Chapter 6 Part B: Memory

Main Memory

- Static & Dynamic RAM
- Basic Memory
- Interleaving
- DRAM Tricks

Virtual Memory

- Basics
- Address Translation
- Address Translation / Cache Interactions
- Miscellaneous

Memory Technologies

Dynamic Random Access Memory (DRAM)

- Must refresh every 2 ms
  - Access every row
- Multiplexed Address/Data pins
  - Row Address Strobe (RAS)
  - Column Address Strobe (CAS)
- Optimized for density, not speed
  - One transistor cells
- Cycle time roughly twice access time
  - Destructive reads
- 1980: 64 kilobit, 150ns access time
- 1990: 4 megabit, 80ns access time
- JSSC 1990: 16 megabit, 55ns access time

Memory Technologies, cont.

Static Random Access Memory (SRAM)

- Static ==> No Refresh
- Optimized for speed, then density
  - 4-6 transistors per cell
  - Separate Address/Data pins
- Greater power dissipation than DRAM
- Access time = cycle time
- 1990: 256 kilobit (1MB?), 25ns access time
- 1990 JSSC: 4 megabit, 15ns access time

Simple Main Memory

Consider a memory with these parameters:

- 1 cycle to send address
- 6 cycles to access each word
- 1 cycle to send word back to CPU/Cache

What's the miss penalty for a 4-word block?

$(1 + 6\ \textit{cycles} + 1\ \textit{cycle}) \times 4\ \textit{words} = 32\ \textit{cycles}$

How can we speed this up?

Wider Main Memory

Make the memory wider

Read out all 2 (or more) words in parallel

Memory parameters:

1 cycle to send address

6 cycles to access each *doubleword*

1 cycle to send doubleword back to CPU/Cache

Miss penalty for a 4-word block:

$(1 + 6\ \textbf{\textit{cycles}} + 1\ \textbf{\textit{cycle}}) \times 2\ \textbf{\textit{doublewords}} = 16\ \textbf{\textit{cycles}}$

Cost

Wider bus

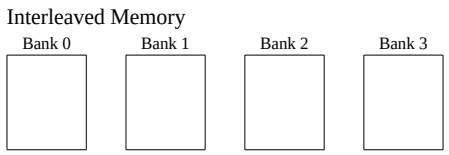
Larger expansion size

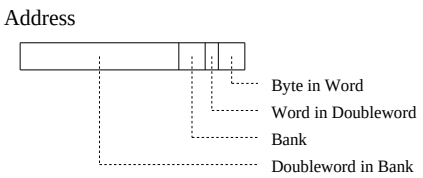
Error-correction is harder

Interleaved Main Memory

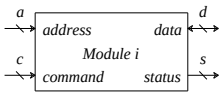
Organize memory into M banks so word A is stored in module (A mod M) at location (A div M)

Word In Module	Bank Number 0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
	...		...	

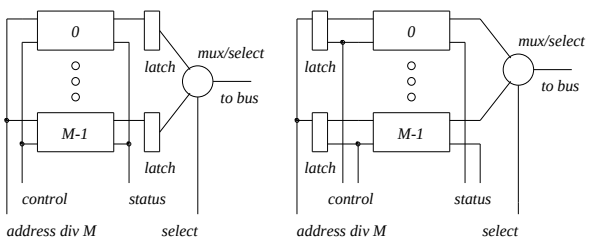




Interleaved Memory, cont.



*2**a by d-bit word memory module*



Simple Interleaving *Complex Interleaving*

See P. Kogge, *The Architecture of Pipelined Computers*, Hemisphere, 1981, pp. 39-44.

Interleaved Memory Examples

For each memory module, let A=2 (access time),
A+B=4 (busy time) & T=1 (transfer time)

Simple

Cycle	Address	Bank Number 0	1	2	3	A Steady-State Period
1	12	A	A	A	A	
2	12	A	A	A	A	
3	-	B/T	B	B	B	*
4	-	B	B/T	B	B	*
5	16	A	A	T/A	A	*
6	16	A	A	A	T/A	*
7	-	B/T	B	B	B	
8	-	B	B/T	B	B	
9	-	-	-	T	-	
10	-	-	-	-	T	

Complex

Cycle	Address	Bank Number 0	1	2	3	A Steady-State Period
1	12	A	-	-	-	
2	13	A	A	-	-	
3	14	B/T	A	A	-	*
4	15	B	B/T	A	A	*
5	16	A	B	B/T	A	*
6	17	A	A	B	B/T	*
7	18	B/T	A	A	B	
8	19	B	B/T	A	A	
9	-	-	B	B/T	A	
10	-	-	-	B	B/T	
11	-	-	-	-	B	

Complex Interleaving with Non-Unit Stride

Addresses = 12, 15, 18, ... (Stride = 3)

Cycle	Address	Bank Number 0	1	2	3	A Steady-State Period
1	12	A	-	-	-	
2	15	A	-	-	A	
3	18	B/T	-	A	A	
4	21	B	A	A	B/T	*
5	24	A	A	B/T	B	*
6	27	A	B/T	B	A	*
7	30	B/T	B	A	A	*
8	33	-	A	A	B/T	
9	-	-	A	B/T	-	
10	-	-	B/T	-	-	
11	-	-	B	-	-	

Addresses = 12, 14, 16, ... (Stride = 2)

Cycle	Address	Bank Number 0	1	2	3	A Steady-State Period
1	12	A	-	-	-	
2	14	A	-	A	-	
3	-	B/T	-	A	-	
4	-	B	-	B/T	-	*
5	16	A	-	B	-	*
6	18	A	-	A	-	*
7	-	B/T	-	A	-	*
8	-	B	-	B/T	-	
9	20	A	-	B	-	
10	22	A	-	A	-	
11	-	B/T	-	A	-	
12	-	B	-	B/T	-	
	...			...		
17	-	-	-	B	-	

Interleaved Memory Conclusions

Simple interleaving for sequential accesses

- Load cache blocks (& unit-stride vectors)

Complex interleaving otherwise

Do Both

- Banks
  - simple interleaving for high-bandwidth cache miss
- Superbanks
  - complex interleaving of banks to support multiple cache misses (non-blocking cache and/or multiple processors)

How many banks? Simplistic answer:

$$2 * \frac{words/cycle\ demanded}{words/cycle/bank}$$

DRAM Tricks

DRAM-Specific Optimizations

- First read entire row
- Then select column from row
- Stores entire row in a buffer

Nibble Mode

- DRAM can supply 3 extra bits from sequential addresses

Page Mode

- Row buffer acts like an SRAM.
- By changing column address, random bits can be access within a row.

Static Column

- Like page mode, but don't need to strobe CAS.

Virtual Memory

Original Motivation

- Avoid Overlays
- Use main memory as a cache for disk

Current motivation

- Relocation
- Protection
- Sharing
- Sparse use
- Fast start-up

Engineered differently than CPU caches

- Miss access time O(1,000,000)
- Miss access time ≫ miss transfer time

Virtual Memory, cont.

Blocks, called *pages*, are 512 to 16K bytes.

Page placement

Fully-associative -- avoid expensive misses

Page identification

Address translation -- virtual to physical address

Indirection through one or two page tables

Translation cached in translation buffer

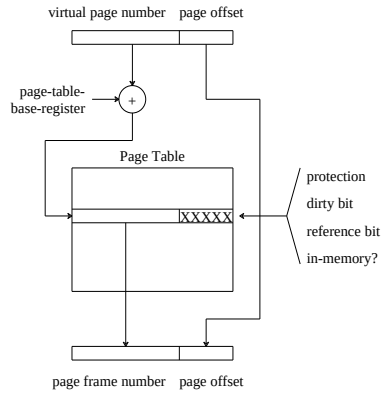
Page replacement

Approx. LRU -- CLOCK
''Working set''

Write strategy

Write-back (with page dirty bit)

Address Translation



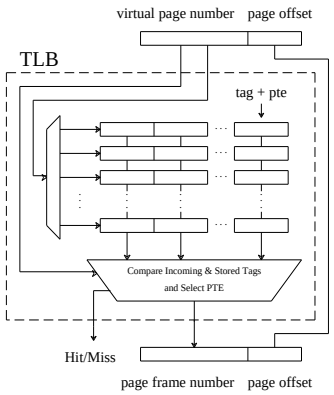
Logical Path

Two memory operations

Often two/three levels of page tables

TOO SLOW!

Address Translation, cont.



Fast Path

Translation Lookaside Buffer (TLB, TB)

A cache w/ PTEs for data

Number of entries 8 to 1024

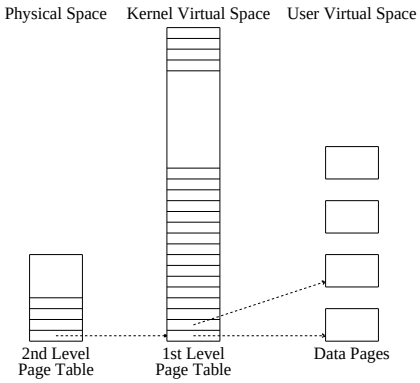
Address Translation, cont.

Multi-level page table (Virtual)

First-level PT in kernel virtual space
Second-level PT in physical memory

Hardware support

Base registers point to PTs
Limit registers may be provided



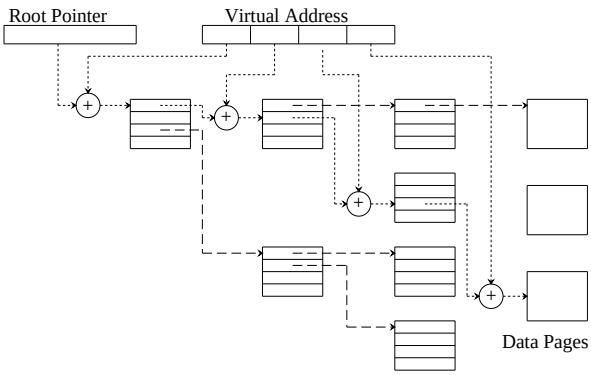
Address Translation, cont.

Multi-level page table (Physical)

All levels in physical memory

Hardware support

Base register points to root PT



E.g., Alpha

Address Translation, cont.

Inverted/Hashed Page Table

Hash with function of virtual address

Size = (physical memory size / page size)

$\times$ entry size $\times$ safety factor (e.g., 2)

Inverted

Entry has tag & pointer to PTE

Hashed

Entry has tag, first PTE, & pointer to 2nd

Less time but more space

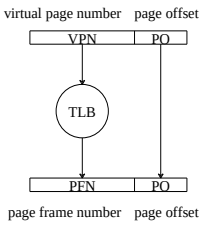
Issues

Open or closed chaining

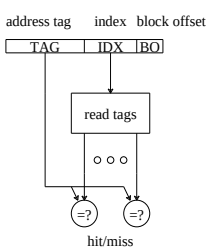
Aliases (synonyms)

Address Translation / Cache Interaction

Address Translation

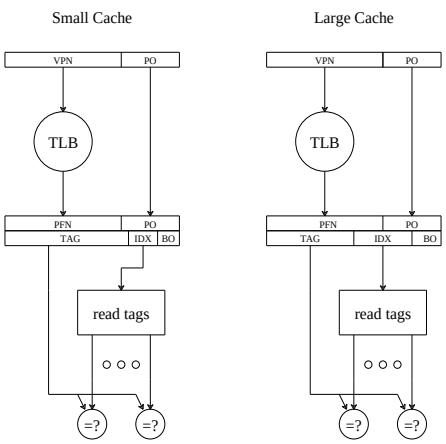


Cache Lookup (tags only)



Sequential TLB Access

Address translation before cache lookup



Problems

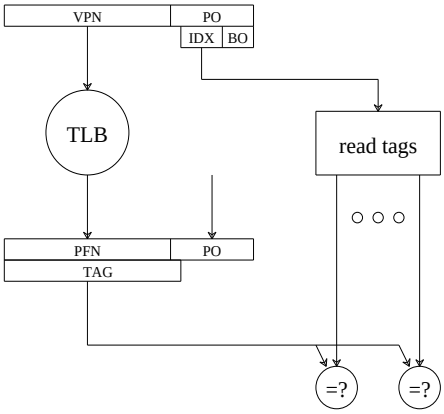
Slow

May increase cycle time, CPI, pipeline depth

Parallel TLB Access

Address translation in parallel with cache lookup

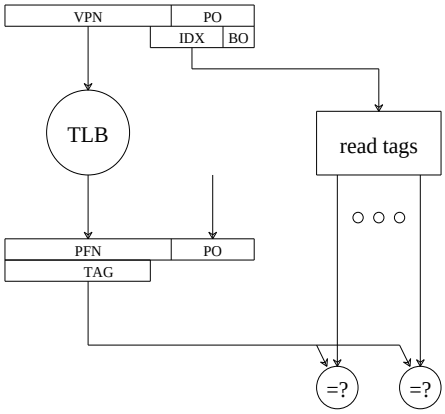
Small Cache



Parallel TLB Access

Address translation in parallel with cache lookup

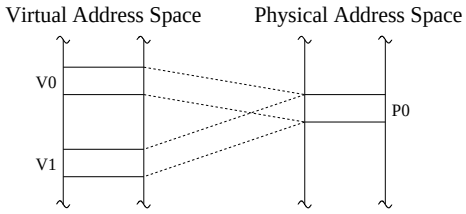
Large Cache

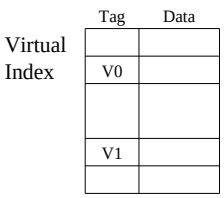


Index taken from virtual page number

Could cause problems from synonyms

Virtual Address Synonyms





Solutions to Synonyms

(1) Limit cache size to page size times assoc

Extract index from page offset

(2) Search all sets in parallel

e.g., 64 KB 4-way cache w/ 4KB pages

Search 4 sets (16 entries) in parallel

(3) Restrict page placement in operating system

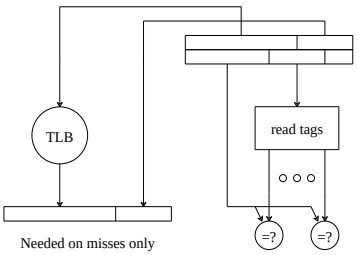
Guarantee that Index(VA) == Index(PA)

(4) Eliminate by operating system convention

Single virtual address space

Restrictive sharing model

Virtual Address Cache



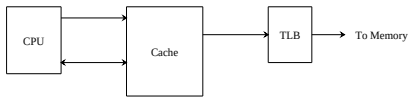
Address translation after cache miss

- Implies fast-lookup even for large caches

Must handle

- Virtual-address synonyms (aliases)
- Virtual-address space changes
- Status and protection bit changes

Virtual Address Cache



Observation:

- (1) A TLB is a small special-purpose cache that hold PTEs.
- (2) The system already has a large general-purpose cache for instructions and data.

Question:

- Can we combine the TLB with the general-purpose cache?

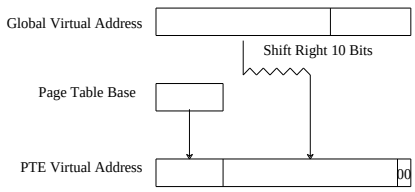
In-Cache Address Translation [Wood 1986]

Cache Hit

- No translation necessary

Cache Miss

- Compute PTE virtual address



- Access the cache

1st-level PTE Miss

- Repeat translation process

2nd-level PTE Miss

- Physical base pointer terminates recursion

Protection

Goal:

- Sequential semantics
- One process should not be able to interfere with the execution of another

Process model

- Privileged kernel
- Independent user processes

Primitives vs. Policy

- Architecture provides the primitives
- Operating system implements the policy
- Problems arise when hardware implements policy

Protection Primitives

User vs. Kernel

- At least one privileged mode
- Special case of *rings*
- Usually implemented as mode bit(s)

How do we switch to kernel mode?

- Protected ‘‘gates’’
- Change mode and continue execution at *pre-determined* location.

Hardware to compare mode bits to access rights.

- Only access certain resources in kernel mode

Protection Primitives, cont.

Base and Bounds

- Privileged registers
- Base ≤ Address ≤ Bounds

Segmentation

- Multiple base and bounds pairs
- Protection bits for each segment

Page-level protection

- Protection bits in page table entry
- Cache them in TLB

Caches & I/O

What happens if I/O device uses DMA (direct memory access) to write memory addresses that are currently cached?

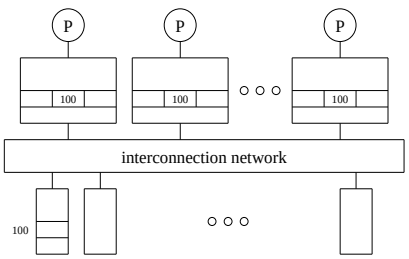
Solution 1: Disallow

- Require that O.S. flush & make inaccessible any buffer used with DMA
- + Simple hardware
- - Complicates O.S. & adds overhead

Solution 2: Cache Coherence Hardware

- Hardware at cache invalidates or updates data as DMA is done
- +/- Complement of above
- Necessary for multiple processors...

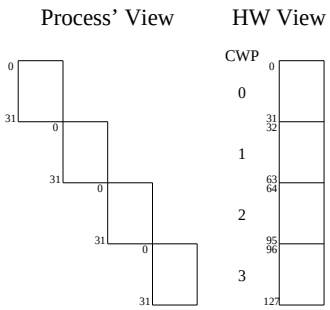
Cache Coherence



Solutions (see CS/ECE 757)

- No caches
- Shared cache
- Private caches for private & read-only data
- Cache coherence protocol
  - Snooping -- caches active
  - Directory -- memory knows
  - See H&P Figure 8.44.

For Registers



Register Windows

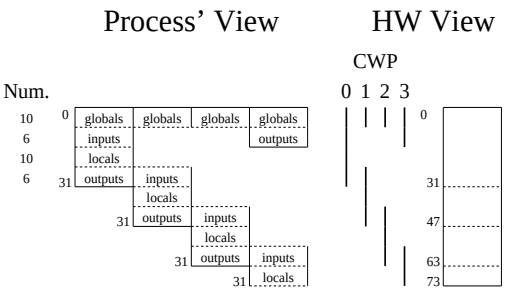
- Improve call/return performance
- Reduce register save/restore traffic
- Allow larger register file

But

- Traps on register window over/under-flow
- Slower on decode and context switches

For Registers, cont.

Add global registers and overlapping windows (RISC II, SPARC)



If ($0 \le REG \le 9$)

- Then physical_reg = REG
- Else physical_reg = ‘‘(CWP<<4) + REG’’†

† $10 < [(CWP<<4) + REG - 10) \bmod 64] + 10 < 74$

Putting It All Together: Alpha AXP 21064

Instructions

- TLB
- Cache
- Prefetch Buffer

Data

- TLB
- Cache
- Write Buffer

Shared

- Level 2 Cache
- Victim Buffer
- Main Memory
- Disks