Lecture 23

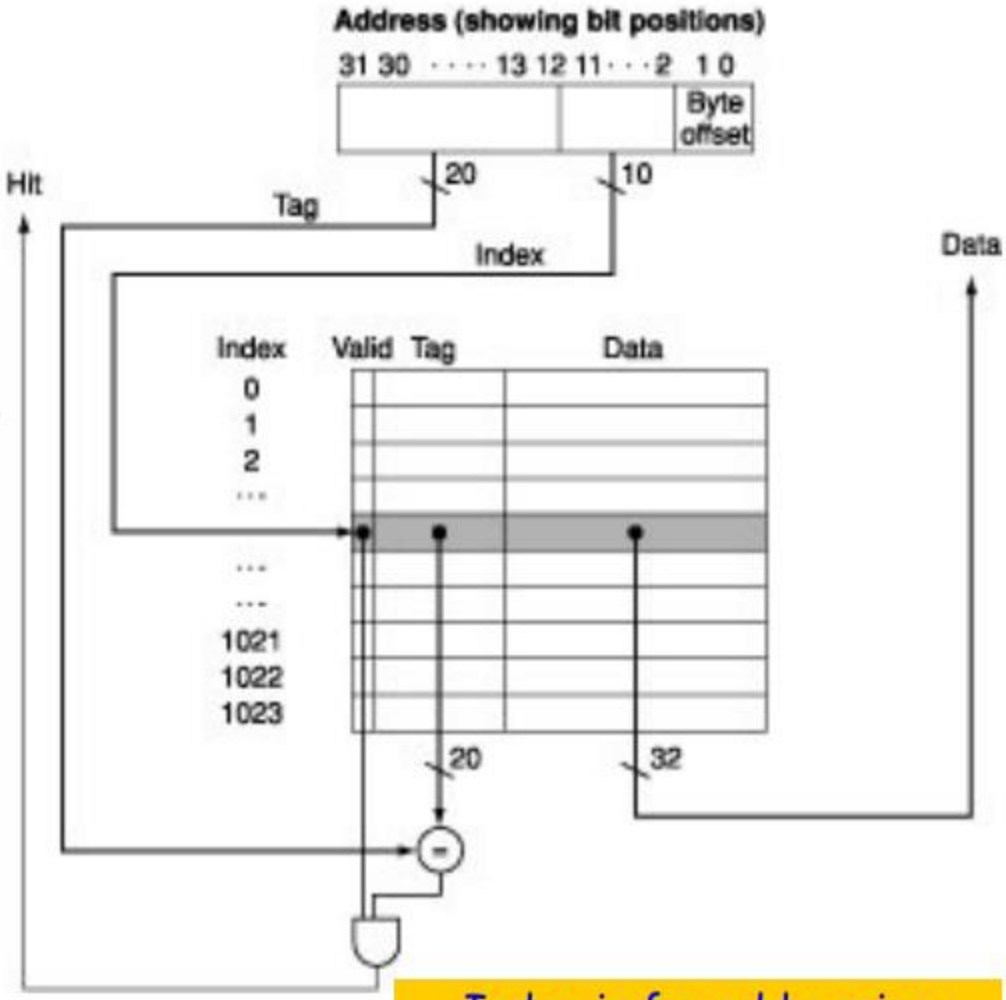
Computer Architecture
(Spring 2020)

Memory Hierarchy Design

Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

Ex: Direct Mapped with 1024 Blocks Frames and Block Size of 1 Word for MIPS-32

- **Block Offset**
  - is just a byte offset because each block of this cache contains 1 word
- **Byte Offset**
  - least significant 2 bits because in MIPS-32 memory words are aligned to multiples of 4 bytes
- **Block Index**
  - 10 low-order address bits because this cache has 1024 block frames
- **Block Tag**
  - remaining 20 address bits in order to check that the address of the requested word matches the cache entry



Index is for addressing
Tag is for checking/searching

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

- **Assumption**
  - **–8 block frames**
  - **– block size = 1 word**
  - **– main memory of 32 words**
  - **•toy example**
  - **– we consider ten subsequent accesses to memory**

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2			
3			
4			
5			
6			
7			
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3			
4			
5			
6			
7			
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4			
5			
6			
7			
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5			
6			
7			
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5	10000	16	miss
6			
7			
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5	10000	16	miss
6	00011	3	miss
7			
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5	10000	16	miss
6	00011	3	miss
7	10000	16	hit
8			
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5	10000	16	miss
6	00011	3	miss
7	10000	16	hit
8	10010	18	miss
9			
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5	10000	16	miss
6	00011	3	miss
7	10000	16	hit
8	10010	18	miss
9	11010	26	miss
10			

Example: Accessing a Direct Mapped Cache with 8 Blocks and Block Size of 1 Word

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

cycle	Memory Address	address in decimal	Cache Event
1	10110	22	miss
2	11010	26	miss
3	11010	26	hit
4	10110	22	hit
5	10000	16	miss
6	00011	3	miss
7	10000	16	hit
8	10010	18	miss
9	11010	26	miss
10	11010	26	hit

A Continuum: From Direct Mapping to Fully Associativity (via Set Associativity)

- **In Direct-Mapped Caches sets and blocks coincide**
  - **– the index points to a "set-block" and the tag comparison is used simply to determine if this set-block is a hit or a miss**
  - **Direct Mapping gives fastest hit times, best for very large caches**
- **In Fully-Associative Caches there are no indexes, all cache accesses address the same set, and it is the tag comparison that is used to determine which block, if any, is returned**
  - **– fully-associative caches require a comparison with each entry, which can be implemented with many parallel comparators. This, however, is a considerable hardware cost that makes sense only for small caches**
  - **Full associativity gives lowest miss rate, best when miss penalty is very high**

Summary: Cache Schemes

- **Direct Mapped Cache**
  - **a block can be only in one place**
- **Set Associative Cache**
  - **a block can be in any element of the set**
    - **all tags of all elements in the set must be searched**
- **Fully Associative Cache**
  - **blocks can go anywhere**
  - **all tags of all blocks in the cache must be searched**
    - **many parallel comparisons (a comparator for each cache entry)**
    - **practical only for caches with small number of blocks**
- **Increasing degree of associativity**
  - **typically decreases miss rate**
  - **increases hit time (due to extra comparison/selection)**
    - **design trade-off between miss penalty and area/time overhead**

Cache Write Policies

Writes are slightly more problematic than reads

- tolerate temporary inconsistencies between cache and memory?

– fortunately, there are typically more reads than writes

- MIPS measures with Spec Int say 10% stores and 37% loads

– writes are 10% / (100% + 47%) = 7% of total memory traffic

– writes are 10% / (47%) = 21% of total data cache traffic

– caches are optimized for reads, but recall Amdahl's law…

- easier "to read": block read and tag comparison done in parallel
- writes can modify a block only after tag comparison is positive

– two write policies

- write back (write only cache block and set a "dirty bit" as reminder)

– uses less memory bandwidth and it is a low-power approach

- write through (write block on cache and memory simultaneously)

– requires a write buffer to minimize write stall

Cache Write Policies – cont.

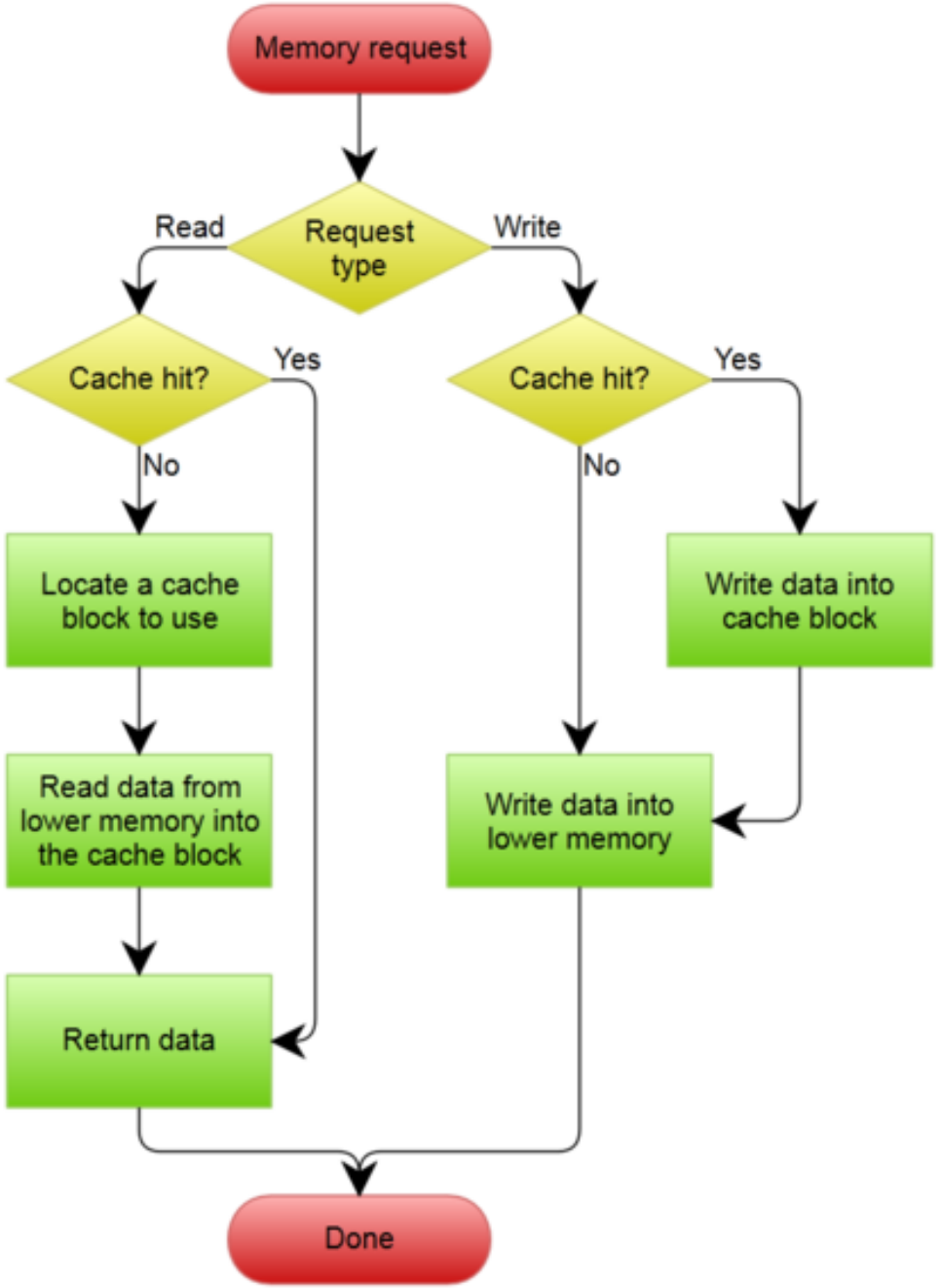
- **Advantages of Write-Back Policy**
  - **+ writes occur at the speed of the cache memory**
  - **+ multiple writes within a block require only one write to the lower-level memory**
  - **+ some writes don't go to memory**
  - **+ write back uses less memory bandwidth (good for multiprocessors) and dissipate less power (good for embedded applications)**
- **Advantages of Write-Through Policy**
  - **+ easier to implement than write back**
  - **+ cache is always clean, so read misses are faster as they never result in writes to lower level**
  - **+ next lower level has the most current copy of the data**
  - **+ simplifies data coherency (good for multiprocessors and I/O)**

What Happens on a Write Miss

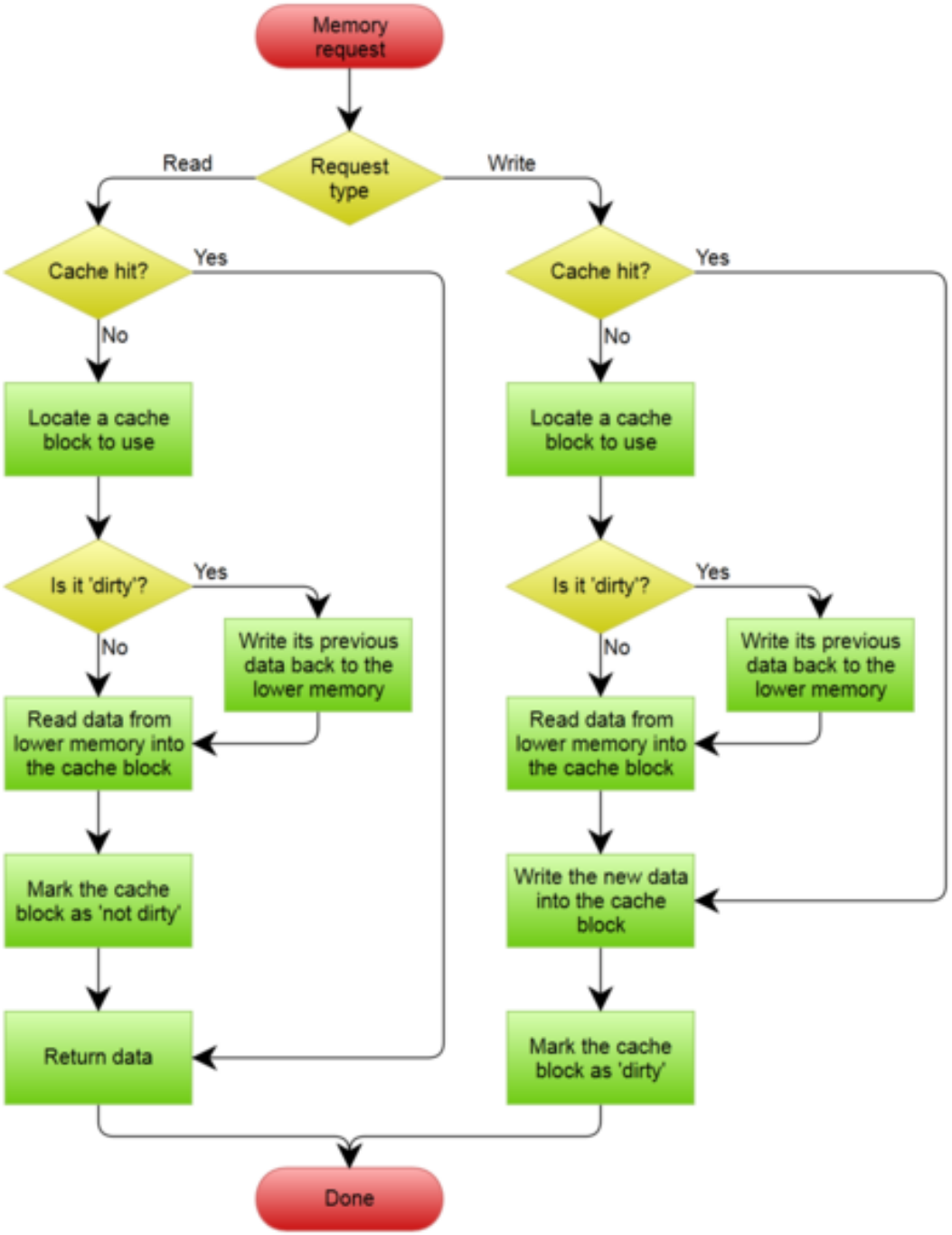
- **since data are "not needed" on a write there are two options:**
  - **Write Allocate (typically used by write-back caches)**
    - **the block is first allocated and then the same policy that is used for a write hit is executed**
  - **No-Write Allocate (typically used by write-through caches)**
    - **the cache is not affected by a write miss while the block is modified only in the lower-level memory**
      - **thus a block stays out of the cache until the program tries to read it**
- **Example**

Memory References	Write Allocate	No-Write Allocate
Store Mem[100]	miss	miss
Store Mem[100]	hit	miss
Load Mem[200]	miss	miss
Store Mem[200]	hit	hit
Store Mem[100]	hit	miss

A Write-Through cache with No-Write Allocation



A Write-Back cache with Write Allocation



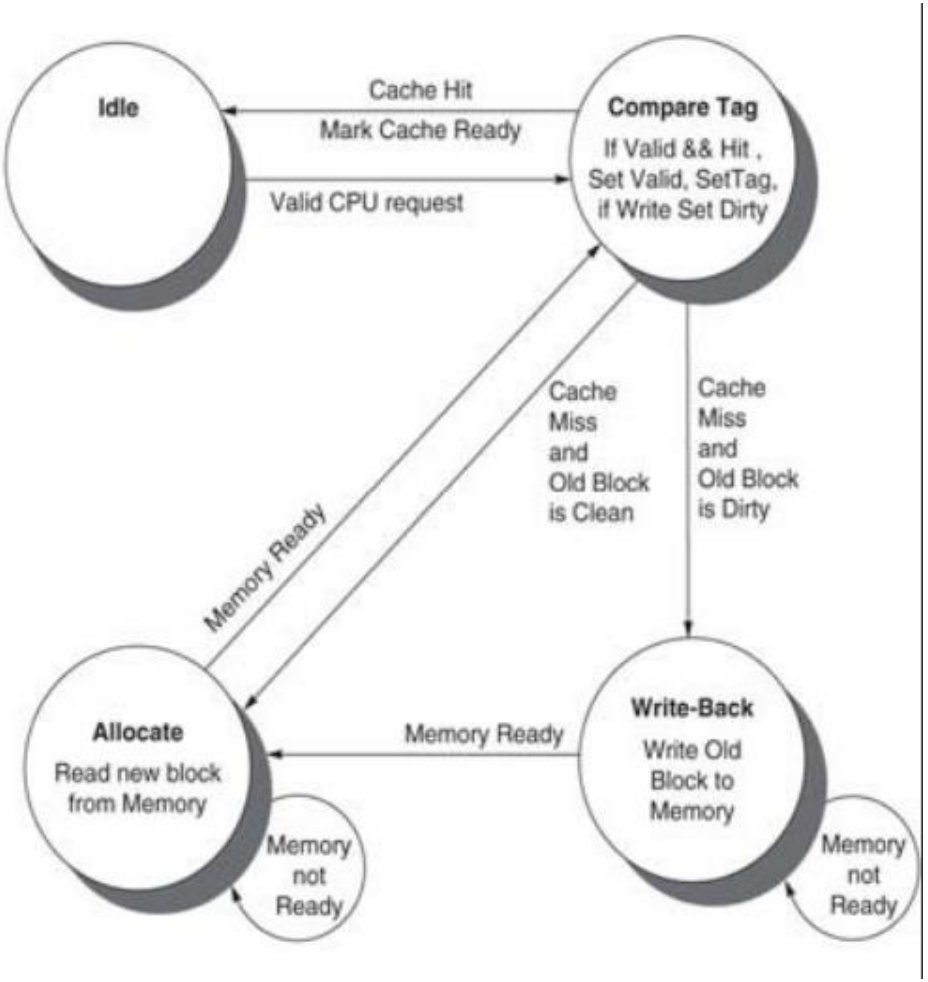
Block Replacement Policies on a Cache Miss

- **Direct mapped caches**
  - **No choice in picking the "victim entry": only one block frame is checked and if necessary replaced**
- **Set-associative or fully associative caches**
  1. **Random**
     - **to spread allocation uniformly**
  2. **LRU (Least-Recently Used)**
     - **application of principle of (temporal) locality**
  3. **FIFO (round-robin)**
     - **similar idea as LRU, but simpler/cheaper to implement**
  4. **NRU (not recently used)**
     - **approximation of LRU, but simpler/cheaper to implement**

     – a shift register is associated to each entry, shifting at each cycle (1 is inserted if entry is used, 0 otherwise); then, the entry with the lowest number is the victim

Example: FSM for a Simple Cache Controller for a Write-Back Cache with Write Allocate Policy

- **Idle**
  - **Wait for read/write request from CPU**
- **Compare Tag**
  - **on a miss, first update the cache tag then go in either Allocate or Write-Back depending on the value of the dirty bit**
- **Write-Back**
  - **remain here until memory can be written**
- **Allocate**
  - **remain here until memory read is complete**



Cache Performance: CPU Execution Time

- When the processor finds a requested data item in the cache, it is called a *cache hit*. Otherwise, a *cache miss* occurs.
- The time required for the cache miss depends on both the latency and bandwidth of the memory.
  - Latency determines the time to retrieve the first word of the block
  - bandwidth determines the time to retrieve the rest of this block.
- The CPU execution time now includes two parts:

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{CPU}_{\text{execution time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{memAccPerInstr.} \times \text{MissRate} \times \text{MissPenalty}) \times \text{CCT}$$

Cache Performance: Memory Stall Cycles

- IC: Instruction Count.
- Memory Accesses:
  - Each instr. has at least 1 memory access – loading the instr.
  - Some may have a data access.
- Miss Rate: the fraction of cache accesses that result in a miss
- Miss Penalty: an average # of stall cycles needed for a miss.

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$= \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

often reported as misses per 1000 instructions

$$= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}$$

- Could be further classified into Write and Read

$$\text{Memory stall clock cycles} = \text{IC} \times \text{Reads per instruction} \times \text{Read miss rate} \times \text{Read miss penalty} + \text{IC} \times \text{Writes per instruction} \times \text{Write miss rate} \times \text{Write miss penalty}$$

Example: The Impact of Cache on Performance

- **Assumptions**
  - **CPI_exec = 1 clock cycle (ignoring memory stalls)**
  - **Miss rate = 2%**
  - **Miss penalty = 200 clock cycles**
  - **Average memory references per instruction = 1.5**
- **$(CPI)_{no\_cache} = 1 + 1.5 \times 200 = 301$**
- **$(CPI)_{with\_cache} = 1 + (1.5 \times 0.02 \times 200) = 7$**

Example: Calculating Cache Performance

- **Assumptions**
  - **– CPI_exec = 2 clock cycles (ignoring memory stalls)**
  - **– Miss rate for instructions = 2% and miss rate for data = 4%**
  - **– Miss penalty = 100 clock cycles**
  - **– Aggregate load and store instruction frequency = 36%**
- **How much faster the processor would run with a perfect cache that never misses?**
  - **– Instruction miss cycles = IC x 2% x 100 = 2.0 x IC**
  - **– Data miss cycles = IC x 36% x 4% x 100 = 1.44 x IC**
  - **– Total number of memory stall cycles = 3.44 x IC**
  - **– (CPI)with_cache = 2 + (3.44) = 5.44**
  - **– (CPI)perfect_cache = 2**
  - **– The performance with perfect cache is 2.72 faster**
- **What if the processor is made faster (e.g. CPI_exec = 1)?**
  - **– The performance with perfect cache would be 4.44 faster**
  - **– The amount of execution time spent on memory stalls would rise from 3.44/5.44 = 63% to 3.44/4.44=77%**

Cache Performance: Average memory access time

- *hit time* is the time to hit in the cache
  - Could be in the unit of time, or # of cycles.
- The time required for the cache miss depends on both the latency and bandwidth of the memory.
  - Latency determines the time to retrieve the first word of the block
  - bandwidth determines the time to retrieve the rest of this block.
- The average memory access time now includes two parts:

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

Example: Cache Performance

Question 1 (PP. B-5)：

Assume we have a computer where the cycles per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instructions were cache hits?

Question 2 (PP. B-6)：

To show equivalency between the two miss rate equations, let's redo the example above, this time assuming a miss rate per 1000 instructions of 30. What is memory stall time in terms of instruction count?

Example: Cache Performance

Question 3 (PP. B-16)：

Which has the lower miss rate: a 16 KB instruction cache with a 16 KB data cache or a 32 KB unified cache? Use the miss rates in Figure B.6 to help calculate the correct answer, assuming 36% of the instructions are data transfer instructions. Assume a hit takes 1 clock cycle and the miss penalty is 100 clock cycles. A load or store hit takes 1 extra clock cycle on a unified cache if there is only one cache port to satisfy two simultaneous requests. Using the pipelining terminology of Chapter 3, the unified cache leads to a structural hazard. What is the average memory access time in each case? Assume write-through caches with a write buffer and ignore stalls due to the write buffer.

Example: Cache Performance

Question 4 (PP. B-18)：

Let's use an in-order execution computer for the first example. Assume that the cache miss penalty is 200 clock cycles, and all instructions normally take 1.0 clock cycles (ignoring memory stalls). Assume that the average miss rate is 2%, there is an average of 1.5 memory references per instruction, and the average number of cache misses per 1000 instructions is 30. What is the impact on performance when behavior of the cache is included? Calculate the impact using both misses per instruction and miss rate.

Equations for Calculating Cache Performance (PP. B-22)

$$2^{\text{index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}}$$

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}} \right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{Total miss latency} - \text{Overlapped miss latency})$$

$$\text{Average memory access time} = \text{Hit time}_{\text{L1}} + \text{Miss rate}_{\text{L1}} \times (\text{Hit time}_{\text{L2}} + \text{Miss rate}_{\text{L2}} \times \text{Miss penalty}_{\text{L2}})$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}_{\text{L1}}}{\text{Instruction}} \times \text{Hit time}_{\text{L2}} + \frac{\text{Misses}_{\text{L2}}}{\text{Instruction}} \times \text{Miss penalty}_{\text{L2}}$$

Quiz #4

Given: A cache can hold 64 KByte data. Data are transferred between main memory and the cache in blocks of 4 bytes each. The main memory consists of 16 Mbytes, with each byte directly accessible by a 24-bit address.

(a) How many cache lines in the cache?

(b) How many blocks in the memory?

(c) For the direct-mapping cache, how many bits for tag and cache line in a memory address, respectively?

(d) For the associative cache, how many bits for tag in a memory address?

(e) For the 4-way set associative cache, how many bits for tag and set in a memory address, respectively?

(f) How many tag comparisons are needed in each memory access for direct-mapping, associative and four-way set associate cache, respectively?