across programs
    - E.g. think it always starts at address 0x0
  - Memory protection
    - Each program has private VA space: no-one else can clobber
  - Better performance
    - Start running a large program before all of it has been loaded from disk

Virtual Memory – Placement

- Main memory managed in larger blocks
  - *Page size* typically 4K – 16K
- Fully flexible placement; fully associative
  - Operating system manages placement
  - Indirection through *page table*
  - Maintain mapping between:
    - Virtual address (seen by programmer)
    - Physical address (seen by main memory)

Virtual Memory – Placement

- Fully associative implies expensive lookup?
  - In caches, yes: check multiple tags in parallel
- In virtual memory, expensive lookup is avoided by using a level of indirection
  - Lookup table or hash table
  - Called a *page table*

Virtual Memory – Identification

Virtual Address	Physical Address	Dirty bit
0x20004000	0x2000	Y/N

- Similar to cache tag array
  - Page table entry contains VA, PA, dirty bit
- Virtual address:
  - Matches programmer view; based on register values
  - Can be the same for multiple programs sharing same system, without conflicts
- Physical address:
  - Invisible to programmer, managed by O/S
  - Created/deleted on demand basis, can change

Virtual Memory – Replacement

- Similar to caches:
  - FIFO
  - LRU; overhead too high
    - Approximated with reference bit checks
    - Clock algorithm
  - Random
- O/S decides, manages
  - CS537

Virtual Memory – Write Policy

- Write back
  - Disks are too slow to write through
- Page table maintains dirty bit
  - Hardware must set dirty bit on first write
  - O/S checks dirty bit on eviction
  - Dirty pages written to backing store
    - Disk write, 10+ ms

Virtual Memory Implementation

- Caches have fixed policies, hardware FSM for control, pipeline stall
- VM has very different miss penalties
  - Remember disks are 10+ ms!
- Hence engineered differently

Page Faults

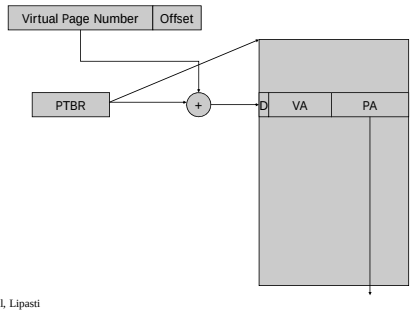
- A virtual memory miss is a page fault
  - Physical memory location does not exist
  - Exception is raised, save PC
  - Invoke OS page fault handler
    - Find a physical page (possibly evict)
    - Initiate fetch from disk
  - Switch to other task that is ready to run
  - Interrupt when disk access complete
  - Restart original instruction
- Why use O/S and not hardware FSM?

Address Translation

VA	PA	Dirty	Ref	Protection
0x20004000	0x2000	Y/N	Y/N	Read/Write/ Execute

- O/S and hardware communicate via PTE
- How do we find a PTE?
  - &PTE = PTBR + page number * sizeof(PTE)
  - PTBR is private for each program
    - Context switch replaces PTBR contents

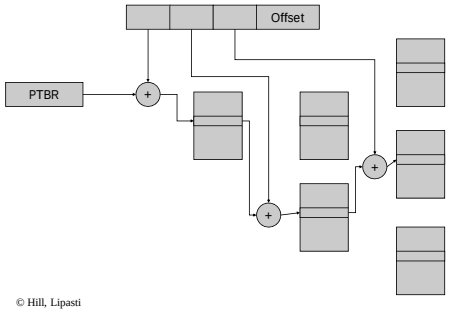
Address Translation



Page Table Size

- How big is page table?
  - 2^{32} / 4K * 4B = 4M per program (!)
  - Much worse for 64-bit machines
- To make it smaller
  - Use limit register(s)
    - If VA exceeds limit, invoke O/S to grow region
  - Use a multi-level page table
  - Make the page table pageable (use VM)

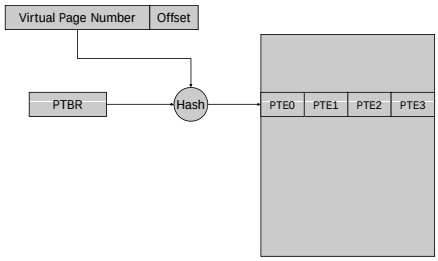
Multilevel Page Table



Hashed Page Table

- Use a hash table or inverted page table
  - PT contains an entry for each real address
    - Instead of entry for every virtual address
  - Entry is found by hashing VA
  - Oversize PT to reduce collisions: #PTE = 4 x (#phys. pages)

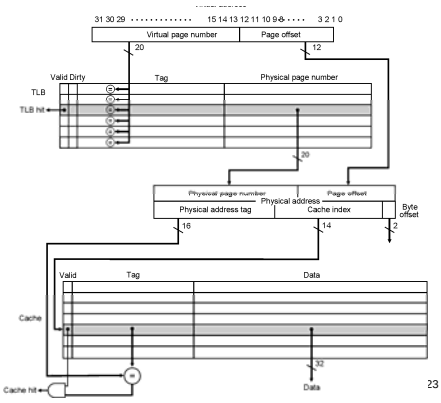
Hashed Page Table



High-Performance VM

- VA translation
  - Additional memory reference to PTE
  - Each instruction fetch/load/store now 2 memory references
    - Or more, with multilevel table or hash collisions
  - Even if PTE are cached, still slow
- Hence, use special-purpose cache for PTEs
  - Called TLB (translation lookaside buffer)
  - Caches PTE entries
  - Exploits temporal and spatial locality (just a cache)

TLB



Virtual Memory Protection

- Each process/program has private virtual address space
  - Automatically protected from rogue programs
- Sharing is possible, necessary, desirable
  - Avoid copying, staleness issues, etc.
- Sharing in a controlled manner
  - Grant specific permissions
    - Read
    - Write
    - Execute
    - Any combination
  - Store permissions in PTE and TLB

VM Sharing

- Share memory locations by:
  - Map shared physical location into both address spaces:
    - E.g. PA 0xC00DA becomes:
      - VA 0x2D000DA for process 0
      - VA 0x4D000DA for process 1
  - Either process can read/write shared location
- However, causes synonym problem

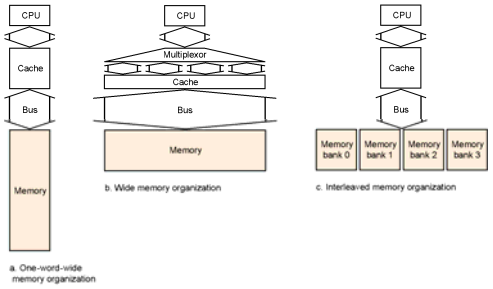
VA Synonyms

- Virtually-addressed caches are desirable
  - No need to translate VA to PA before cache lookup
  - Faster hit time, translate only on misses
- However, VA synonyms cause problems
  - Can end up with two copies of same physical line
- Solutions:
  - Flush caches/TLBs on context switch
  - Extend cache tags to include PID & prevent duplicates
    - Effectively a shared VA space (PID becomes part of address)

Main Memory Design

- Storage in commodity DRAM
- How do we map these to logical cache organization?
  - Block size
  - Bus width
  - Etc.

Main Memory Design



Main Memory Access

- Each memory access
  - 1 cycle address
  - 5 cycle DRAM (really >> 10)
  - 1 cycle data
  - 4 word cache block
- one word wide: (a=addr, d=delay, b=bus)
  - adddddbddddbddddbddddbddddb
  - 1 + 4 *(5+1) = 25 cycles

Main Memory Access

- Four word wide:
  - adddddb
  - 1 + 5 + 1 = 7 cycles
- Interleaved (pipelined)
  - adddddb
  - ddddd b
  - ddddd b
  - ddddd b
  - 1 + 5 + 4 = 10 cycles

Error Detection and Correction

- Main memory stores a huge number of bits
  - Probability of bit flip becomes nontrivial
  - Bit flips (called soft errors) caused by
    - Slight manufacturing defects
    - Gamma rays and alpha particles
    - Interference
    - Etc.
  - Getting worse with smaller feature sizes
- Reliable systems must be protected from soft errors via ECC (error correction codes)
  - Even PCs support ECC these days

Error Correcting Codes

- Probabilities:

  P(1 word no errors) > P(single error) > P(two errors) >> P(>2 errors)

- Detection - signal a problem
- Correction - restore data to correct value
- Most common
  - Parity - single error detection
  - SECDED - single error correction; double bit detection
- Supplemental reading on course web page!

1-bit ECC

Power	Correct	#bits	Comments
Nothing	0,1	1	
SED	00,11	2	01,10 detect errors
SEC	000,111	3	001,010,100 => 0 110,101,011 => 1
SECDED	0000,1111	4	One 1 => 0 Two 1's => error Three 1's => 1

ECC

# 1's	0	1	2	3	4
Result	0	0	Err	1	1

- Hamming distance
  - No. of changes to convert one code to another
  - All legal SECDED codes are at Hamming distance of 4
    - I.e. in single-bit SECDED, all 4 bits flip to go from representation for '0' to representation for '1'

ECC

- Reduce overhead by doing codes on word, not bit

# bits	SED overhead	SECDED overhead
1	1 (100%)	3 (300%)
32	1 (3%)	7 (22%)
64	1 (1.6%)	8 (13%)
n	1 (1/n)	1 + $\log_2$ n + a little

64-bit ECC

- 64 bits data with 8 check bits

  dddd…..d cccccccc
- Use eight by 9 SIMMS = 72 bits
- Intuition
  - One check bit is parity
  - Other check bits point to
    - Error in data, or
    - Error in all check bits, or
    - No error

ECC

- To store (write)
  - Use $data_0$ to compute $check_0$
  - Store $data_0$ and $check_0$
- To load
  - Read $data_1$ and $check_1$
  - Use $data_1$ to compute $check_2$
  - Syndrome = $check_1$ xor $check_2$
    - I.e. make sure check bits are equal

ECC Syndrome

Syndrome	Parity	Implications
0	OK	$data_1$==$data_0$
n != 0	Not OK	Flip bit n of $data_1$ to get $data_0$
n != 0	OK	Signal uncorrectable error

4-bit SECDED Code

Bit Position	001	010	011	100	101	110	110	111
Codeword	C_1	C_2	b_1	C_3	b_2	b_3	b_4	P
C_1	X		X		X		X	
C_2		X	X			X	X	
C_3				X	X	X	X	
P	X	X	X	X	X	X	X	X

$C_1 = b_1 \oplus b_2 \oplus b_4$
$C_2 = b_1 \oplus b_3 \oplus b_4$
$C_3 = b_2 \oplus b_3 \oplus b_4$
$P = even\_parity$

- C_n parity bits chosen specifically to:
  - Identify errors in bits where bit n of the index is 1
  - C_1 checks all odd bit positions (where LSB=1)
  - C_2 checks all positions where middle bit=1
  - C_3 checks all positions where MSB=1
- Hence, nonzero syndrome points to faulty bit

4-bit SECDED Example

$C_1 = b_1 \oplus b_2 \oplus b_4$
$C_2 = b_1 \oplus b_3 \oplus b_4$
$C_3 = b_2 \oplus b_3 \oplus b_4$
$P = even\_parity$

Bit Position	1	2	3	4	5	6	7	8	
Codeword	C_1	C_2	b_1	C_3	b_2	b_3	b_4	P	
Original data	1	0	1	1	0	1	0	0	Syndrome
No corruption	1	0	1	1	0	1	0	0	0 0 0, P ok
1 bit corrupted	1	0	0	1	0	1	0	0	0 1 1, P !ok
2 bits corrupted	1	0	0	1	1	1	0	0	1 1 0, P ok

- 4 data bits, 3 check bits, 1 parity bit
- Syndrome is xor of check bits C_{1-3}
  - If (syndrome==0) and (parity OK) => no error
  - If (syndrome != 0) and (parity !OK) => flip bit position pointed to by syndrome
  - If syndrome != 0) and (parity OK) => double-bit error

Summary

- Memory hierarchy: Register file
  - Under compiler/programmer control
  - Complex register allocation algorithms to optimize utilization
- Memory hierarchy: Virtual Memory
  - Placement: fully flexible
  - Identification: through page table
  - Replacement: approximate LRU or LFU
  - Write policy: write-through

Summary

- Page tables
  - Forward page table
    - &PTE = PTBR + VPN * sizeof(PTE)
  - Multilevel page table
    - Tree structure enables more compact storage for sparsely populated address space
  - Inverted or hashed page table
    - Stores PTE for each real page instead of each virtual page
    - HPT size scales up with physical memory
  - Also used for protection, sharing at page level

Summary

- TLB
  - Special-purpose cache for PTEs
  - Often accessed in parallel with L1 cache
- Main memory design
  - Commodity DRAM chips
  - Wide design space for
    - Minimizing cost, latency
    - Maximizing bandwidth, storage
  - Susceptible to soft errors
    - Protect with ECC (SECDED)
    - ECC also widely used in on-chip memories, busses