CE/CZ 3001:
Advanced Computer Architecture

Module 5: Memory Systems

- cache replacement and write policies

Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Big picture

Burning questions:

- Placement (In which cache line can a block of memory be placed?)
  - Depends on the type of cache and associativity
- Identification (How do I find a block of memory in the cache?)
  - By decoding the tag bits and index field
- Replacement (Which block should I replace if there is a miss?)
- Write Policy (How do I keep the memory updated when I write on the cache?)

Replacement Policy

- How do we choose *victim cache line*?
  - Verbs: *Victimize, evict, replace, cast out*

- Several policies are possible
  - FIFO (first-in-first-out)
  - LRU (least recently used)
    - Takes care of temporal locality
    - Needs to keep history of access and hence slows down the system
  - NMRU (not most recently used)
  - Pseudo-random (yes, really!)

- Pick victim within *set* where a = *associativity*
  - If a <= 2, LRU is cheap and easy (1 bit)
  - If a > 2, it gets harder
  - Pseudo-random works pretty well for caches

Write Policies (Part 1/2)

- Memory hierarchy
  - Two or more copies of the same block
    - Main memory and/or disk
    - Caches
- What to do on a write?
  - Eventually, all copies must be changed
  - Cache write must *propagate* to all levels of memory hierarchy

Write Policies (Part 2/2)

- Easiest policy: *write-through*
- Every write propagates directly through hierarchy
  - Write in L1, L2, memory, disk
- Why is this a bad idea?
  - Very high bandwidth requirement
  - Memory becomes slow when its size increases
- Popular in real systems only for writing to the L2
  - Every write updates L1 and L2
  - Beyond L2, use *write-back* policy

- Most widely used : *write-back*
- Maintain *state* of each line in a cache
  - Invalid – not present in the cache
  - Clean – present, but not written (unmodified)
  - Dirty – present and written (modified)
- Store state in tag array, next to address tag
  - Mark dirty bit on a write
- On eviction, check dirty bit
  - If set, write back dirty line to next level
  - Called a *writeback* or *castout*

What are the complications in write policies

- Complications of write-back policy
  - More stale copies in the lower level of hierarchy
  - Must always check higher level for dirty copies before accessing copy in a lower level
- Not a big problem in uniprocessors
  - In multiprocessors: *the cache coherence problem*
- I/O devices that use DMA (direct memory access) can cause problems even in uniprocessors
  - Called coherent I/O
  - Must check caches for dirty copies before reading main memory

CE/CZ 3001:
Advanced Computer Architecture

Module 5: Memory Systems

- **a cache example**
- **instruction cache vs data cache**

Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Cache example (Part 1/8)

Memory 64 byte:-
address bits =6

BS=4 bytes/block,
S=4 sets,
E=2(two way set associative),
Total number of cache blocks = E*S = 8 blocks.

o=2, i=2, t=2;
2-way set-associative

Only tag array shown here

- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss

Tag Array

Tag0/LRU	Tag1/LRU

Cache example (Part 2/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss

Tag Array

Tag0/LRU	Tag1/LRU
10/0	

Cache example (Part 3/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit

Tag Array

Tag0/LRU	Tag1/LRU
10/0	

Cache example (Part 4/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss

Tag Array

Tag0/LRU	Tag1/LRU
10/0	
11/0	

Cache example (Part 5/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss

Tag Array

Tag0/LRU	Tag1/LRU
10/0	
10/0	
11/0	

Cache example (Part 6/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss

Tag Array

Tag0/LRU	Tag1/LRU
10/1	11/0
10/0	
11/0	

Cache example (Part 7/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss
Load 0x11	010001	0/0 (lru)	Miss/Evict

Tag Array

Tag0/LRU	Tag1/LRU
01/0	11/1
10/0	
11/0	

Cache example (Part 8/8)

o=2, i=2, t=2;

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss
Load 0x11	010001	0/0 (lru)	Miss/Evict
Store 0x29	101001	2/0	Hit/Dirty

Tag Array

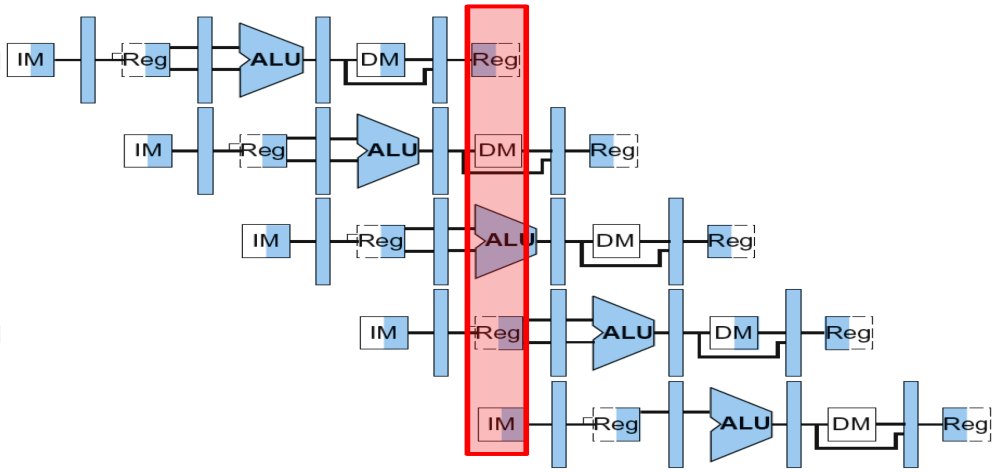
Tag0/LRU	Tag1/LRU
01/0	11/1
10/0 (d)	
11/0	

Instruction cache vs data cache (Part 1/2)

- Till now we considered cache to be unified- (Von Neuman)
- Data cache:– stores data only
  - When data is requested it looks in data cache
  - If found, retrieves information and execution continues
  - If not found, data is retrieved from memory and placed in data cache
  - For data that exhibit good locality – high hit-rate
- Instruction cache:– stores instructions only
  - Most programs instructions occurs sequentially, with occasional branching
  - Tends to have high locality and thus high hit rate

Instruction cache vs data cache (Part 2/2)

- Unified memory
  - Has only one port for data and instructions
    - Results in conflicts between two



- Hence better to have two separate memories (Harvard)