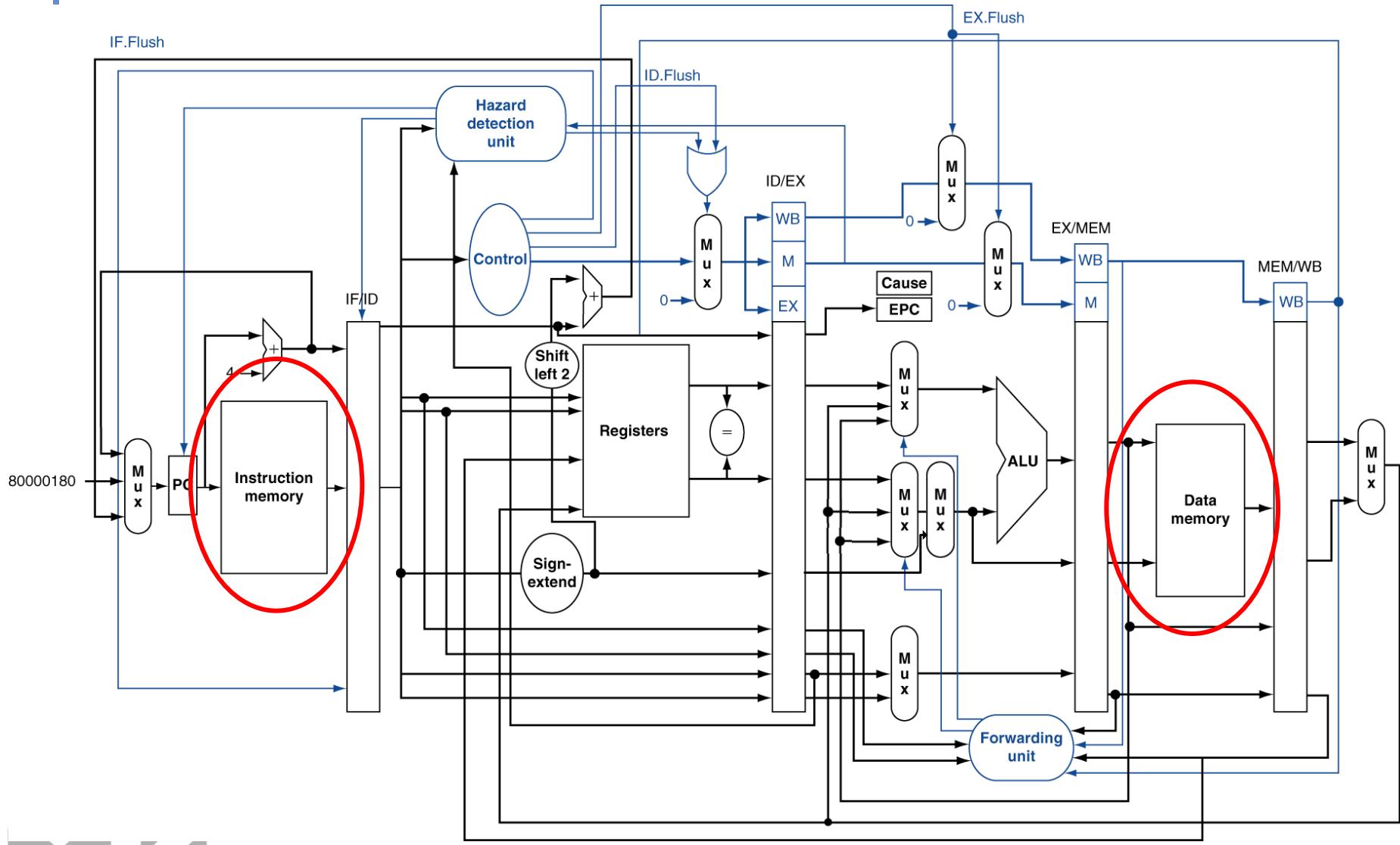




Topic 11

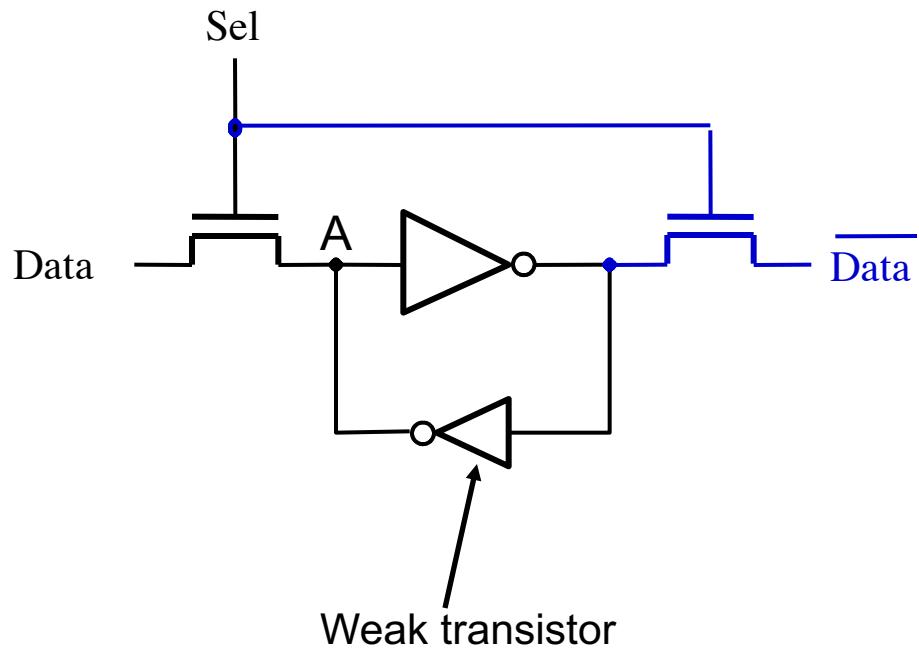
Memory Hierarchy - Cache

MIPS Pipeline Architecture



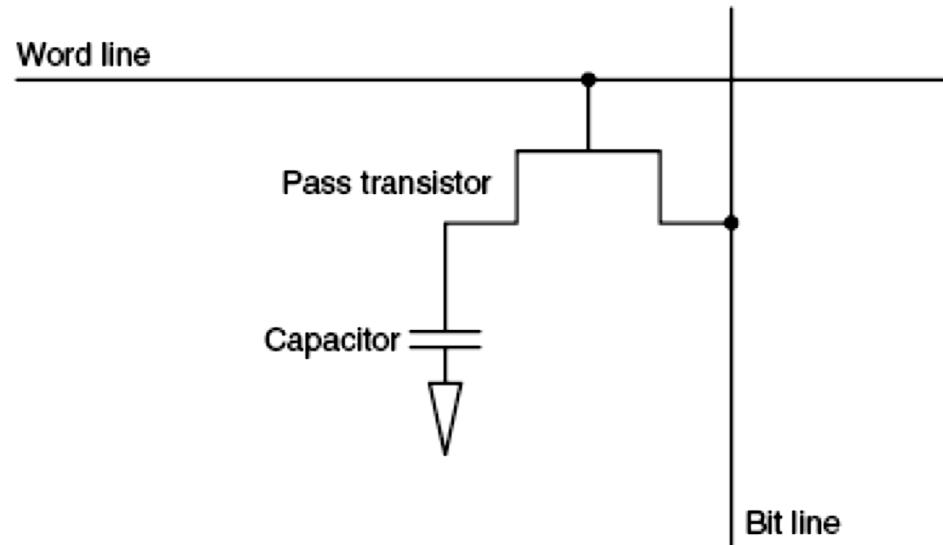
Static RAM (SRAM)

- When Sel = 1, Data is stored and retained in the SRAM cell by the feedback loop
- When Sel = 0, Data can be read out on the same port
- Point A is driven by both the Data transistor and the smaller inverter, but the Data transistor wins because the inverter is implemented using a weak transistor



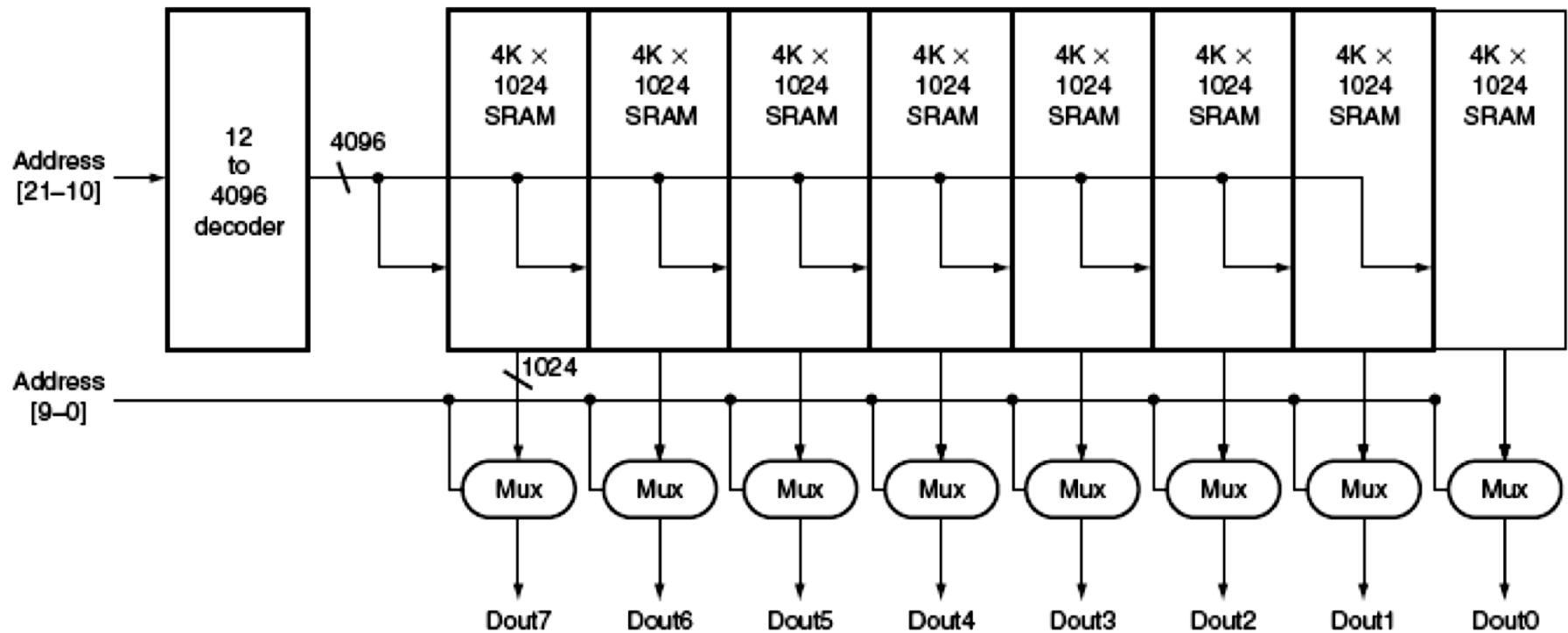
Dynamic RAM (DRAM)

- Write: turn on word line, charge capacitor through pass transistor by bit line
- Read: charge bit line halfway between high and low, turn on word line, then sense the voltage change on bit line
 - 1 if voltage increases
 - 0 if voltage decreases



Memory

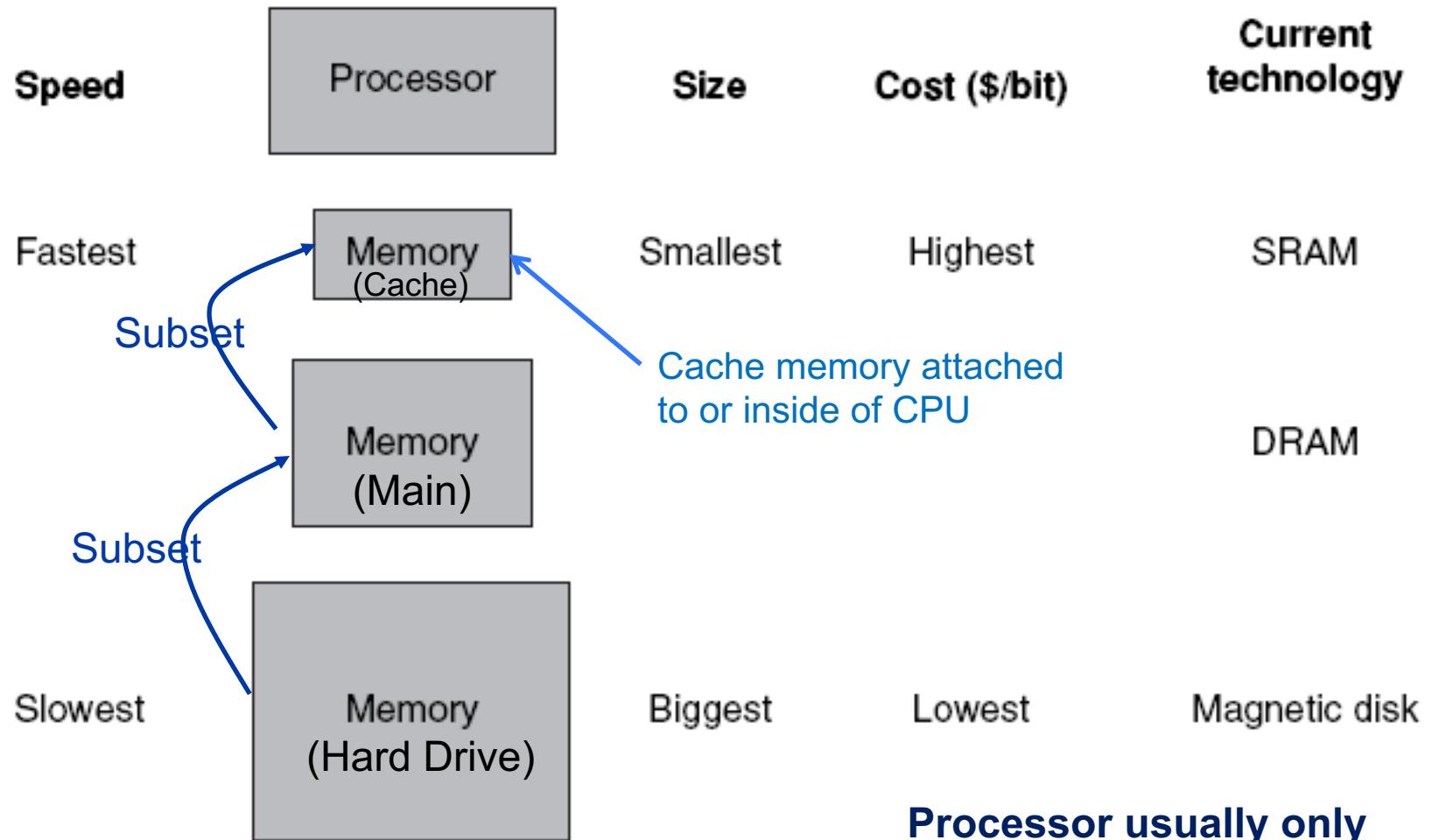
- Typical memory organization



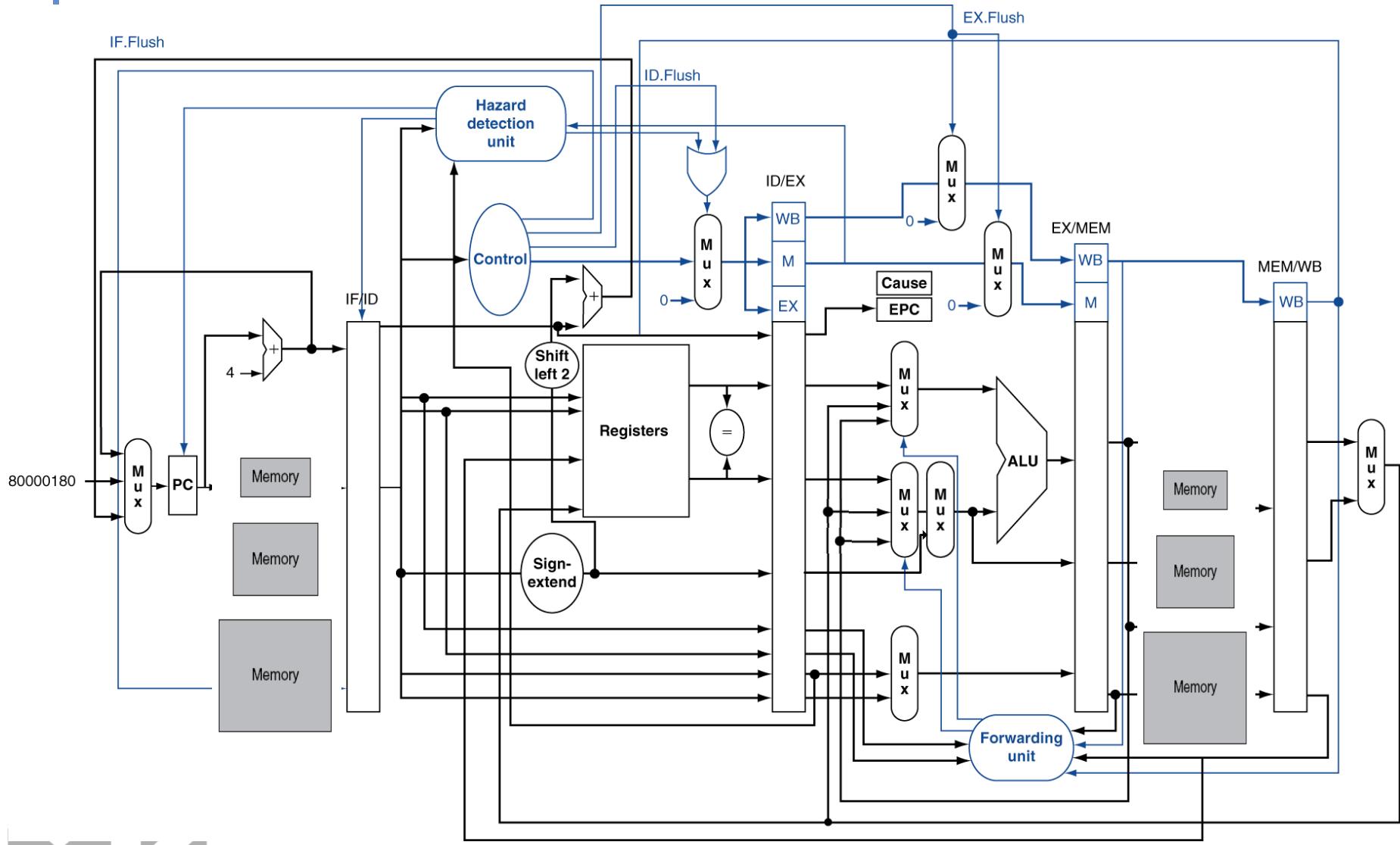
Memory Technology

- Static RAM (SRAM)
 - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)
 - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk
 - 5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory
 - Access time of SRAM
 - Capacity and cost of disk

Memory Hierarchy



MIPS Pipeline Architecture



Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
 - Items that are accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop
- Spatial locality
 - Items near those that are accessed recently are likely to be accessed soon
 - E.g., sequential instruction access, array data



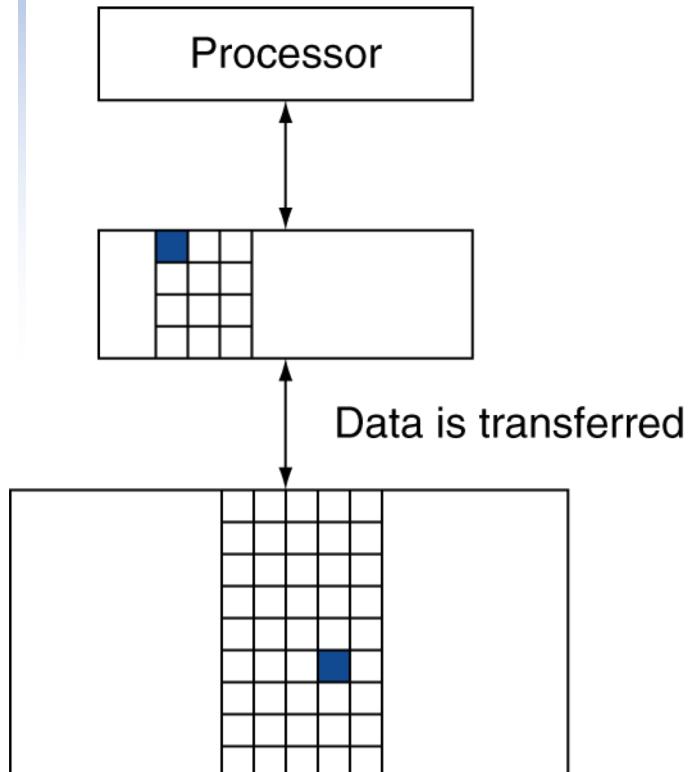
Principle of Memory Access

- Taking Advantage of Locality
- If a word of data/instruction is referenced
 - Copy this recently accessed word (temporal locality) and nearby items (spatial locality) together as a block from lower level memory to higher level memory (Hard Drive to Main memory or Main memory to Cache)



Memory Hierarchy Levels

- Concepts:
 - Block (aka line)
 - Unit of data referencing to take advantage of temporal and spatial locality
 - May be one or multiple words
 - Block also has an address
 - Hit
 - If accessed data is present in upper level, access satisfied by upper level
 - Hit rate: hits/accesses
 - Miss
 - If accessed data is absent
 - Block copied from lower to higher level
 - Then accessed data supplied from upper level
 - Time taken: miss penalty
 - Miss rate: misses/accesses
 $= 1 - \text{hit rate}$



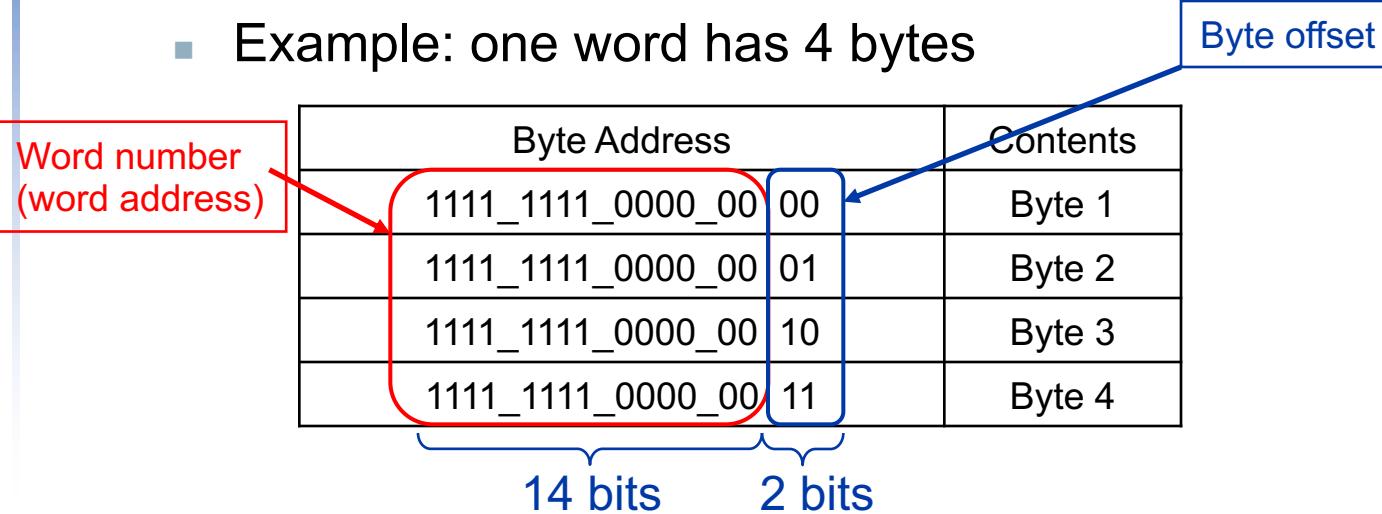
Hit Time and Miss Penalty

- Hit time
 - Time to access a memory including
 - Time to determine whether a hit or miss
 - Time to pass block to requestor
- Miss (time) penalty
 - Time to fetch a block from lower level upon a miss including
 - Time to access the block
 - Time to transfer it between levels
 - Time to overwrite the higher level block
 - Time to pass block to requestor

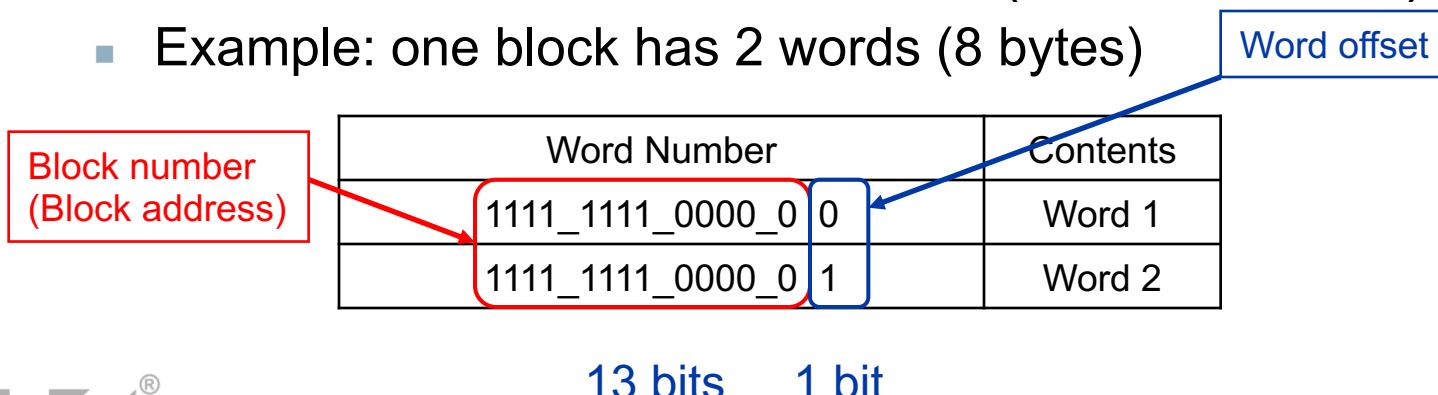


Block, Word, Byte Addresses

- Byte address vs. word address (assume 16-bit address)
 - Example: one word has 4 bytes

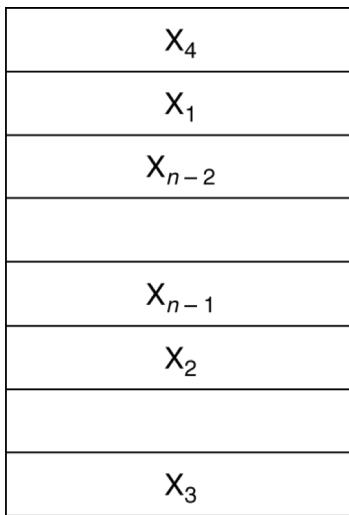


- Word address vs. block address (block number)
 - Example: one block has 2 words (8 bytes)

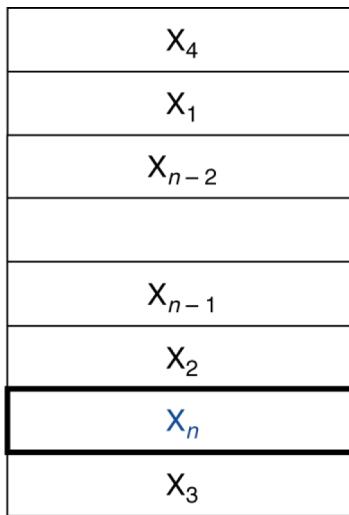


Cache Memory

- Cache memory
 - The level of the memory hierarchy closest to the CPU
- Example: assume a cache with 1-word blocks X_1, \dots, X_{n-1} (word address = block address). Now CPU requests X_n



a. Before the reference to X_n



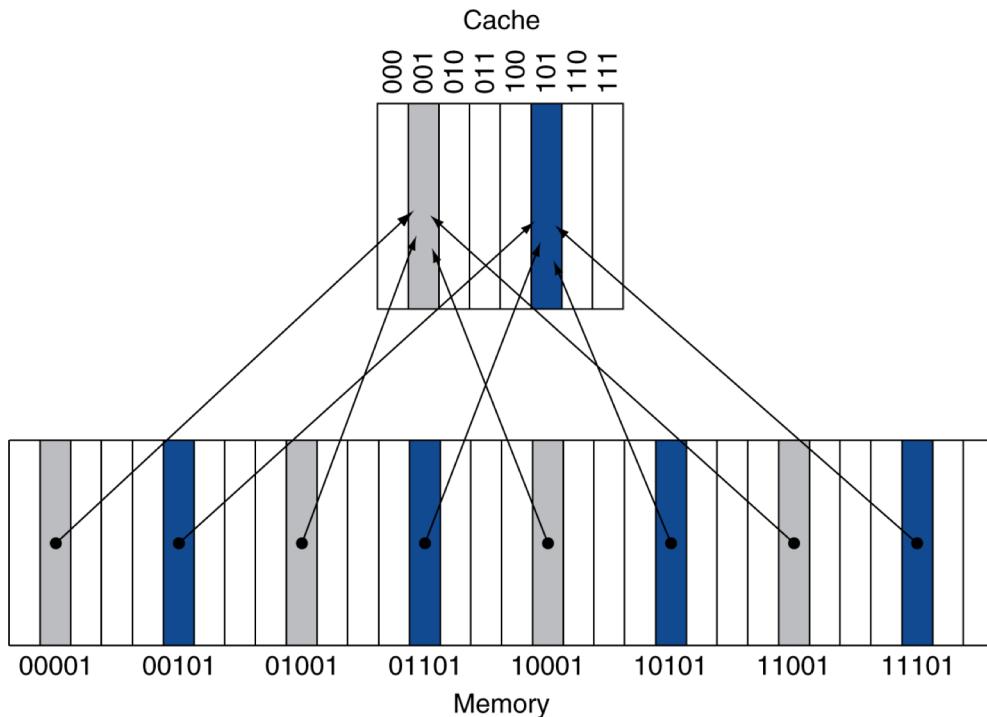
b. After the reference to X_n

- Miss: X_n is brought in from lower level
- But
 - How do we know if there is a hit or miss?
 - Where do we look?



Direct Mapped Cache

- Location of a block in cache is determined by address of the requested word
- Direct mapped cache: each memory location corresponds to one choice in cache
 - *Cache location (or cache index) = (Block address in memory) modulo (Number of blocks in cache)*
 - Direct mapped memory uses an n-to-1 mapping



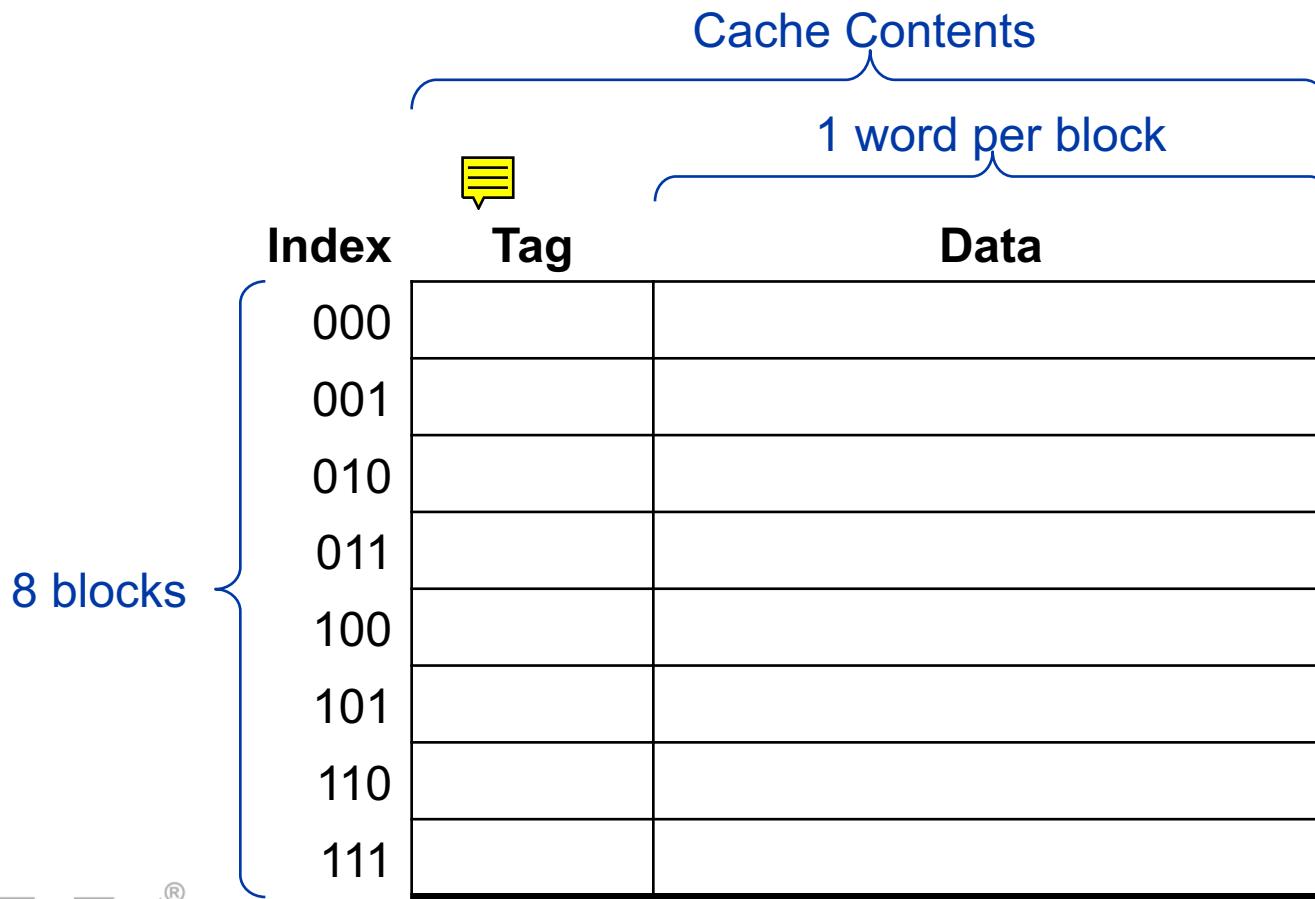
- Number of blocks in a cache is a power of 2
- Low-order bits of block address in memory = cache index

Tags in Cache

- How do we know which block of data is present in cache since it's n-to-1 mapping?
 - Store part of block address together with the data when the data is copied to higher level
 - Only need to store the high-order bits of memory block address
 - Called *tag*

Direct Mapped Cache Example

- 8-block cache
- 1 word per block (word address = block address)

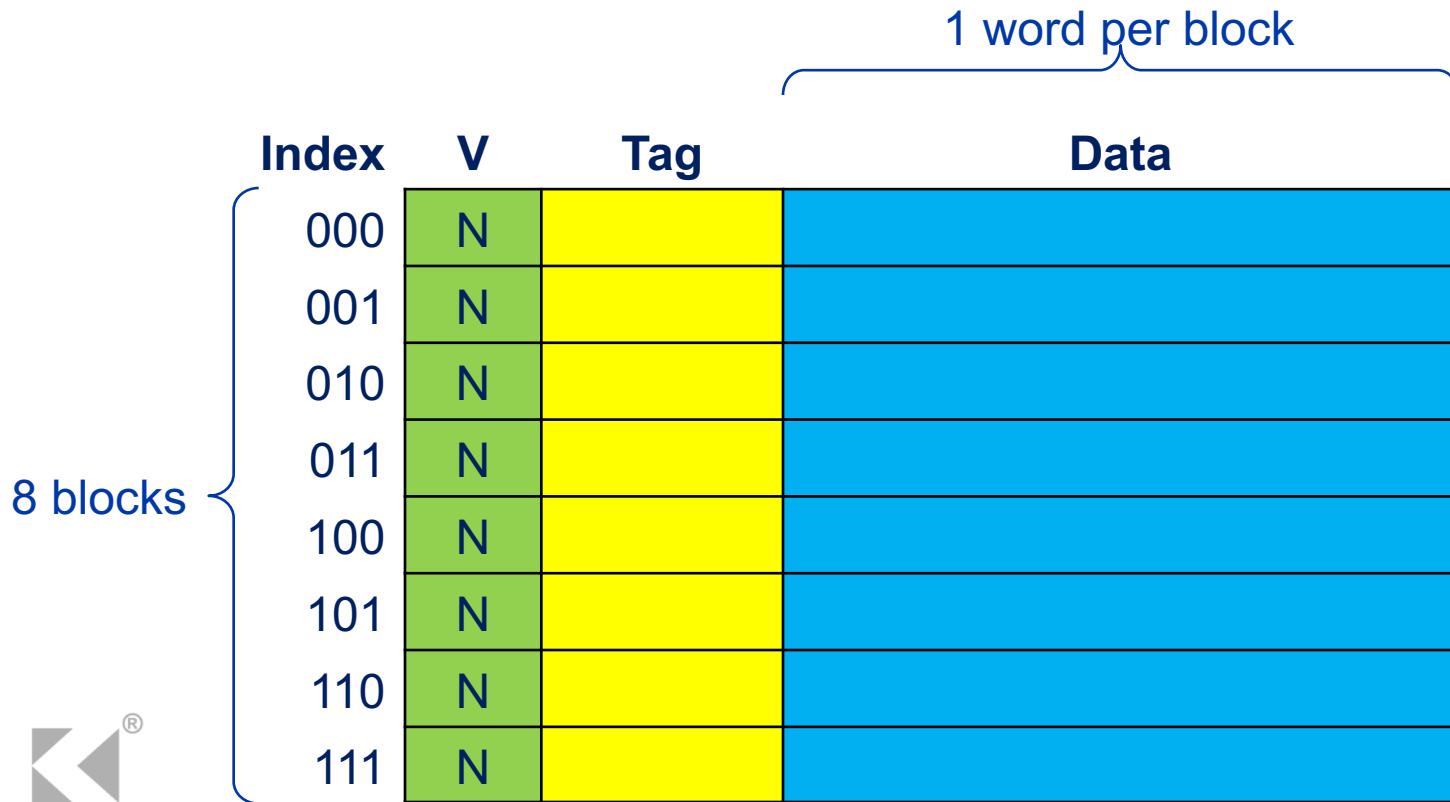


Valid Bits in Cache

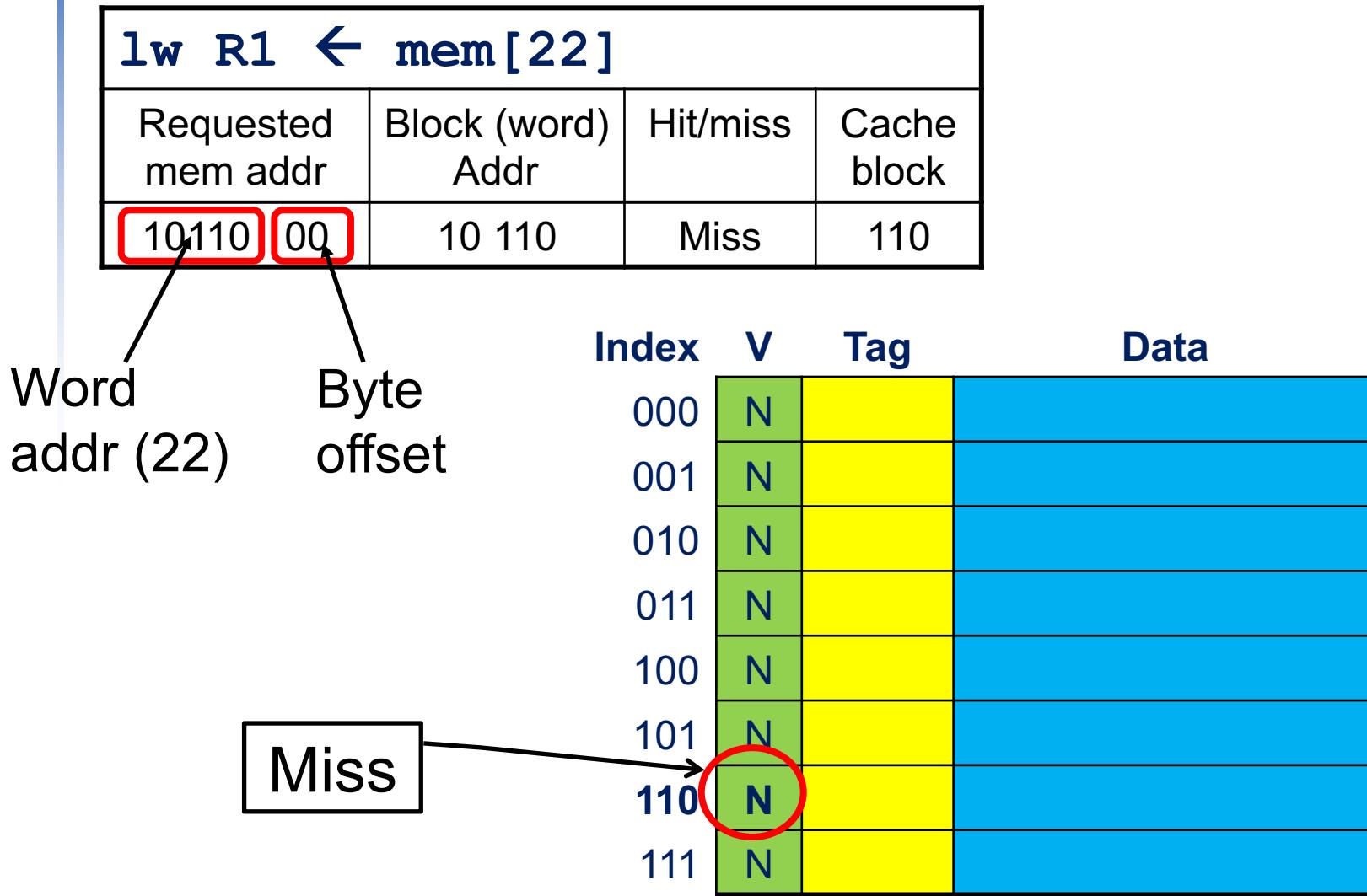
- Waste searching time if there is no valid data in a location
 - Valid bit: 1 = present, 0 = not present
 - Initially 0 (N), to improve search speed

Cache Example 1

- 8-block cache
- 1 word per block (word address = block address)
- direct mapped once know the address, know the location in the cache0
- Assuming 7-bit byte addresses



Cache Example 1 (cont.)



Cache Miss

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Using processor control unit and a cache controller
 - Freezes registers and waits for memory
 - Fetch block from next level of hierarchy

Cache Example 1 (cont.)

lw R1 ← mem[22]			
Requested mem addr	Block (word) Addr	Hit/miss	Cache block
10110 00	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

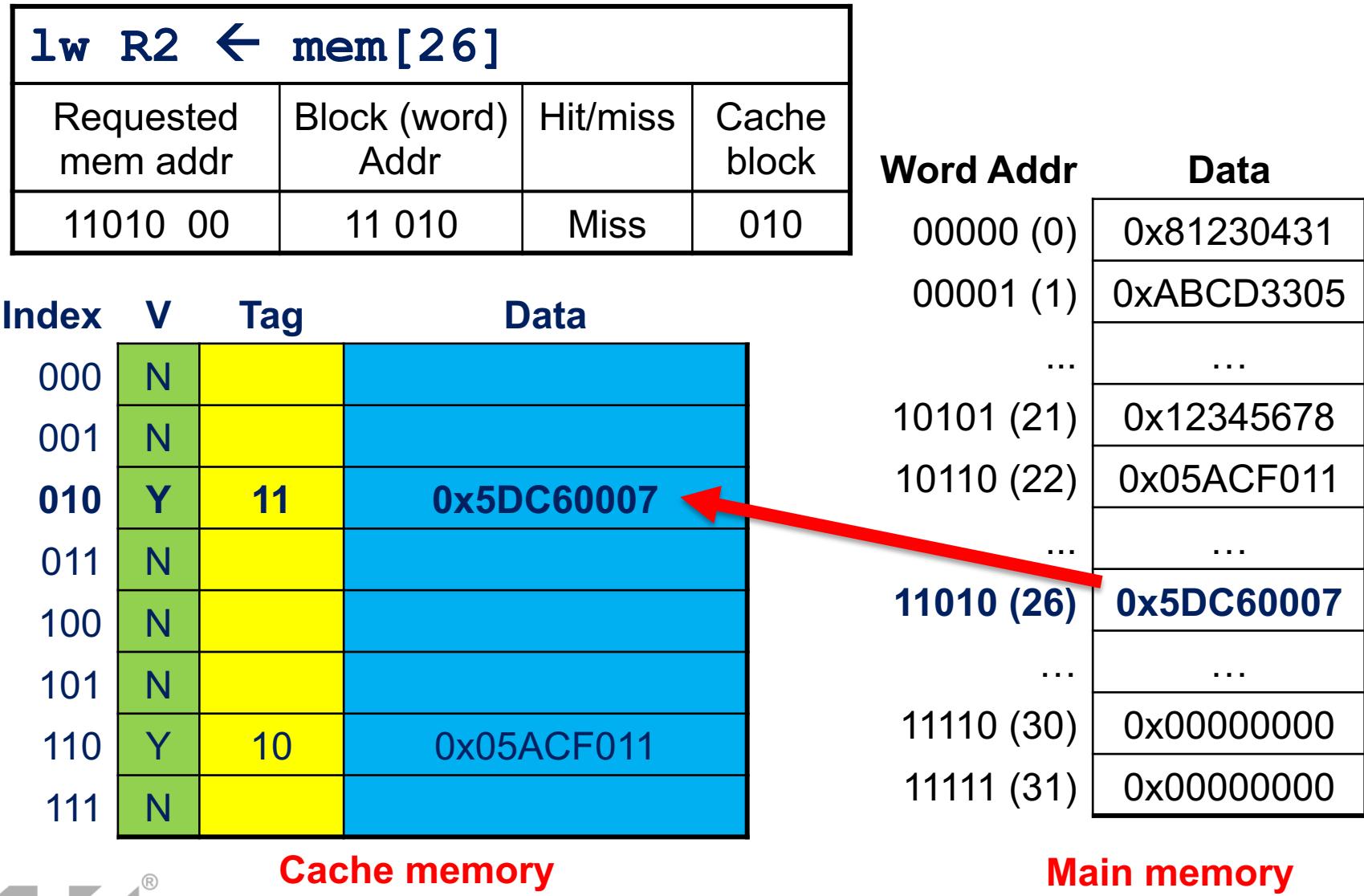


Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
...	...
10101(21)	0x12345678
10110(22)	0x05ACF011
...	...
11110(30)	0x00000000
11111(31)	0x00000000

Cache memory

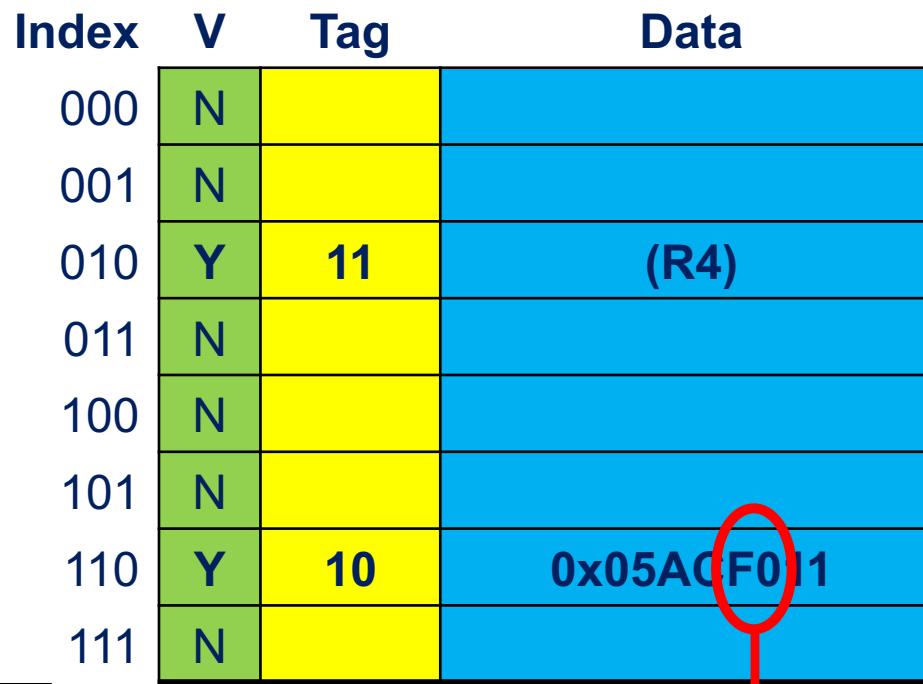
Main memory

Cache Example 1 (cont.)



Cache Example 1 (cont.)

	Requested mem addr	Block (word) Addr	Hit/miss	Cache block
lb R3 ← mem[22]byte1	10110 01	10 110	Hit	110
sw R4 → mem[26]	11010 00	11 010	Hit	010



Hit due to temporal locality

R3 FFFFFFFFFF ←

Cache Example 1 (cont.)

Requested mem Addr	Block (word) Addr	Hit/miss	Cache block
1000000(lw)	10 000	Miss	000
0001100(sw R5)	00 011	Miss	011
1000000(lw)	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	0x8765ABCD
001	N		
010	Y	11	(R4)
011	Y	00	0xFFFF0002 → (R5)
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
00010(2)	0xFFFF0001
00011(3)	0xFFFF0002
...	...
10000(16)	0x8765ABCD
...	...
10101(21)	0x12345678
10110(22)	0x05ACF011
...	...
11010(26)	0x5DC60007
...	...
11110(30)	0x00000000
11111(31)	0x00000000

Cache Example 1 (cont.)

Requested mem Addr	Block (word) Addr	Hit/miss	Cache block
1001000(lw)	10 010	Miss	010

Currently occupied by Mem[26]

Index	V	Tag	Data
000	Y	10	0x8765ABCD
001	N		
010	Y	10	(R4)→0x0000FF00
011	Y	00	0xFFFF0002→(R5)
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
00010(2)	0xFFFF0001
00011(3)	0xFFFF0002
...	...
10000(16)	0x8765ABCD
10010(17)	0x0000FF00
10101(18)	0x12345678
...	...
11010(26)	0x5DC60007
...	...
11110(30)	0x00000000
11111(31)	0x00000000



Block Size Considerations

- ***Larger blocks should reduce miss rate***
 - Due to spatial locality
- But in a fixed-sized cache
 - Larger blocks \Rightarrow fewer of them
 - More competition \Rightarrow increased miss rate
- Larger miss penalty
 - Primarily result of longer time to fetch block
 - Latency to first word
 - Transfer time for the rest of the block
 - Can override benefit of reduced miss rate
 - Early restart and critical-word-first can help



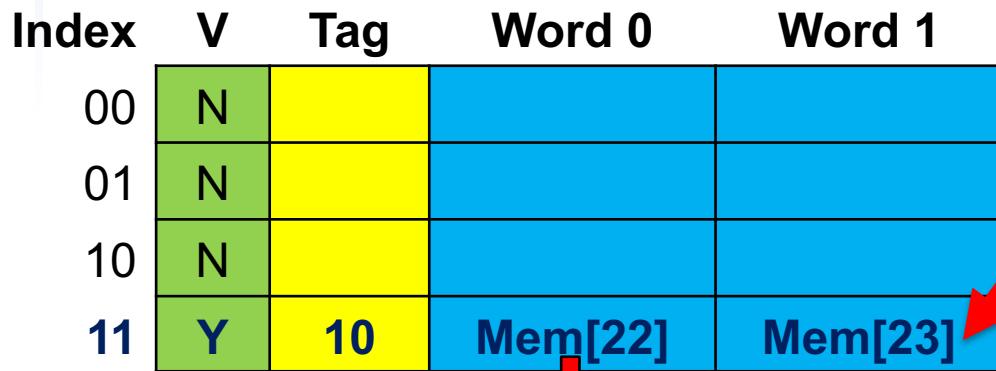
Cache Example 2

- 4-block cache
- 2 words per block
- direct mapped
- Assuming 7-bit byte addresses



Cache Example 2 (cont.)

lw R1 ← mem[22]			
Requested mem addr	word addr	Hit/miss	Cache block
10110 00	10 11 0	Miss	11



R1



Word Addr	Data
00000 (0)	0x81230431
00001 (1)	0xABCD3305
00010 (2)	0xFFFF0001
00011 (3)	0xFFFF0002
...	...
10110 (22)	0x05AC0011
10111 (23)	0x0000FFEE
...	...
10110 (26)	0x1153A4D6
10111 (27)	0xAABB1234
...	...
11110 (30)	0x00000000
11111 (31)	0x00000000



Cache Example 2 (cont.)

sw \$0 → mem[23]			
lw R2 ← mem[27]			
Requested mem addr	word addr	Hit/miss	Cache block
10111 00	10 11 1	Hit	11
11011 00	11 01 1	Miss	01

Hit due to
spatial locality

Index	V	Tag	Word 0	Word 1
00	N			
01	Y	11	Mem[26]	Mem[27]
10	N			
11	Y	10	Mem[22]	0

Word Addr	Data
00000 (0)	0x81230431
00001 (1)	0xABCD3305
00010 (2)	0xFFFF0001
00011 (3)	0xFFFF0002
...	...
10110 (22)	0x05AC0011
10111 (23)	0x0000FFEE
...	...
11010 (26)	0x1153A4D6
11011 (27)	0xAABB1234
...	...
11110 (30)	0x00000000
11111 (31)	0x00000000



Cache Example 2 (cont.)

lw R3 ← mem[6]			
Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	00 11 0	miss	11

Index	V	Tag	Word 0	Word 1
00	N			
01	Y	11	Mem[26]	Mem[27]
10	N			
11	Y	10	Mem[6]	Mem[7]

N-to-1 mapping causes competition,
original block was replaced

Word Addr	Data
00000 (0)	0x81230431
...	...
00110 (6)	0xFFFF0126
00111 (7)	0xFFFF0127
...	...
10110 (22)	0x05AC0011
10111 (23)	0x0000FFEE
...	...
11010 (26)	0x1153A4D6
11011 (27)	0xAABB1234
...	...
11110 (30)	0x00000000
11111 (31)	0x00000000



Cache Example 2 (cont.)

lw R4 ← mem [22]			
Requested mem addr	Word addr	Hit/miss	Cache block
10110 00	10 11 0	miss	11

Index	V	Tag	Word 0	Word 1
00	N			
01	Y	11	Mem[26]	Mem[27]
10	N			
11	Y	10	Mem[22]	Mem[23]

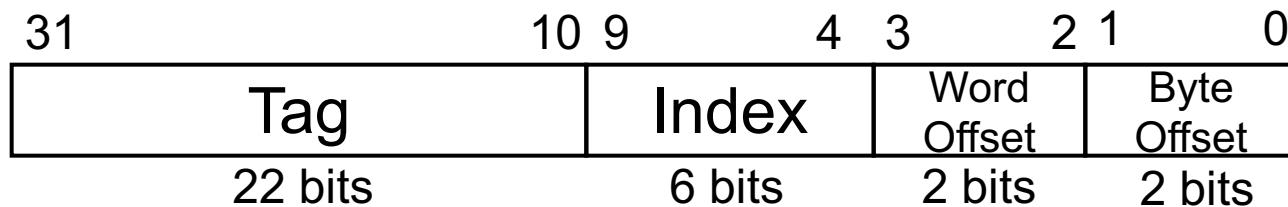
Replaced again

Word Addr	Data
00000 (0)	0x81230431
...	...
00110 (6)	0xFFFF0126
00111 (7)	0xFFFF0127
...	...
10110 (22)	0x05AC0011
10111 (23)	0x0000FFEE
...	...
11010 (26)	0x1153A4D6
11011 (27)	0xAABB1234
...	...
11110 (30)	0x00000000
11111 (31)	0x00000000



Example 3: Larger Block Size

- 64 blocks, 4 words/block
 - What cache block number does byte address 1200 map to?
 - Word number = $1200/4 = 300$
 - Block (address) number = $300/4 = 75$
- Block index in cache = $75 \text{ modulo } 64 = 11$



Cache Size in Bits

- Given
 - 32-bit byte address
 - 2^n blocks in cache
 - 2^m words per block, 2^{m+2} bytes
- Size of tag field = $32 - (n + m + 2)$
 - n bits to index blocks in cache
 - m bits used to select words in a block
 - 2 bits used to select the 4 bytes in a word
 - Tag field decreases when n and m increase
- Cache size = $2^n \times (\text{block size} + \text{tag size} + \text{valid field size})$

Class Exercise

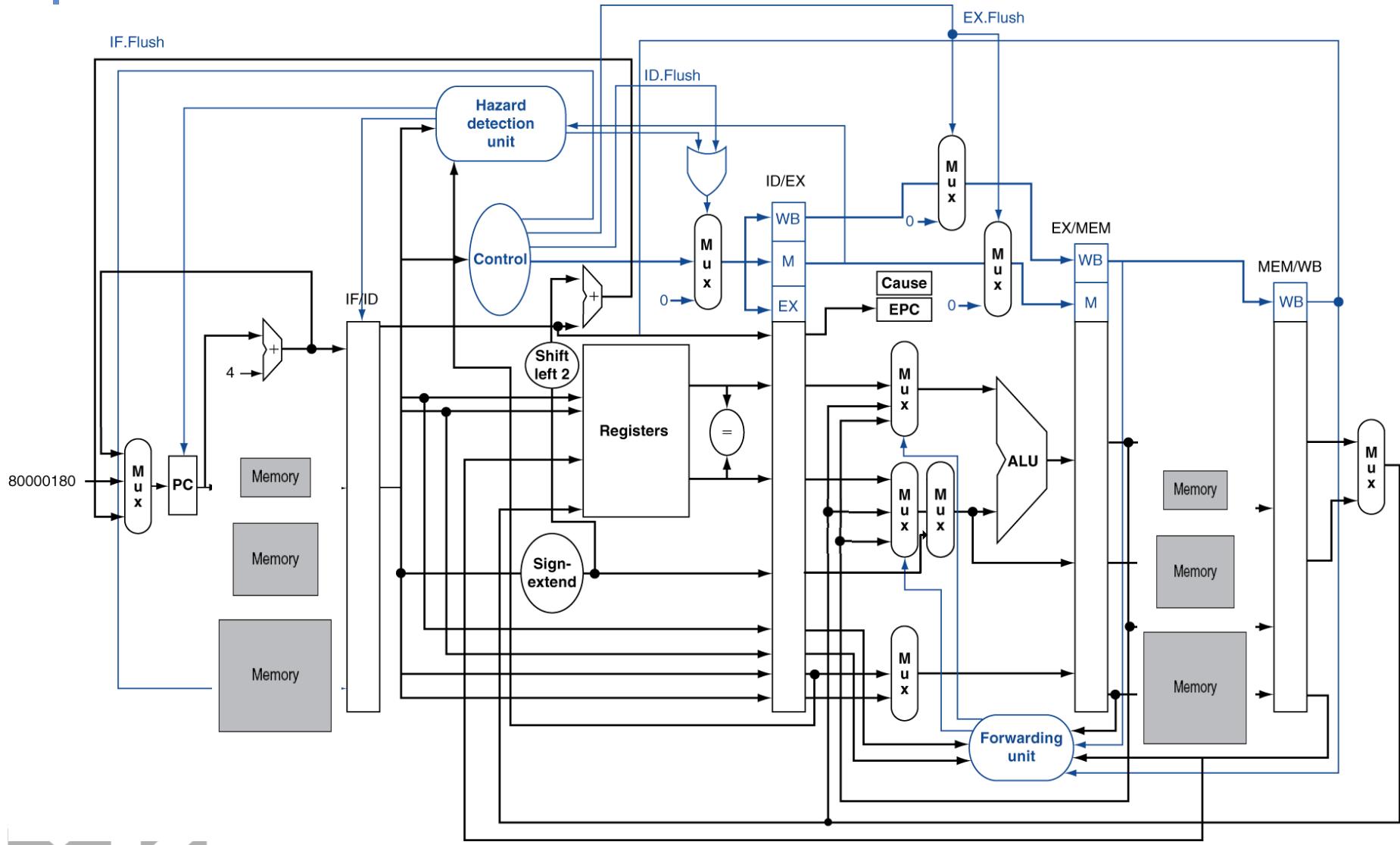
- Given

- 2K blocks in cache
 - 8 words in each block
 - 32-bit byte address 0x810023FE requested by CPU

000000000000

- Show address of the target cache block and organization of the entire cache

MIPS Pipeline Architecture



Miss in Instruction Cache

1. Send original PC ($PC - 4$) to memory
 - From the Adder
2. Read main (lower level) memory and wait for data
3. Write data into cache data field from main memory, write upper bits of address into cache tag field, set valid field
4. Restart the missing instruction

Miss in Data Cache – Reads

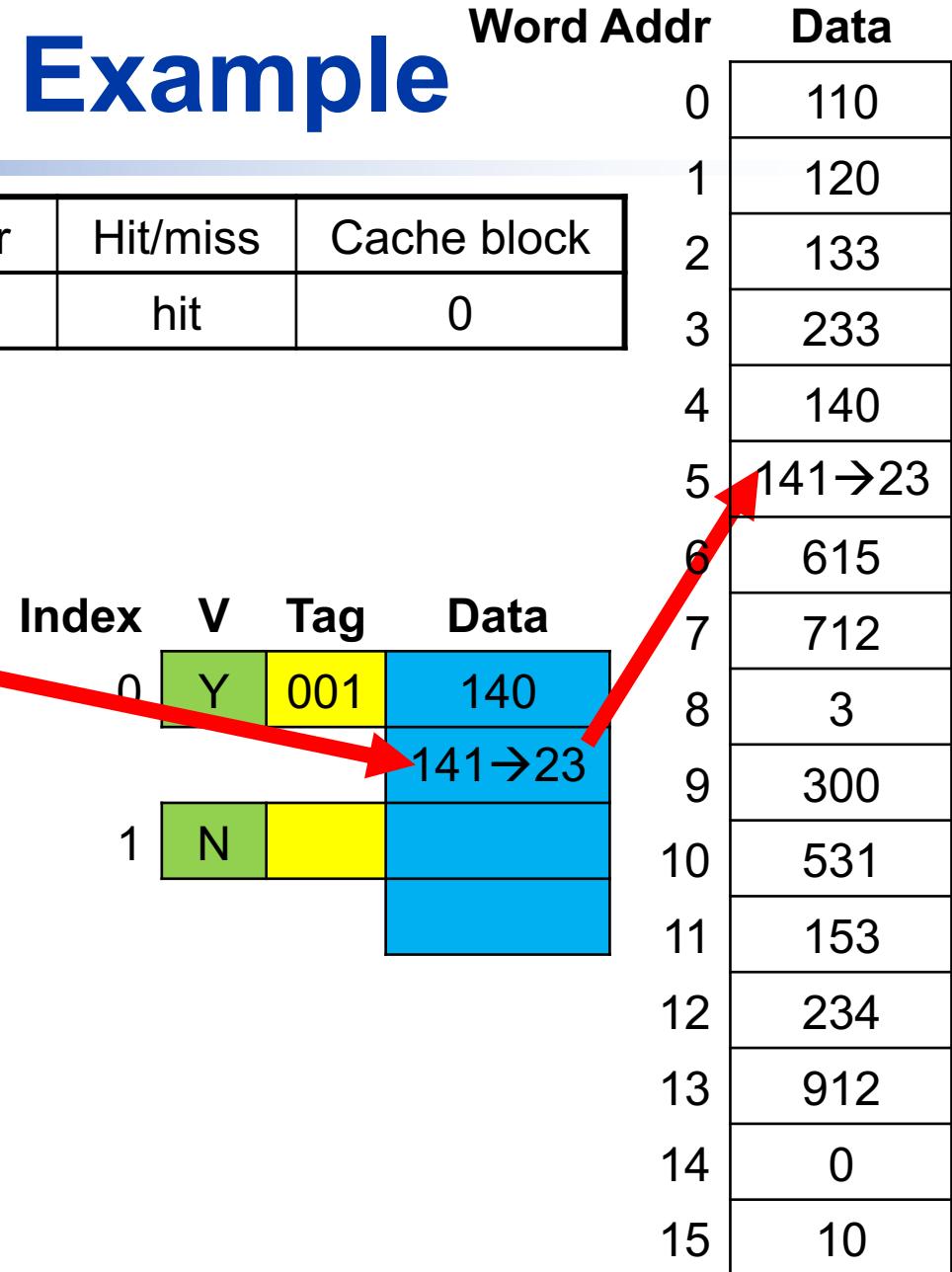
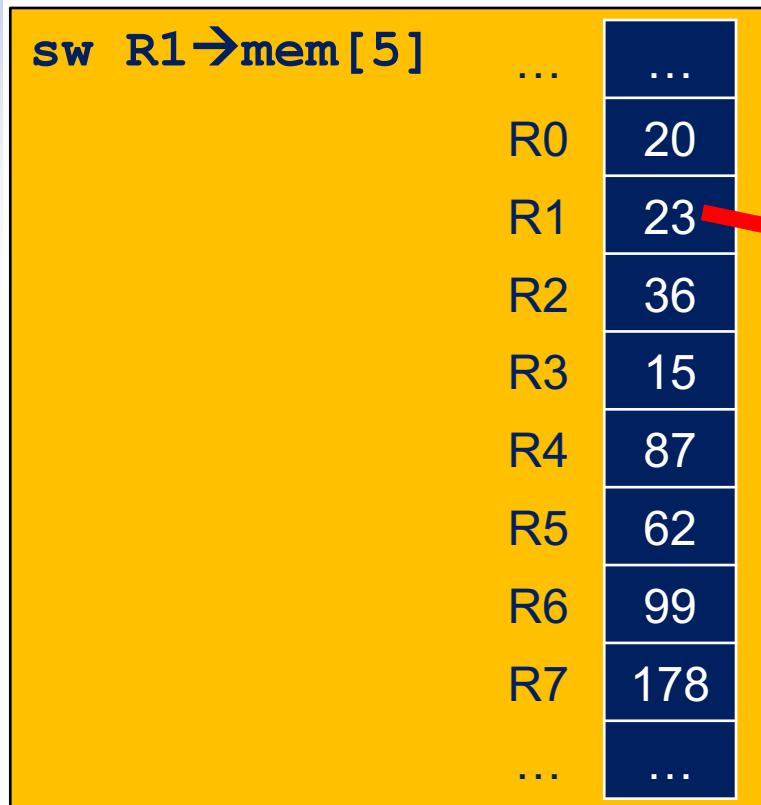
1. Hold the pipeline
2. Read main (lower level) memory and wait for data
3. Write data into cache data field from main memory, write upper bits of address into cache tag field, set valid field
4. Read cache again, proceed

Handling Data Writes – Write Through

- On data-write (e.g. sw) hit, could just update the block in cache
 - But then cache and memory would be inconsistent
- Write through: also update the word in memory

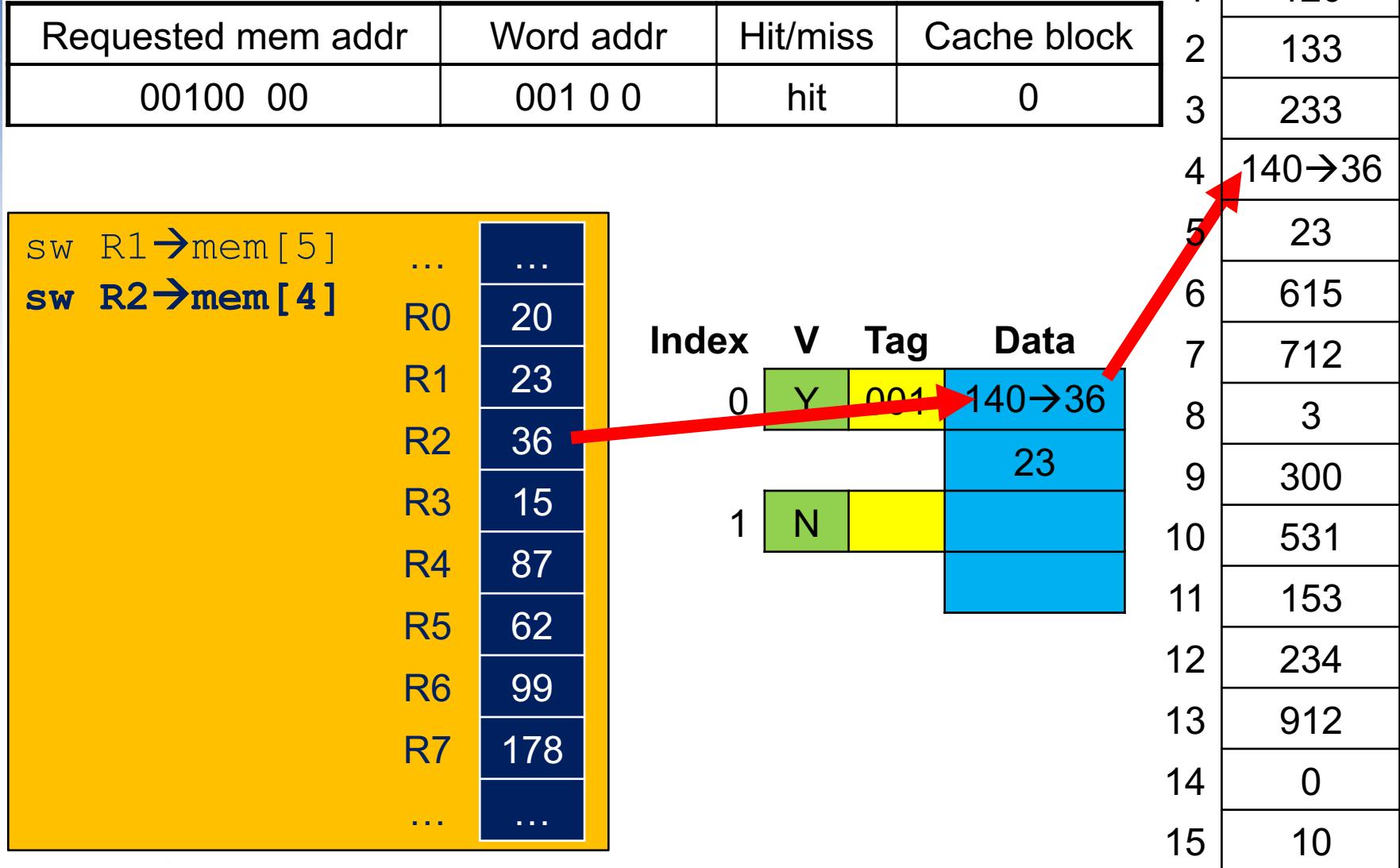
Write Through Example

Requested mem addr	Word addr	Hit/miss	Cache block
00101 00	001 0 1	hit	0



CPU

Write Through Example



CPU

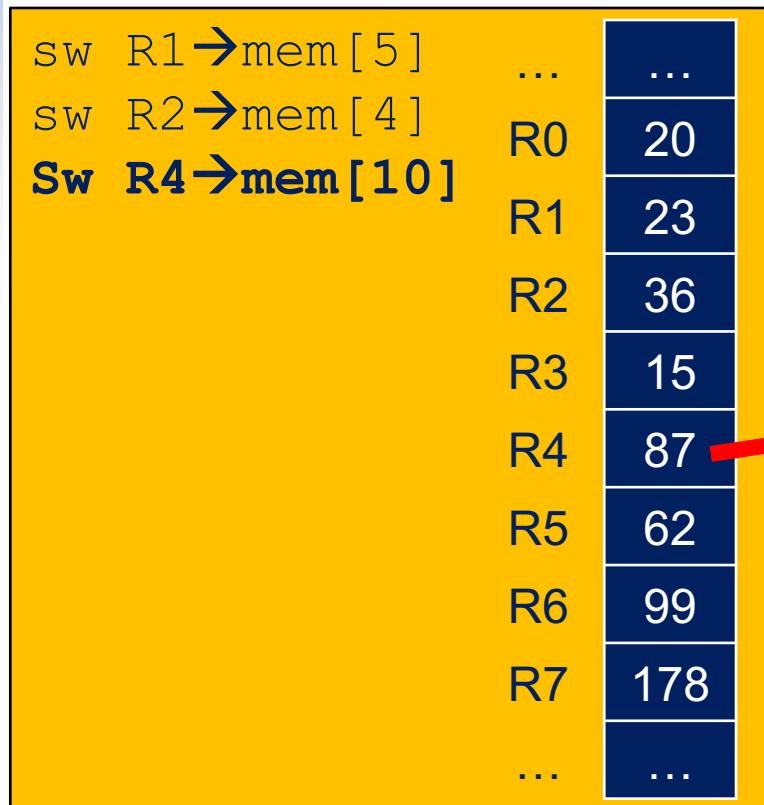
Handling Data Writes – Write Through

- But makes writes take longer time
 - Must wait till the update finishes
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = base CPI + write time (cycles) per instruction
 - Effective CPI = $1 + 0.1 \times 100 = 11$
- Even worse for write miss
 - Detect a miss on target address
 - Fetch the block from main memory to cache
 - Overwrite the word in cache
 - Write the block back to main memory



Write Through Example

Requested mem addr	Word addr	Hit/miss	Cache block
01010 00	010 1 0	miss	1



Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	531
11	153
12	234
13	912
14	0
15	10

Index	V	Tag	Data
0	Y	001	36
1	N		87

Miss



CPU

Write Through Example

Requested mem addr	Word addr	Hit/miss	Cache block
01010 00	010 1 0	miss	1

sw R1 → mem[5]
 sw R2 → mem[4]
Sw R4 → mem[10]

...	...
R0	20
R1	23
R2	36
R3	15
R4	87
R5	62
R6	99
R7	178
...	...

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	531
11	135
12	234
13	912
14	0
15	10

Index V Tag Data

0	Y	001	36
1	Y	010	531
			135

Fetch block



CPU

Write Through Example

Requested mem addr	Word addr	Hit/miss	Cache block
01010 00	010 1 0	hit	1

sw R1 → mem[5]
sw R2 → mem[4]	R0	20
Sw R4 → mem[10]	R1	23
	R2	36
	R3	15
	R4	87
	R5	62
	R6	99
	R7	178

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	531 → 87
11	135
12	234
13	912
14	0
15	10

Index V Tag Data

0	Y	001	36
1	Y	010	135

Write Through



CPU

Write Buffer

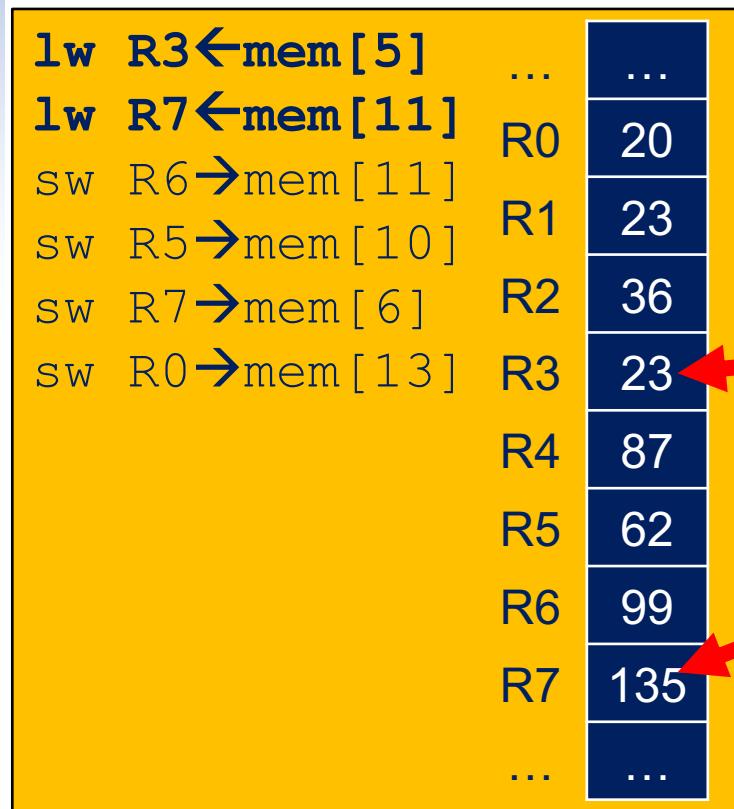
- Solution to time consuming write through technique (for both hit and miss)
 - Buffer stores data to be written to memory
 - May have one or more entries
 - CPU proceeds to next step, while letting buffer to complete write through
 - Frees buffer when completing write to memory
 - CPU stalls if buffer is full

Handling Data Writes – Write Back

- Alternative of write through: On data-write hit, just update the block in cache
 - CPU keeps track of whether each block is *dirty* (updated with new values)
- Write a block back to memory
 - Only when a dirty block has to be replaced (on miss)
 - More complex than write through

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00101 00	001 0 1	hit	0
01011 00	010 1 1	hit	1



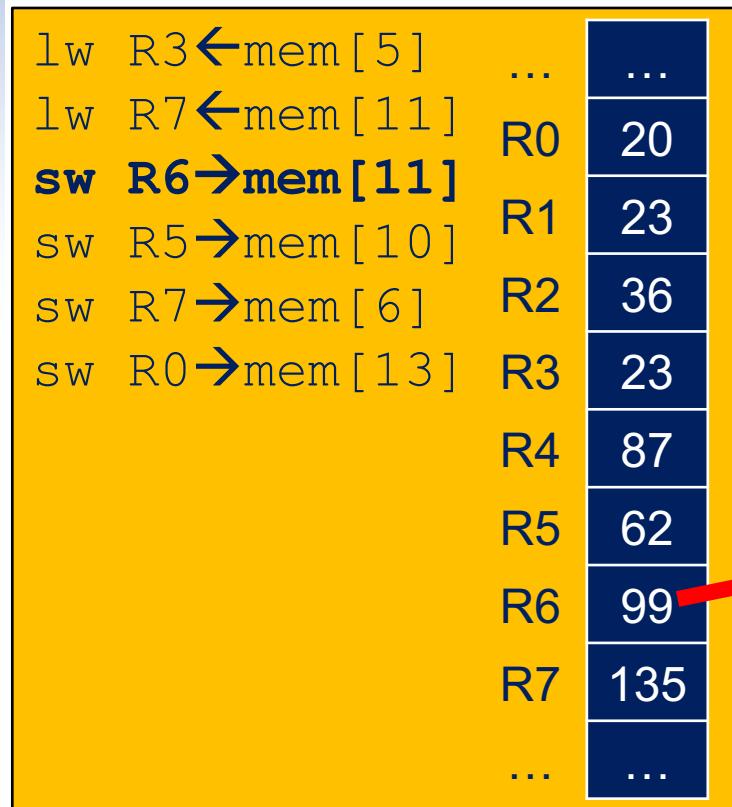
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	87
11	135
12	234
13	912
14	0
15	10



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
01011 00	010 1 1	hit	1



Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	87
11	135
12	234
13	912
14	0
15	10

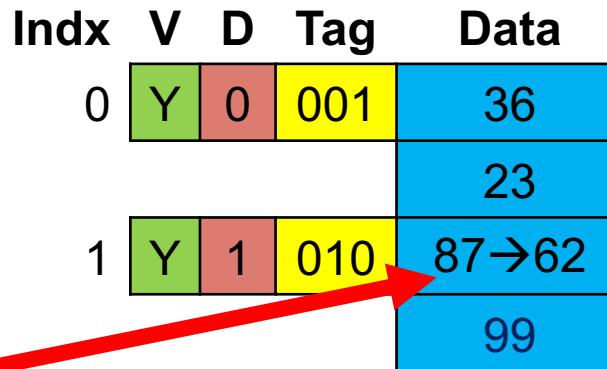
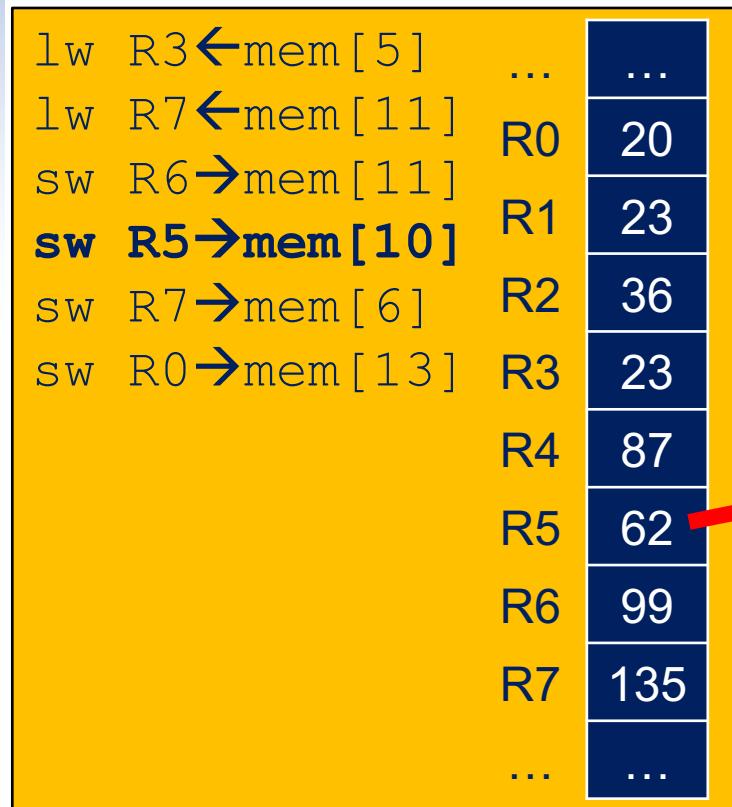
Write cache, Dirty



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
01010 00	010 1 0	hit	1



Write cache, Dirty



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	00110	miss	1

lw R3 ← mem[5]
lw R7 ← mem[11]	R0	20
sw R6 → mem[11]	R1	23
sw R5 → mem[10]	R2	36
sw R7 → mem[6]	R3	23
sw R0 → mem[13]	R4	87
	R5	62
	R6	99
	R7	135

Miss match

Idx	V	D	Tag	Data
0	Y	0	001	36
1	Y	1	010	62

Miss

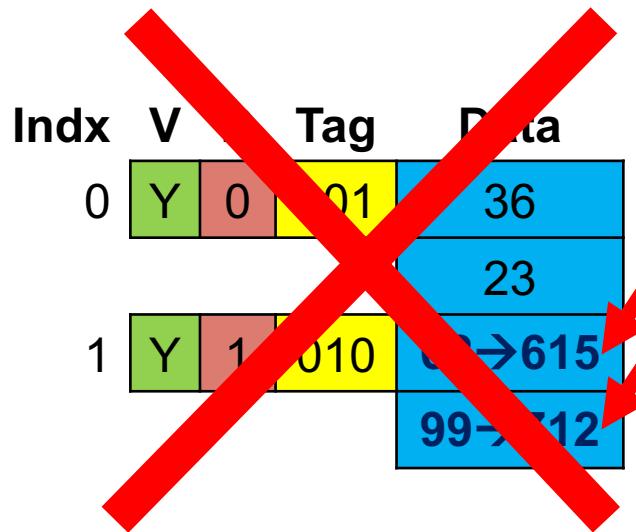
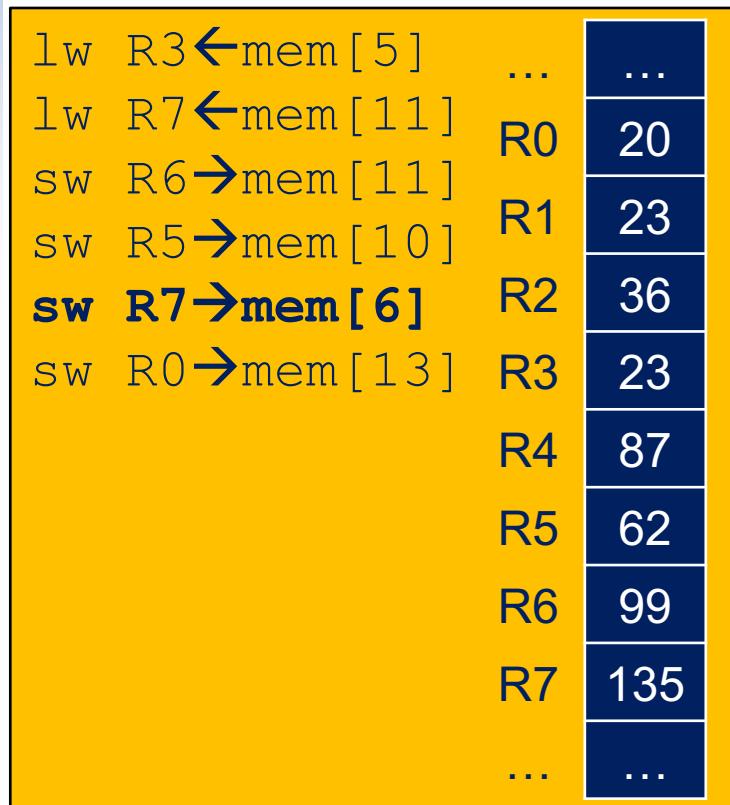
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	87
11	135
12	234
13	912
14	0
15	10



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	001 1 0	miss	1



Replace, but overwritten



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	001 1 0	miss	1

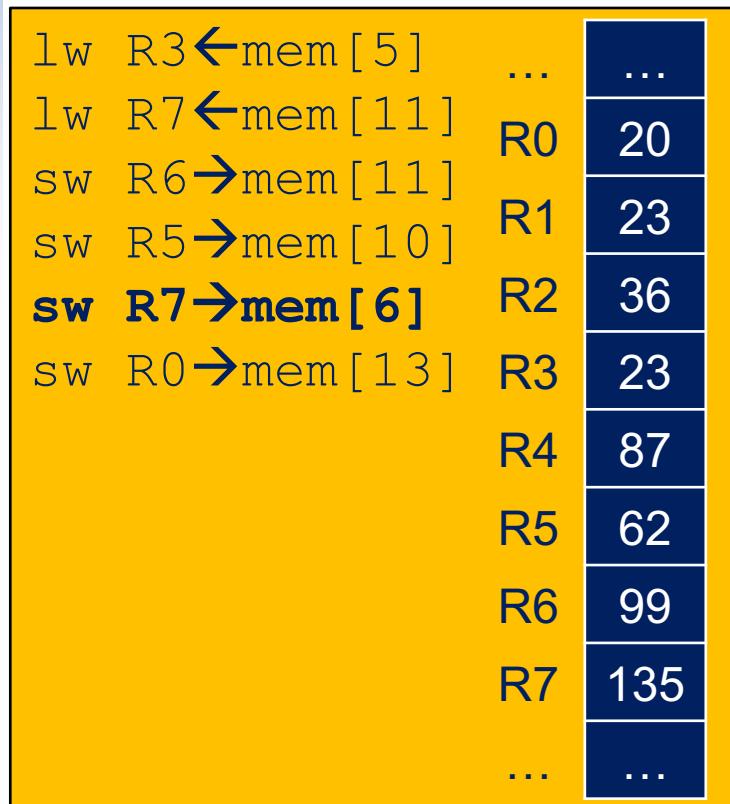
lw R3 ← mem[5]
lw R7 ← mem[11]	R0	20
sw R6 → mem[11]	R1	23
sw R5 → mem[10]	R2	36
sw R7 → mem[6]	R3	23
sw R0 → mem[13]	R4	87
	R5	62
	R6	99
	R7	135



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	001 1 0	miss	1



Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

Cache State:

Idx	V	D	Tag	Data
0	Y	0	001	36
1	Y	0	001	23

A red circle highlights the dirty bit (D) in row 1. Red arrows point from the circled D to the text "Replace now, Reset dirty".

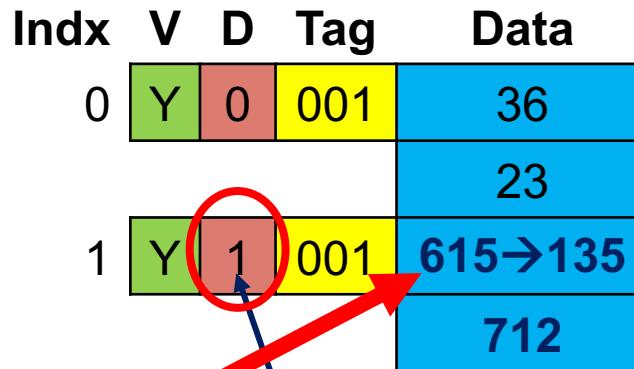
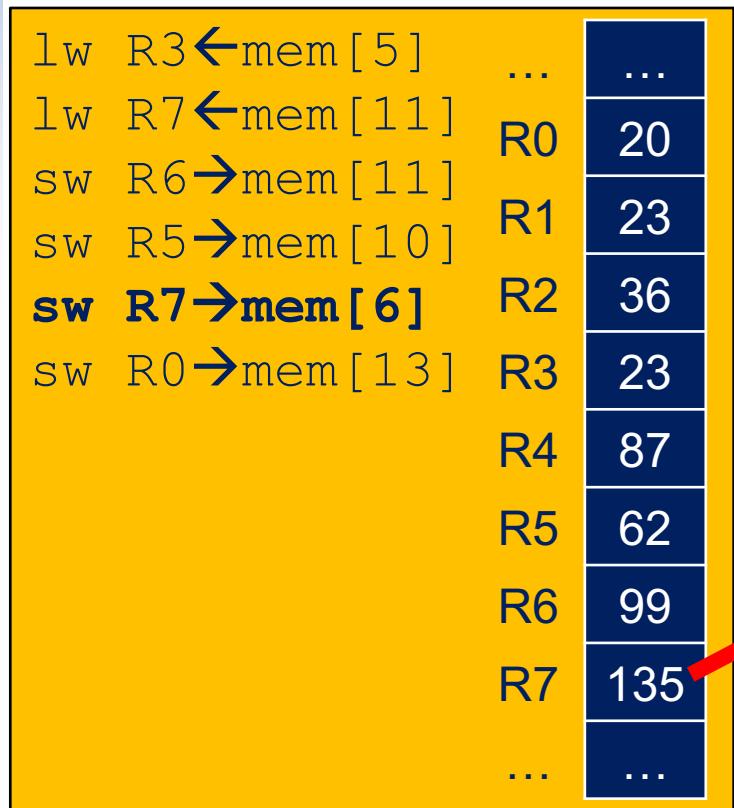
Text: Replace now, Reset dirty



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	001 1 0	hit	1



Write,
set dirty

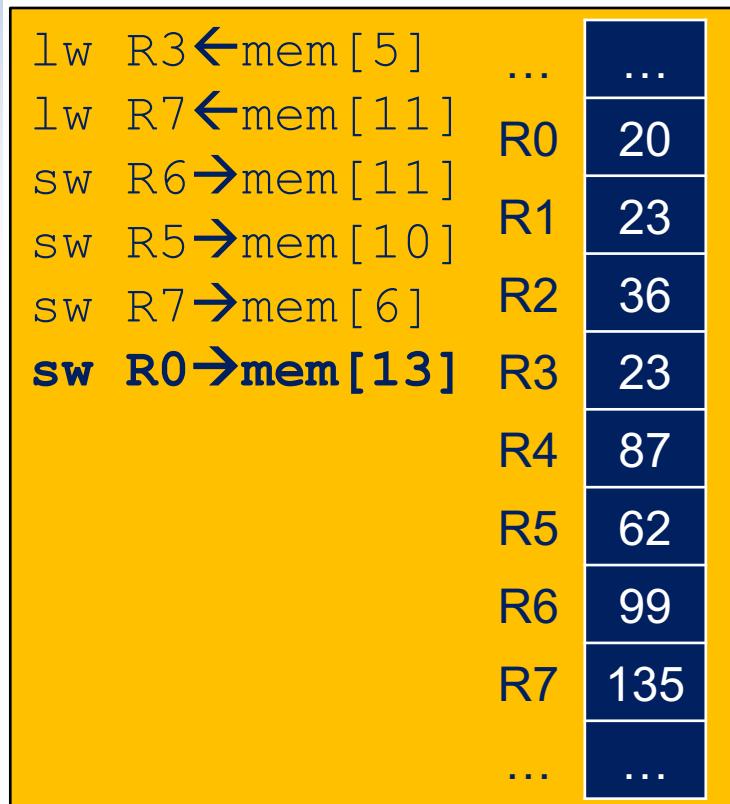
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
01101 00	0110 1	miss	0



Miss match

Idx	V	D	Tag	Data
0	Y	0	001	36
1	Y	1	001	135

Miss

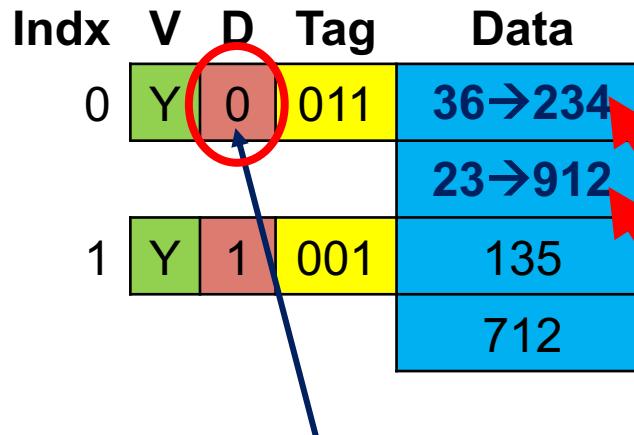
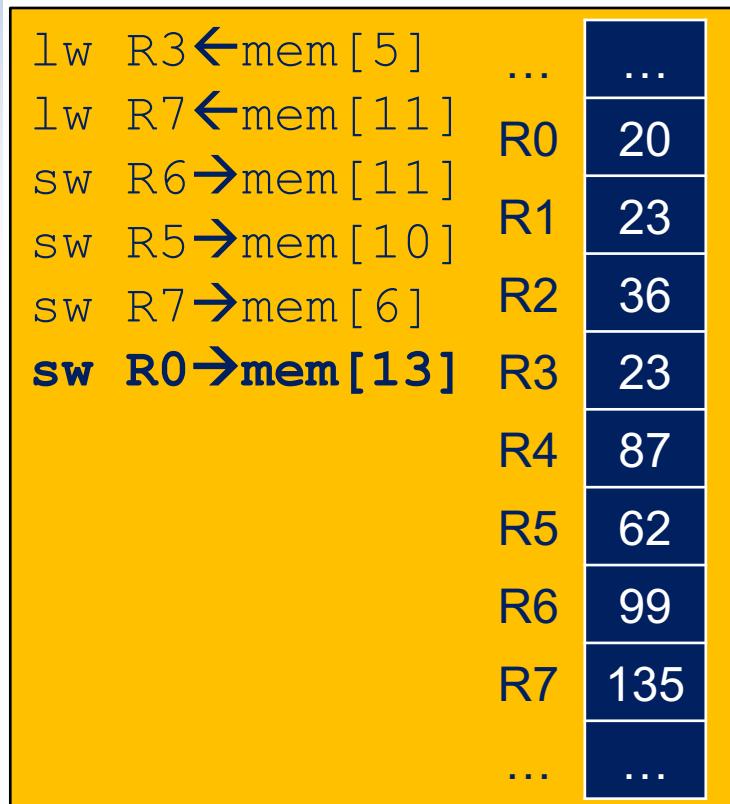
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
01101 00	011 0 1	miss	0



Not Dirty,
Replace directly

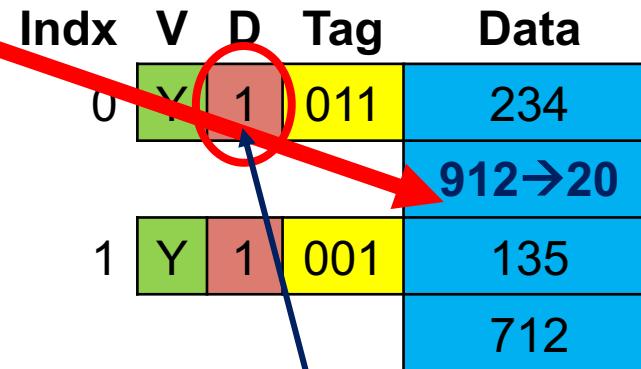
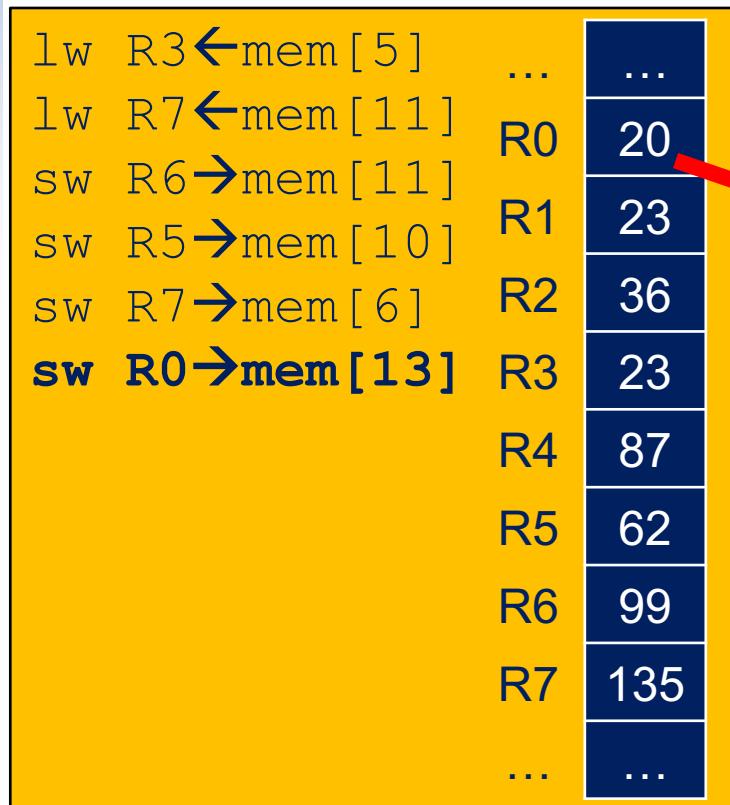
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
01101 00	011 0 1	hit	0



Write,
Set dirty



CPU

Write Through/Back Sequences

- Write back sequence
 - Two steps:
 - 1. check match,
 - 2. write data
 - Otherwise, will destroy the mismatch block, and there is no backup copy
 - May use write buffer
 - Writing buffer and checking match simultaneously

Write Through/Back Sequences

- Write through sequence
 - Write data
 - Check for match
 - Mismatch doesn't matter
 - Because the mismatch block to be replaced anyway
 - For hit, saves a step, less time for write through

make the hit time faster

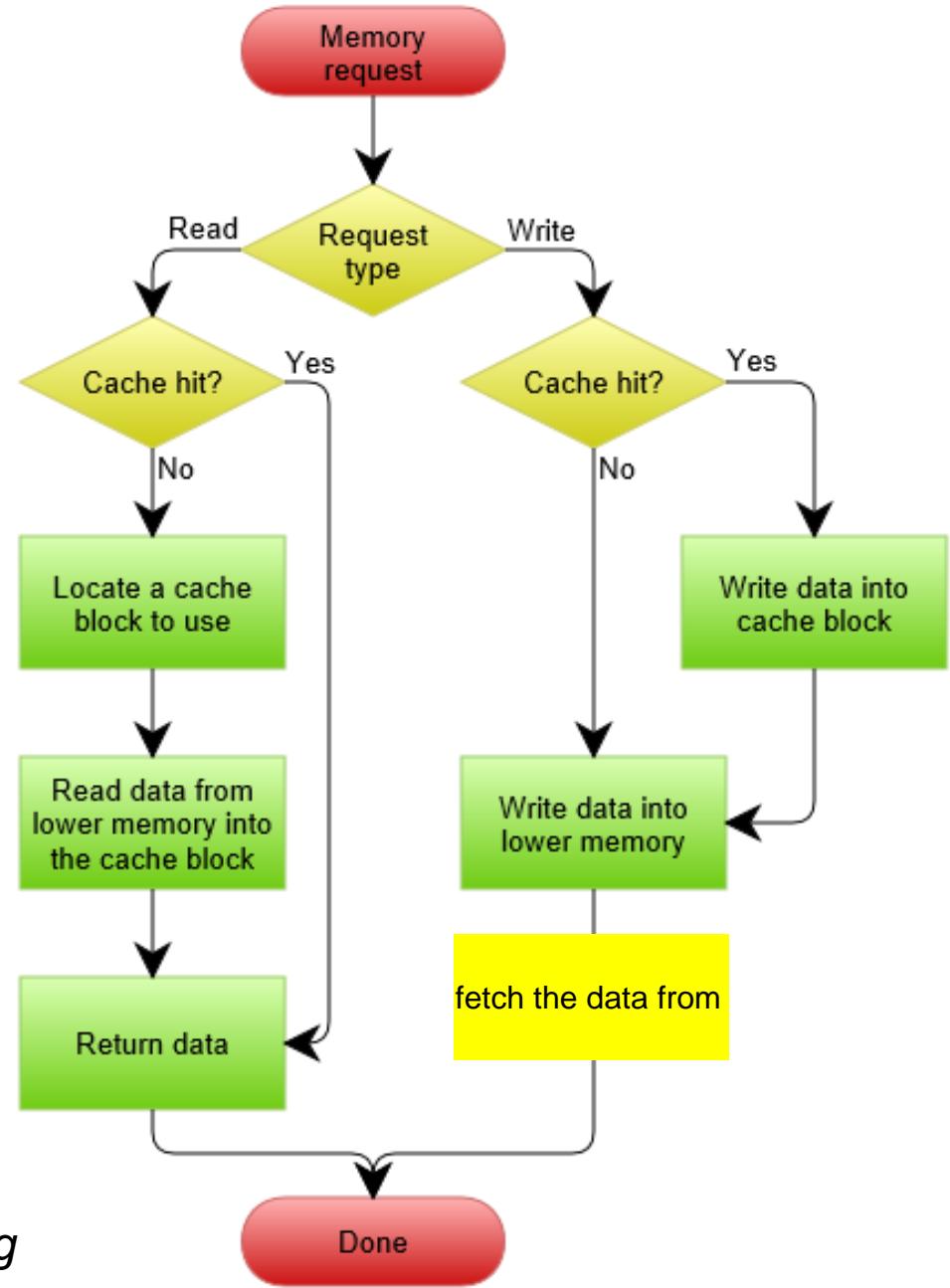


Write Allocation on Miss

- Ways of cache handlings for write-through
 - Write Allocate
 - Allocate cache block on miss by fetching corresponding memory block
 - Update cache block
 - Update memory block
 - No Write Allocate
 - Write around: write directly to memory
 - Then fetch from memory to cache
- For write-back
 - Usually fetch the block (write allocate)



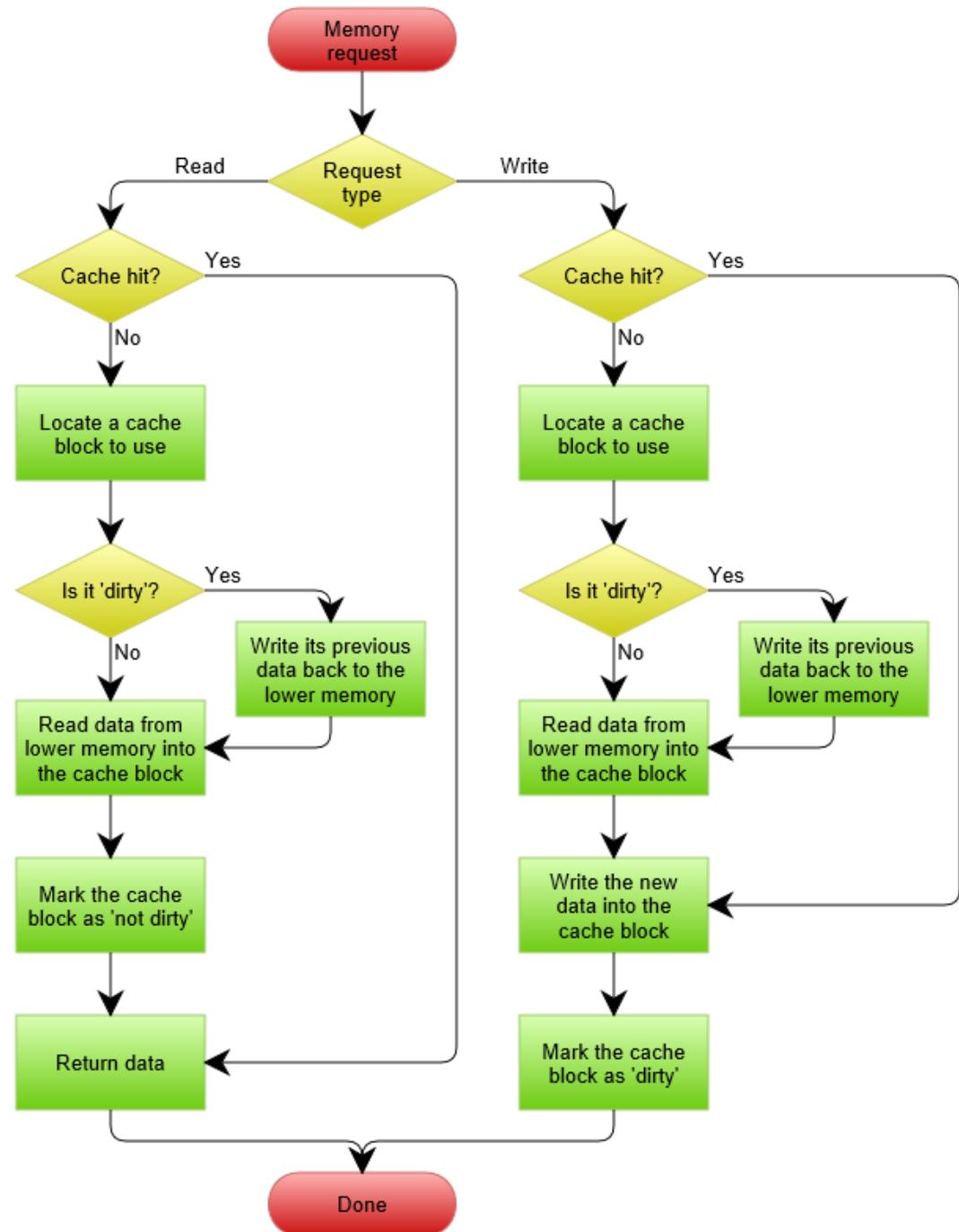
Write Through with no Write Allocation



Source: Wikipedia.org



Write Back with Write Allocation



Source: Wikipedia.org

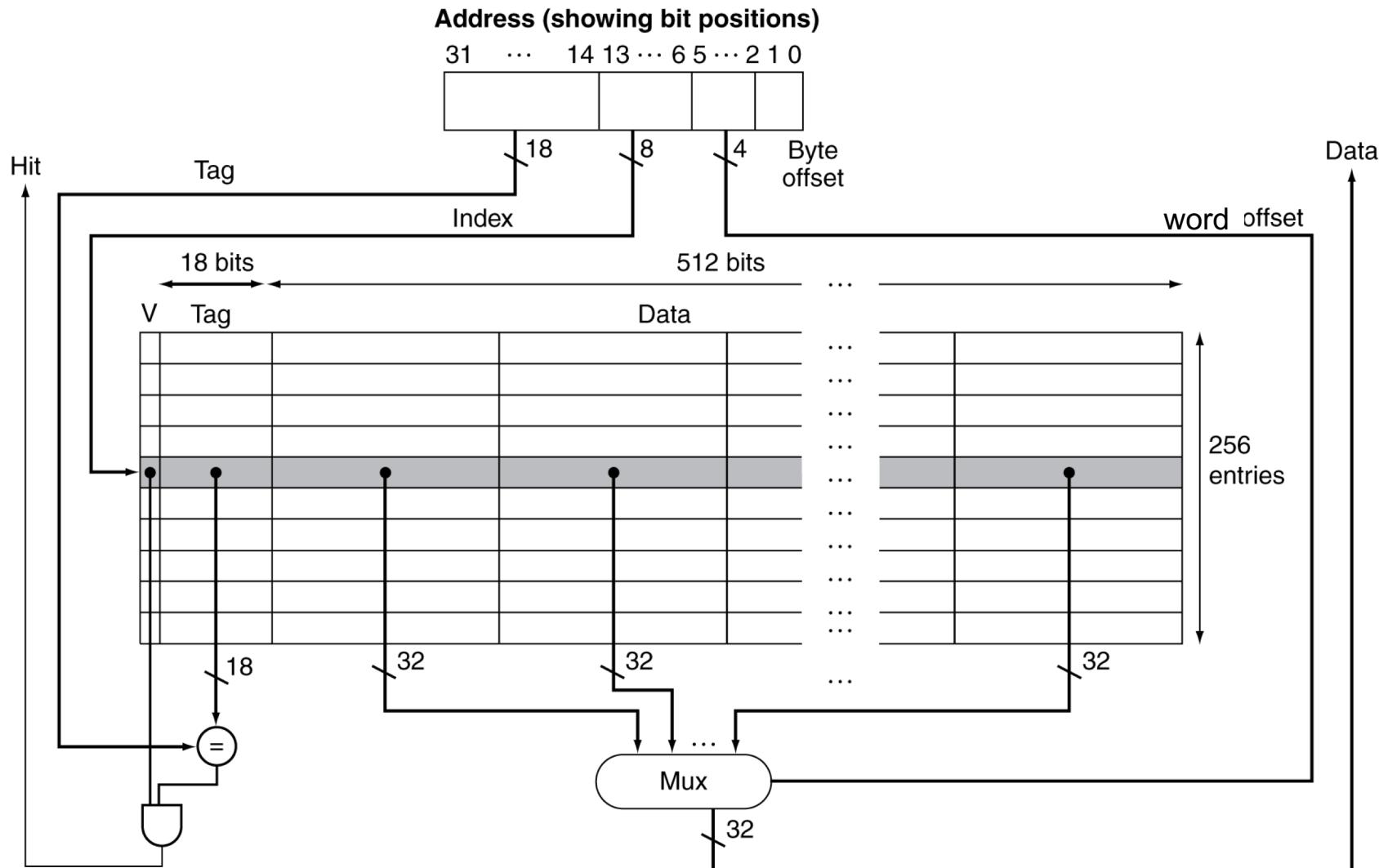


Example: Intrinsity FastMATH

- Intrinsity
 - Fabless microprocessor company
 - Acquired by Apple in 2010
- FastMATH – Embedded MIPS processor
 - 12-stage pipeline
 - Instruction and data access on each cycle
 - Split cache: separate I-cache and D-cache
 - Each 16KB: 256 blocks × 16 words/block
 - D-cache: write-through or write-back
- SPEC2000 miss rates
 - I-cache: 0.4%
 - D-cache: 11.4%



Example: Intrinsity FastMATH



Reducing Miss Penalty

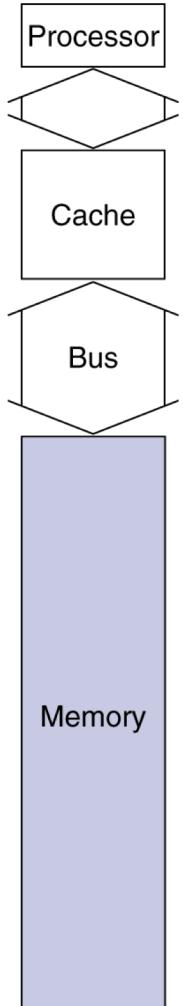
- Early restart
 - Restart execution as soon as the requested word if available, instead of waiting for the entire block
 - More effective for instruction memory because instructions are accessed sequentially
- Critical word first
 - Requires specially organized memory
 - Transfer the requested words first

Reducing Miss Penalty by Main Memory organization

- Use DRAMs for main memory
 - Fixed width (e.g., 1 word)
 - Connected by fixed-width bus
 - Bus clock is typically slower than CPU clock
- Example cache block read
 - 1 bus cycle for address transfer
 - 15 bus cycles per DRAM access
 - 1 bus cycle per data transfer

Reducing Miss Penalty

by Increasing
Memory
Bandwidth



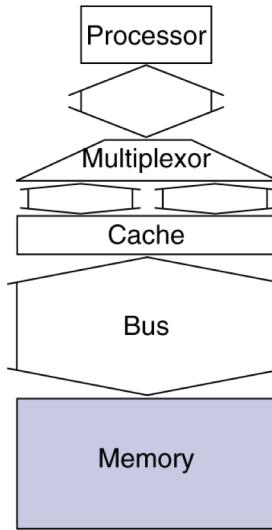
a. One-word-wide
memory organization

- For 4-word block, 1-word-wide DRAM
 - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle



Reducing Miss Penalty

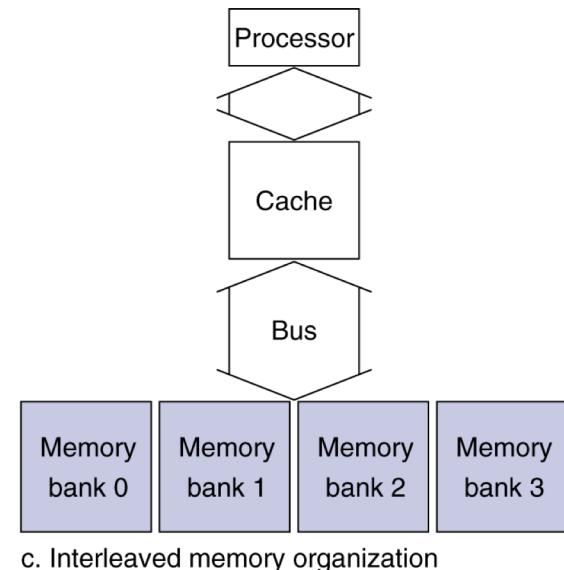
by Increasing
Memory
Bandwidth



b. Wider memory organization

- 2-word wide memory

- Miss penalty = $1 + 2 \times 15 + 2 \times 1 = 33$ bus cycles
- Bandwidth = 16 bytes / 33 cycles = 0.48 B/cycle



c. Interleaved memory organization



Measuring Cache Performance

- Components of CPU time
 - Program execution cycles
 - Include cache hit time
 - Memory stall (miss) cycles
 - Mainly from cache misses
- With simplified assumptions:

Memory stall cycles

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

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



Cache Performance Example

- Given
 - I-cache miss rate = 2% (2 misses per 100 instructions)
 - D-cache miss rate = 4% (4 misses per 100 memory access instructions)
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: $100\% \times 2\% \times 100 = 2$
 - D-cache: $36\% \times 4\% \times 100 = 1.44$
- Total CPI = base CPI + Miss (stall) cycles per instruction
 - Actual CPI = $2 + 2 + 1.44 = 5.44$
 - Ideal CPU is $5.44/2 = 2.72$ times faster



Average Memory Access Time

- Hit time is important to performance
- Average memory access time (AMAT)
 - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
 - $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$
 - 2 cycles per instruction



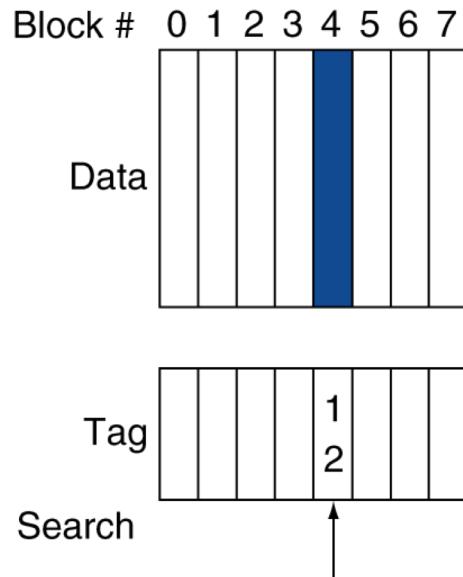
Improve Performance – Associative Caches

- **n -way set associative cache**
 - Each set contains n blocks
 - Each address maps to a unique set
 - Set address = (Block address) modulo (number of sets in cache)
 - A mem block maps to any block within the corresponding set
 - However, to locate a block in a set, need to compare n times
 - all n tags in a set must be checked and compared
 - n comparators (more effective - faster)
- Fully associative – opposite **extreme** of direct mapped
 - Allow a given block to go in any cache entry
 - Must search all entries to find a hit
 - One comparator each block (expensive)
 - # of comparator = cache size (block number)



Associative Cache Example

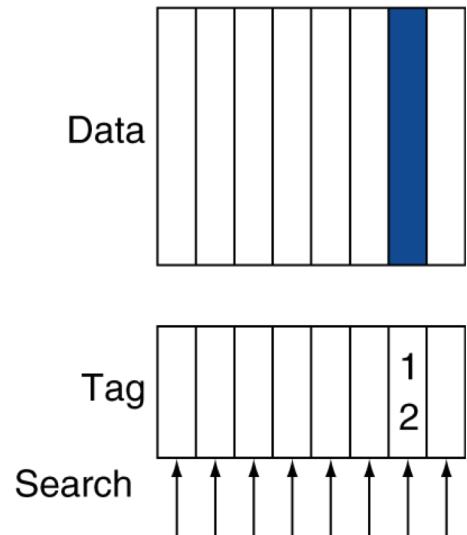
Direct mapped



Set associative



Fully associative



Spectrum of Associativity

- For a cache with 8 blocks

**One-way set associative
(direct mapped)**

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data												

Associativity Example

- Compare caches of 4 two-word blocks
 - Direct mapped, 2-way set associative, fully associative
 - Block access sequence: 0, 8, 0, 12, 8

Associativity Example

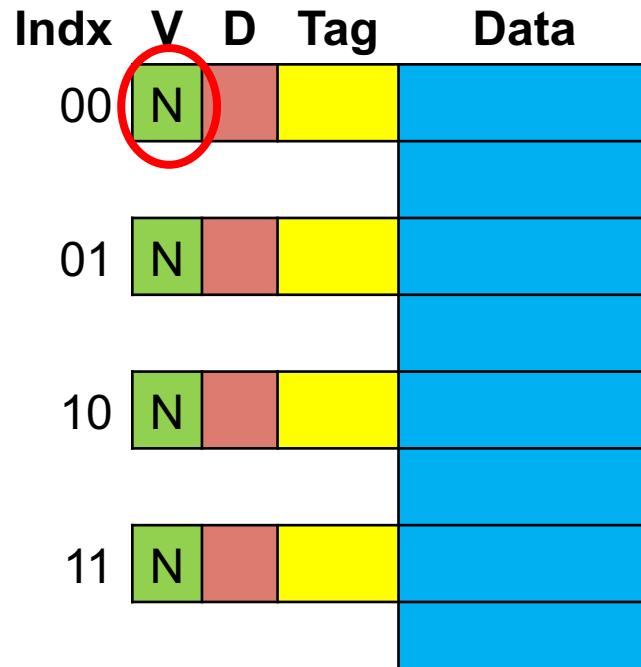
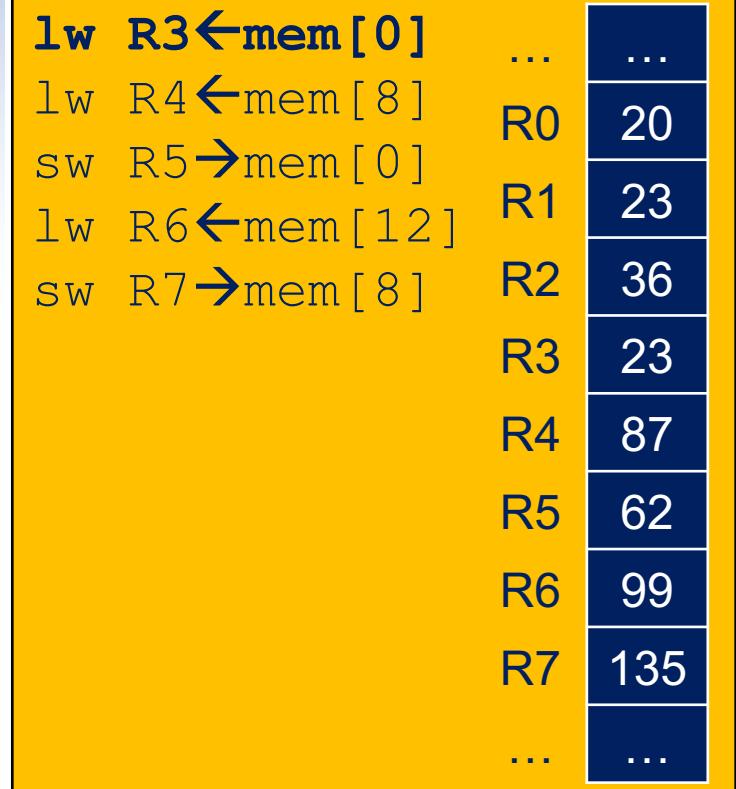
Direct mapped (1-way associative)

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	00 00 0	miss	00

m



Miss



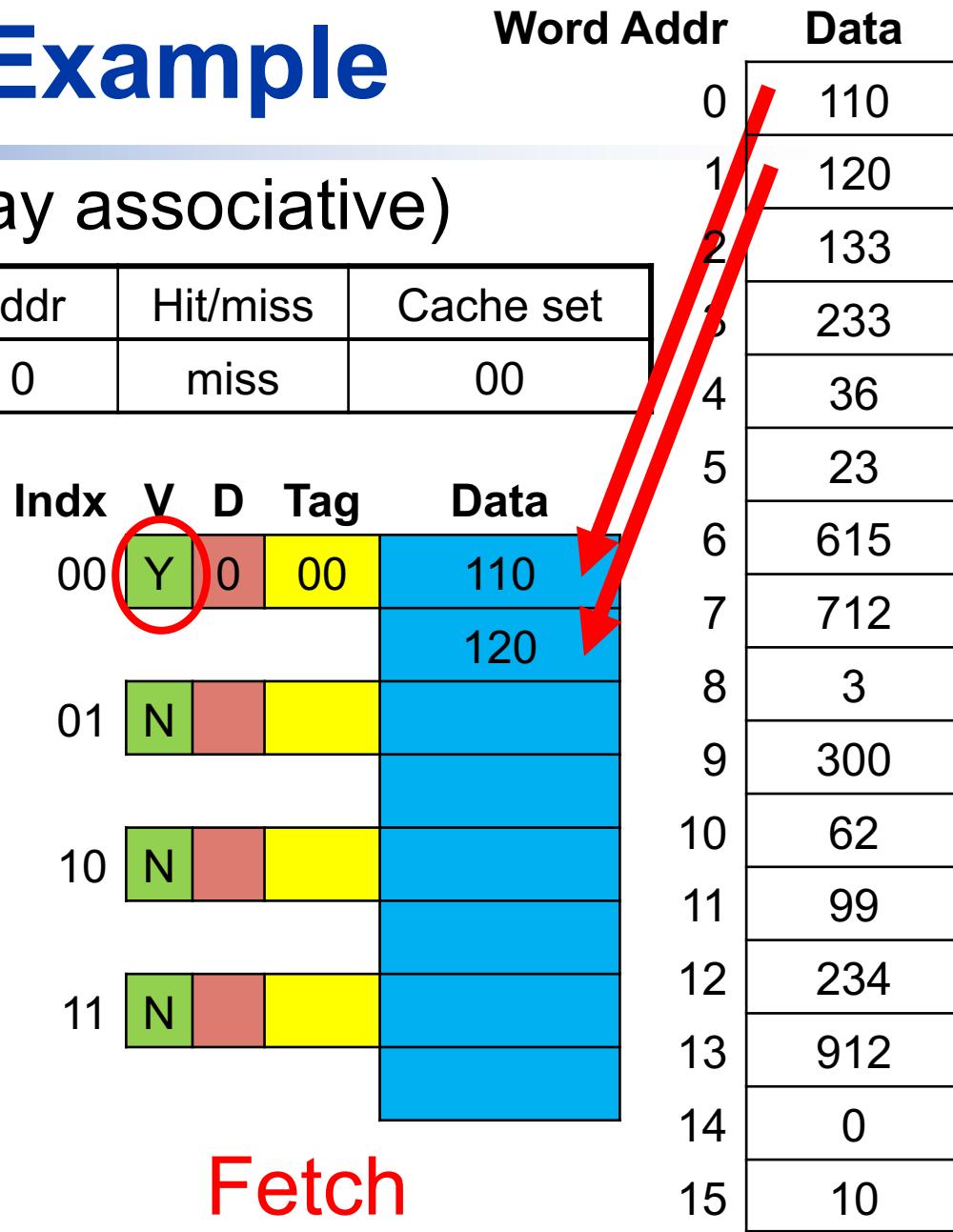
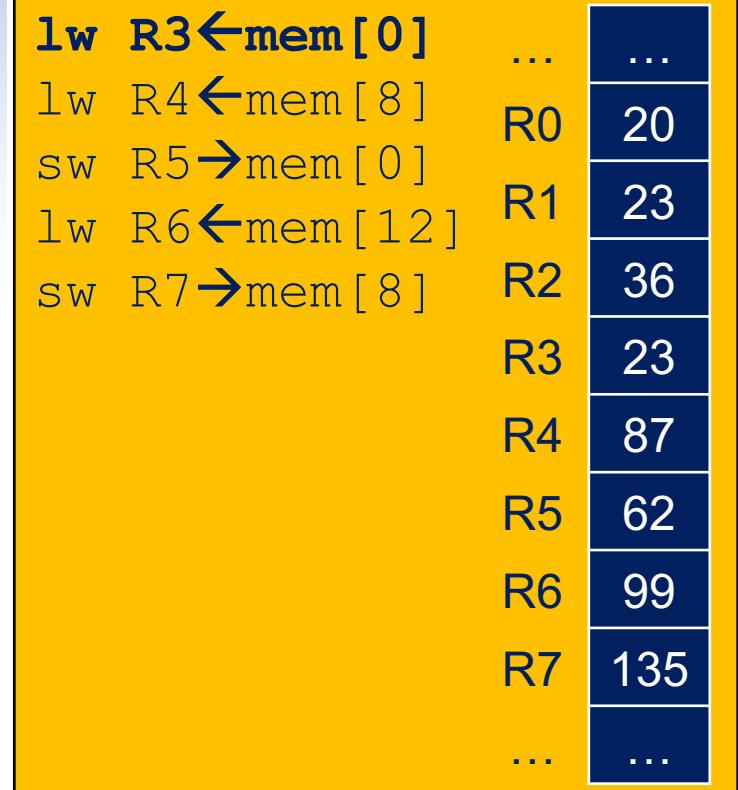
CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	00 00 0	miss	00

m



CPU

Associativity Example

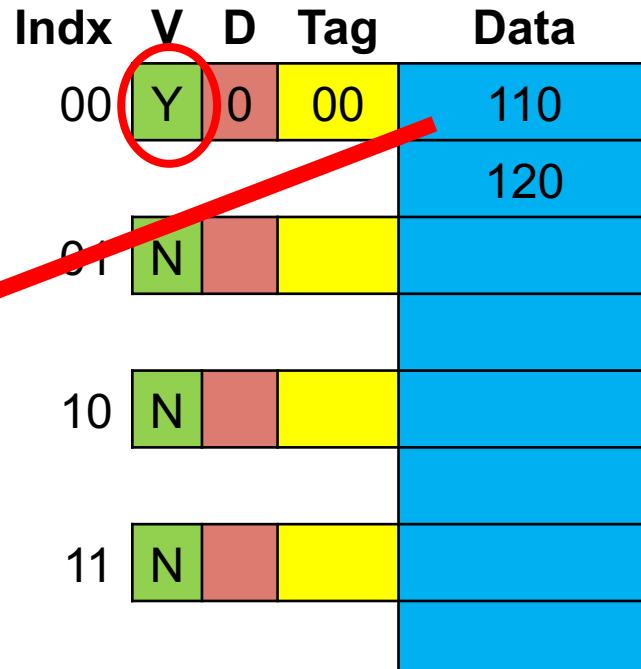
- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	00 00 0	hit	00

m

lw R3 ← mem[0]
lw R4 ← mem[8]
sw R5 → mem[0]
lw R6 ← mem[12]
sw R7 → mem[8]

...	...
R0	20
R1	23
R2	36
R3	110
R4	87
R5	62
R6	99
R7	135
...	...



Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

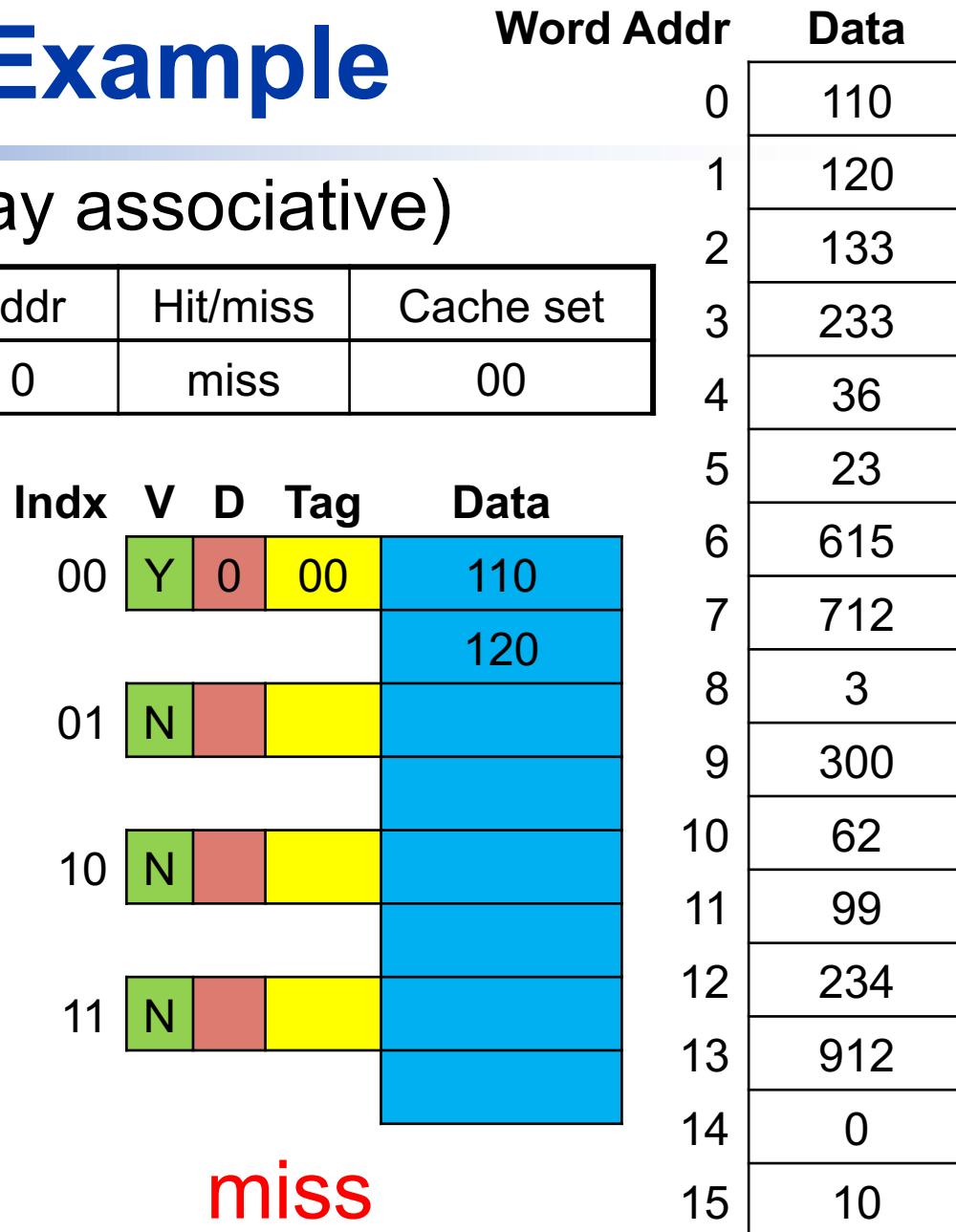
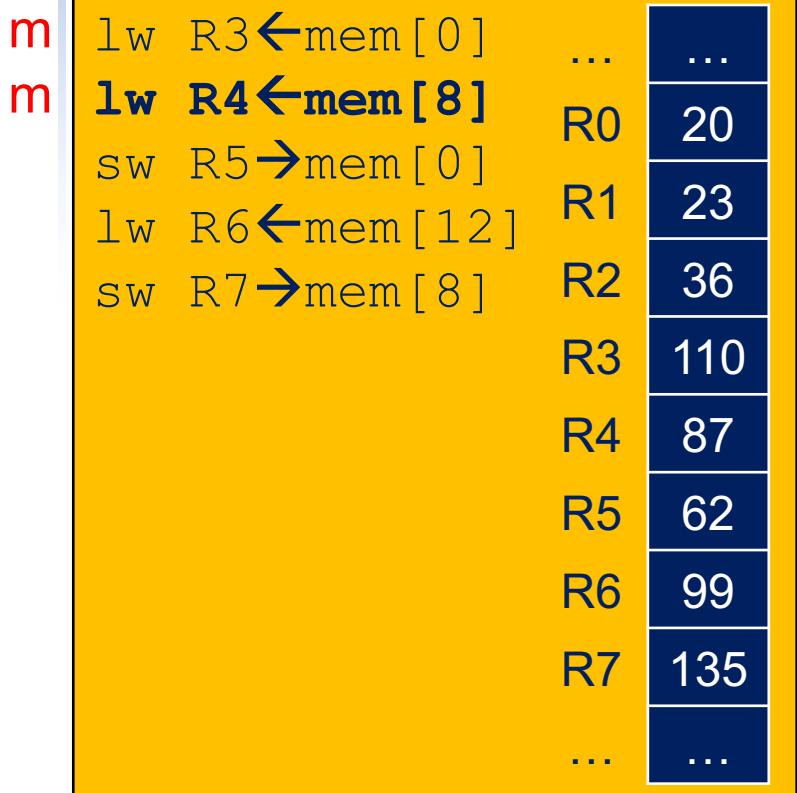


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	miss	00

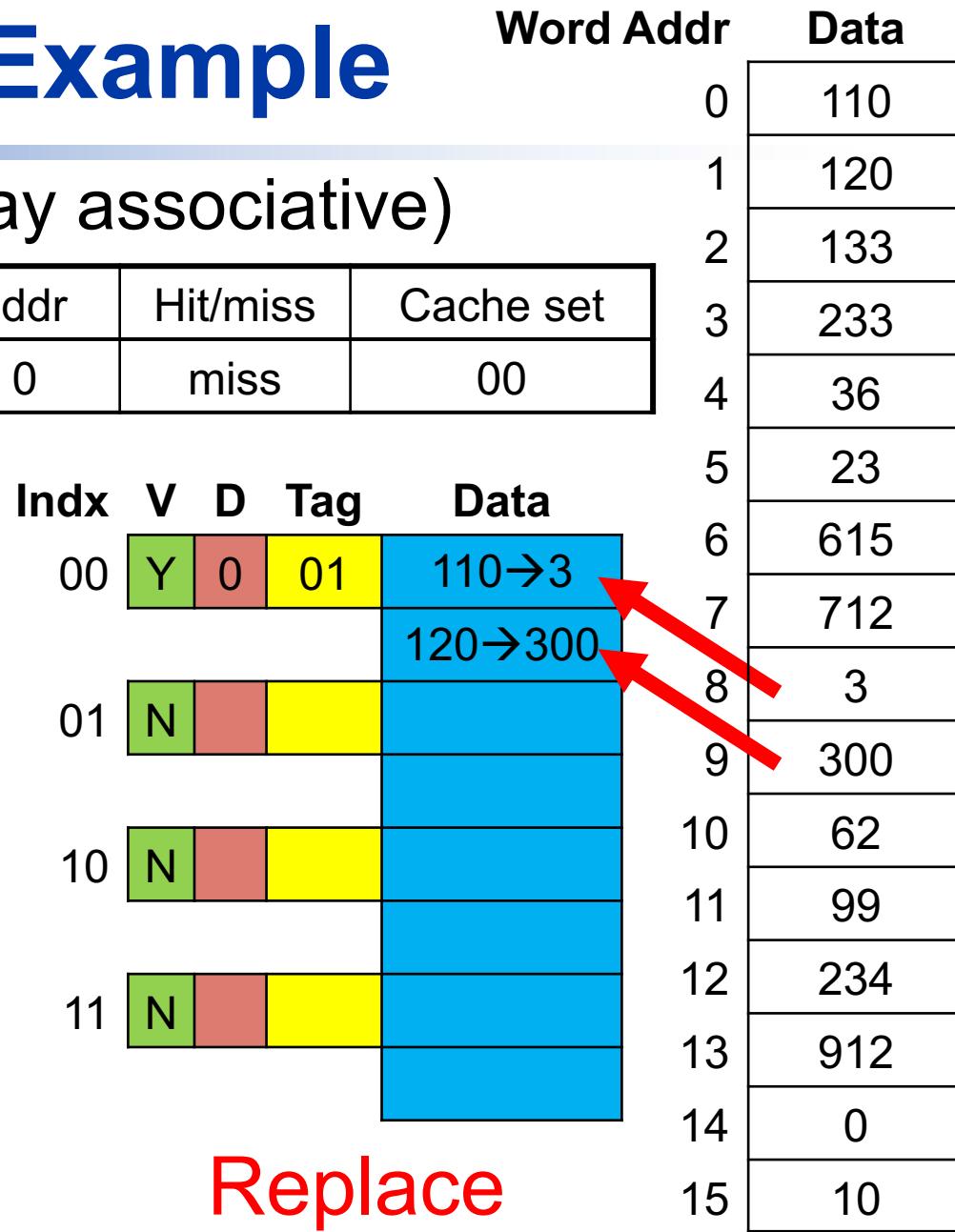
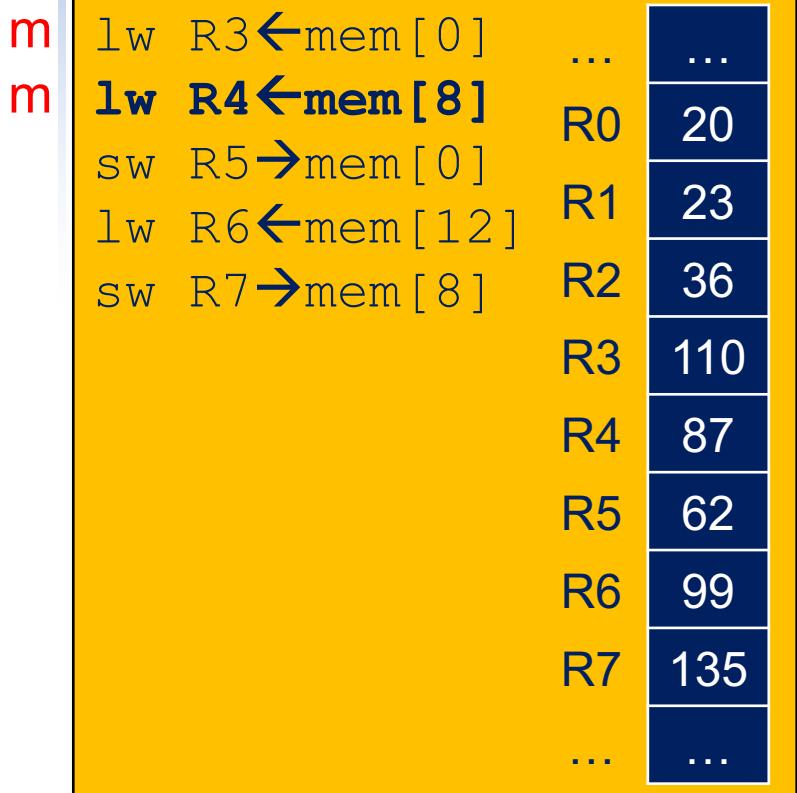


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	miss	00

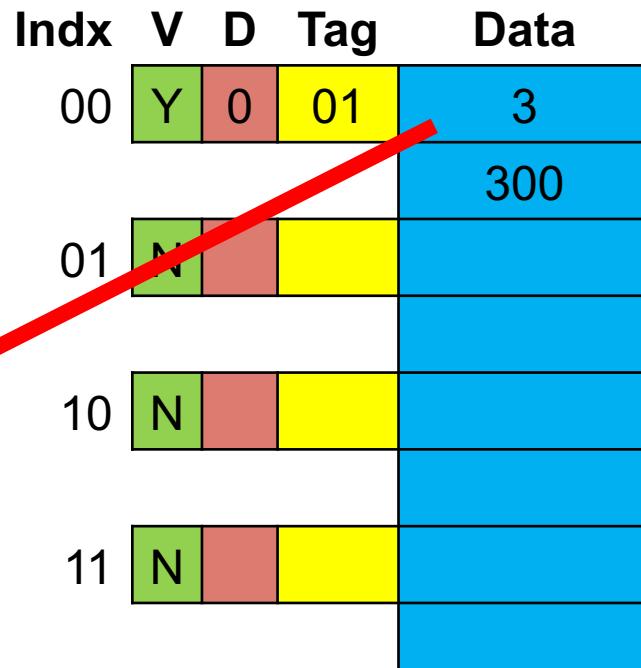
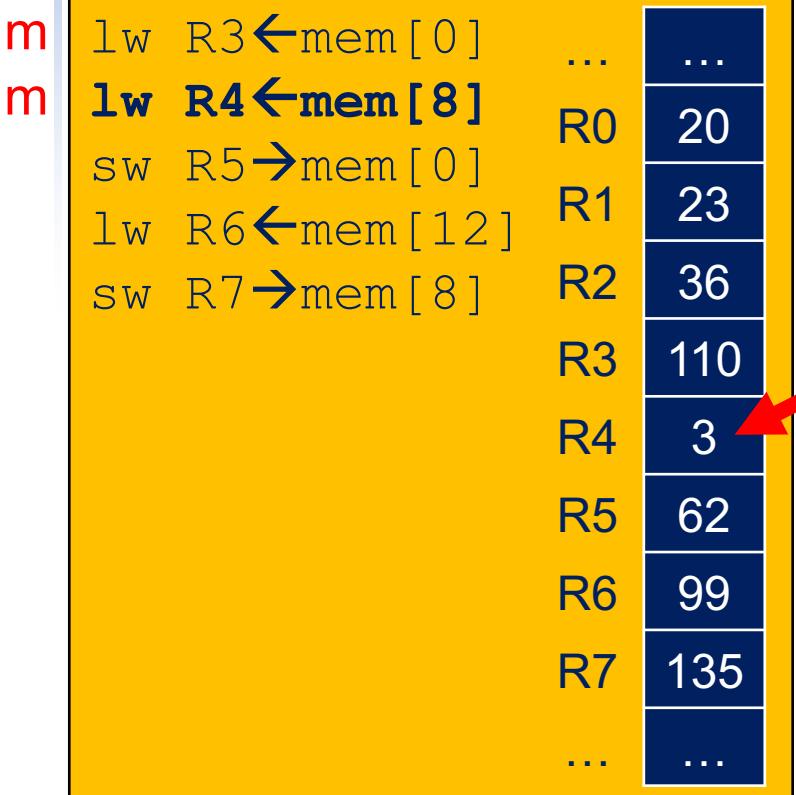


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	hit	00



Load again

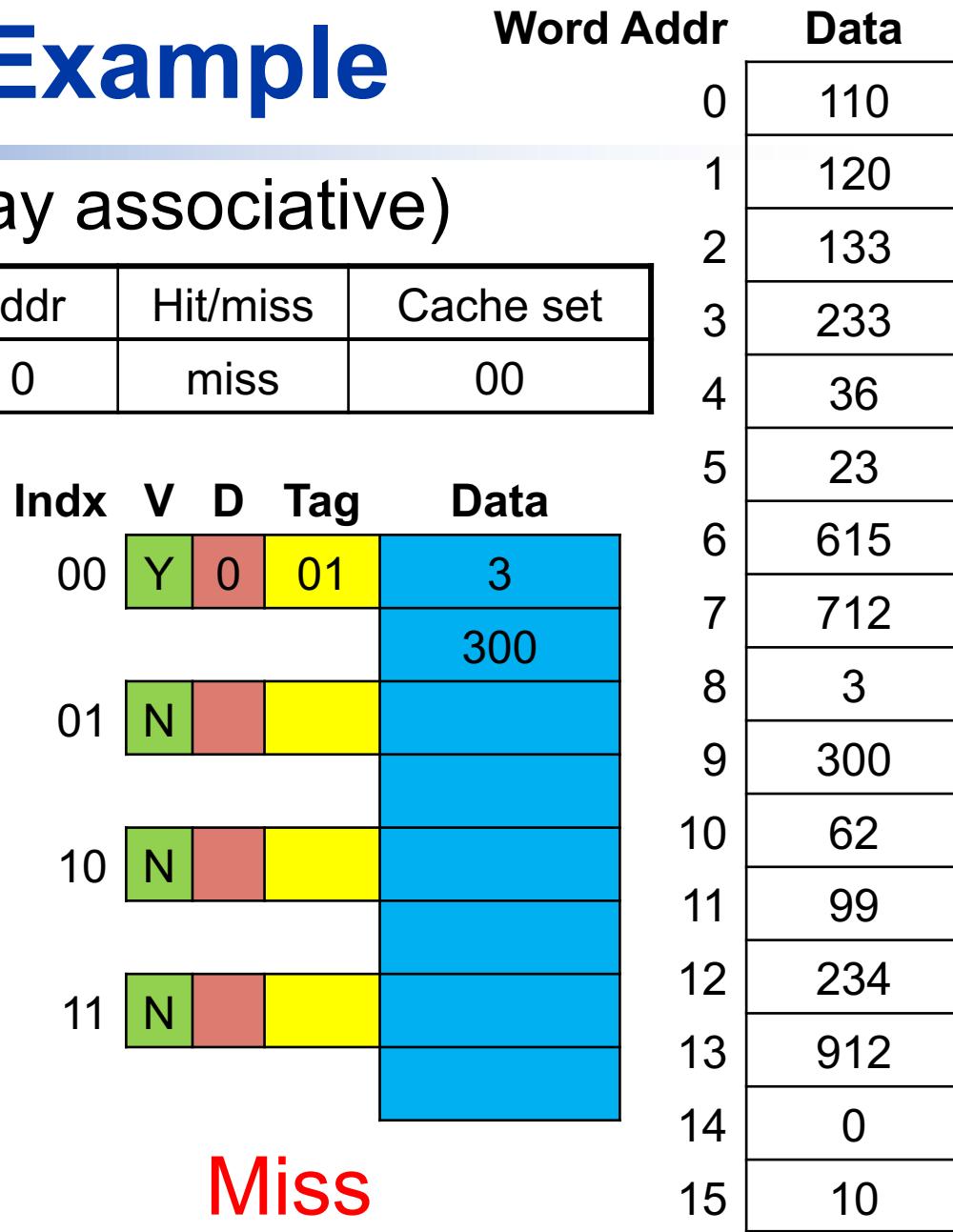
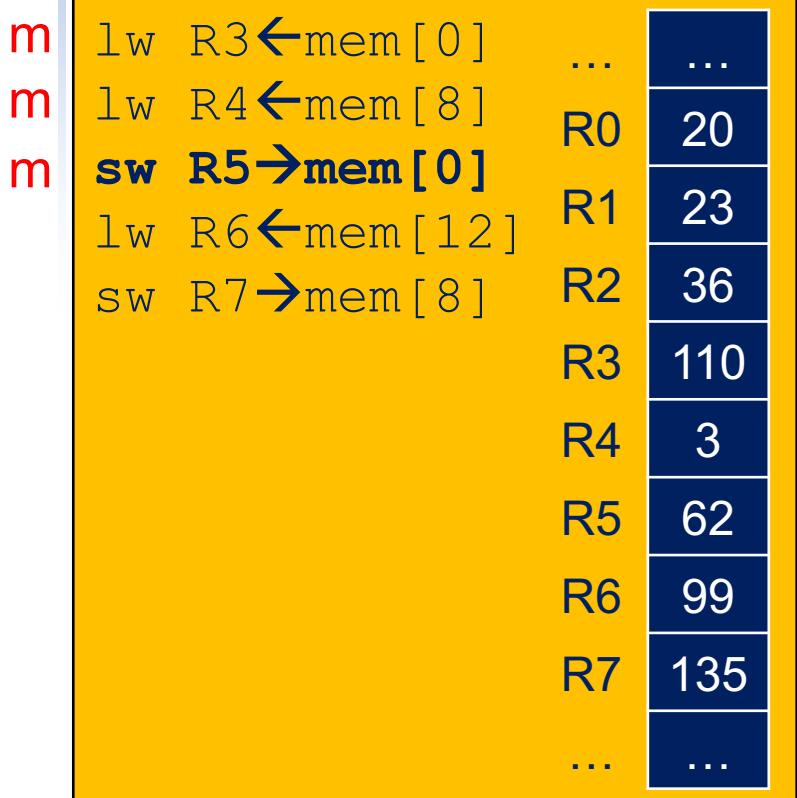


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	00 00 0	miss	00

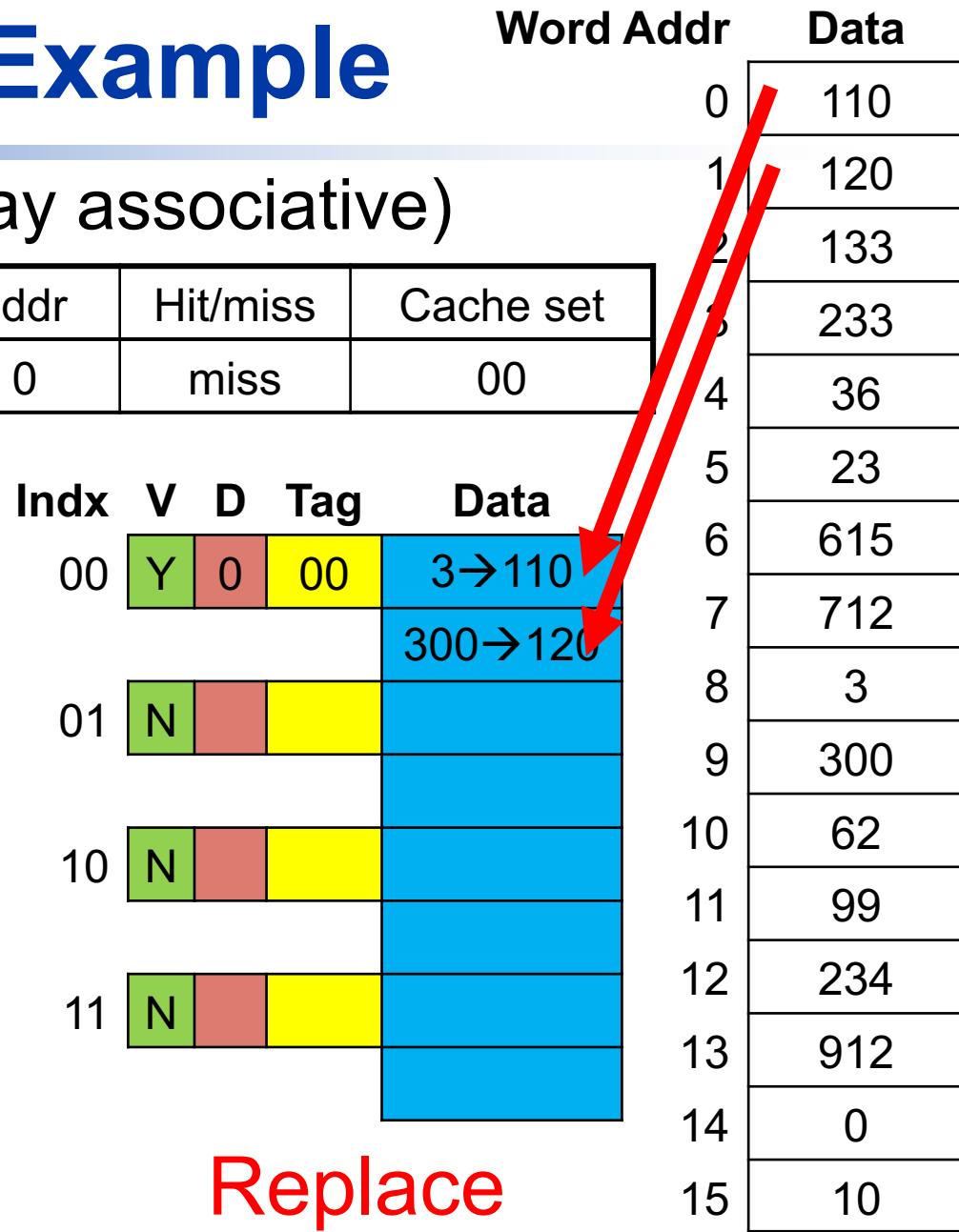
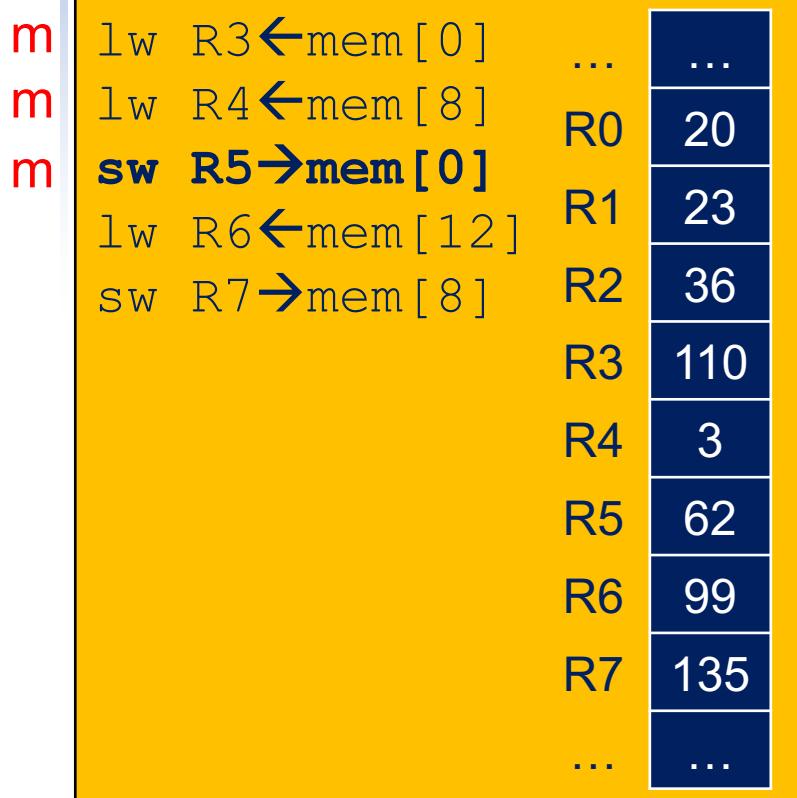


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	00 00 0	miss	00

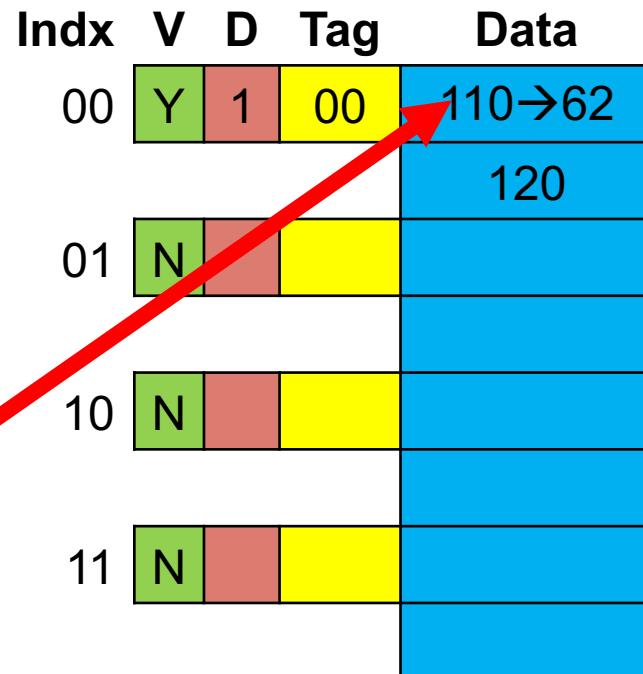
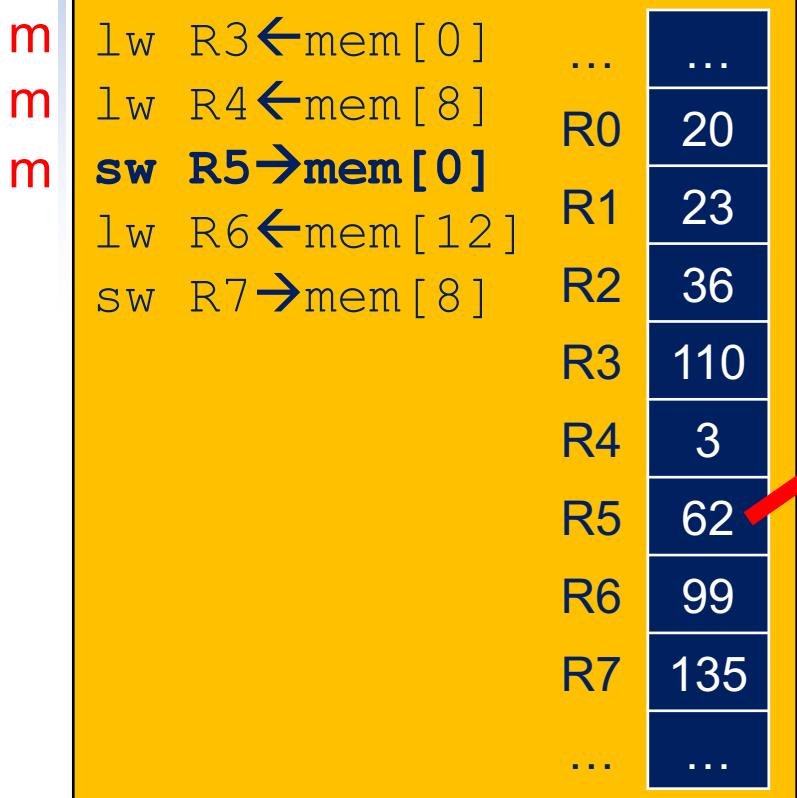


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	00 00 0	hit	00



Write, set dirty

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

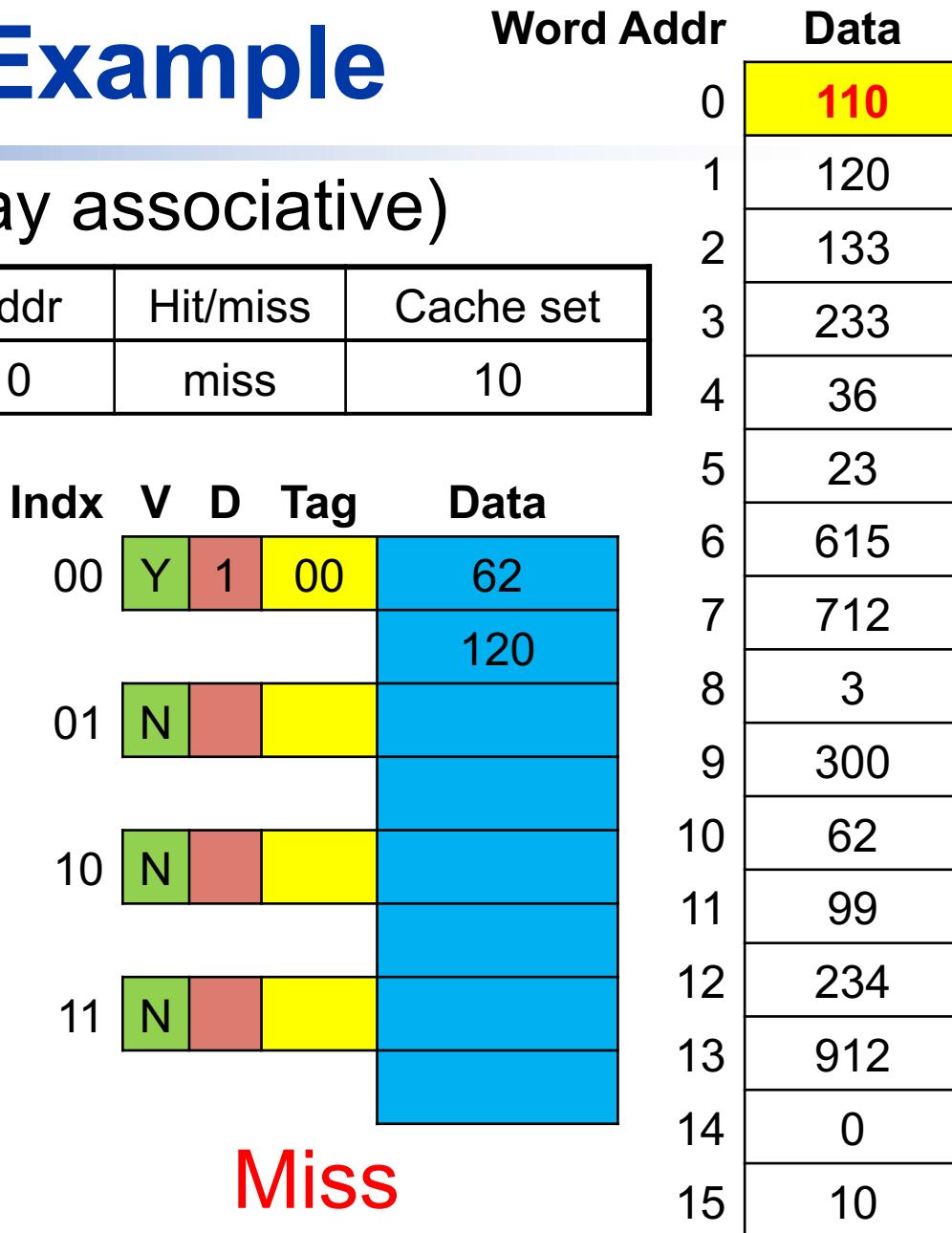
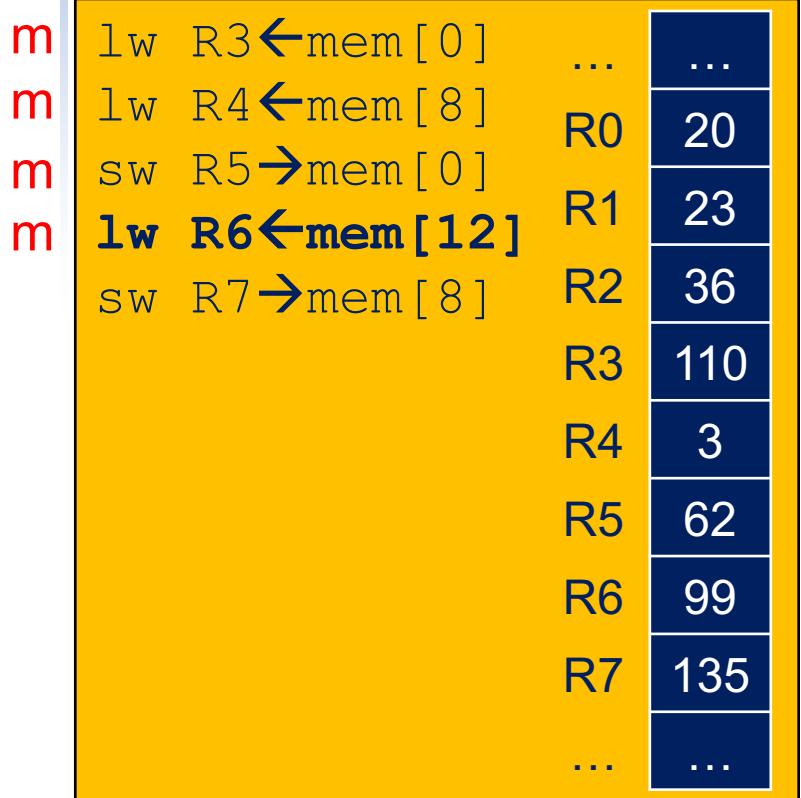


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	01 10 0	miss	10

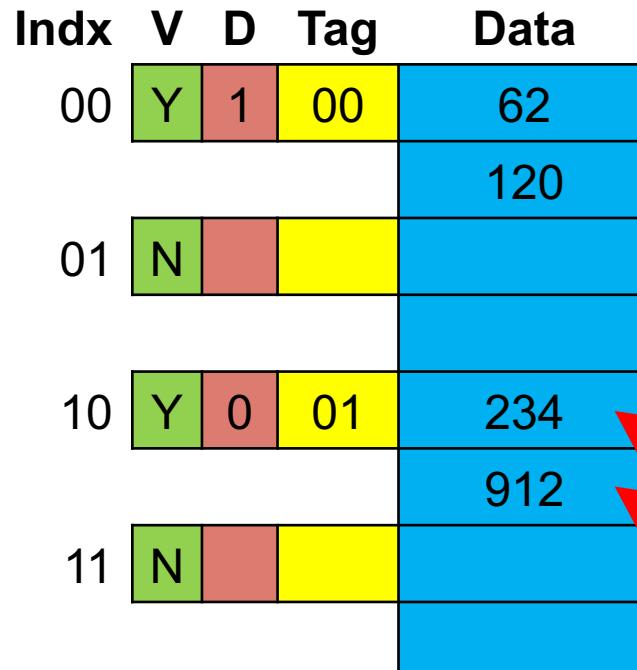
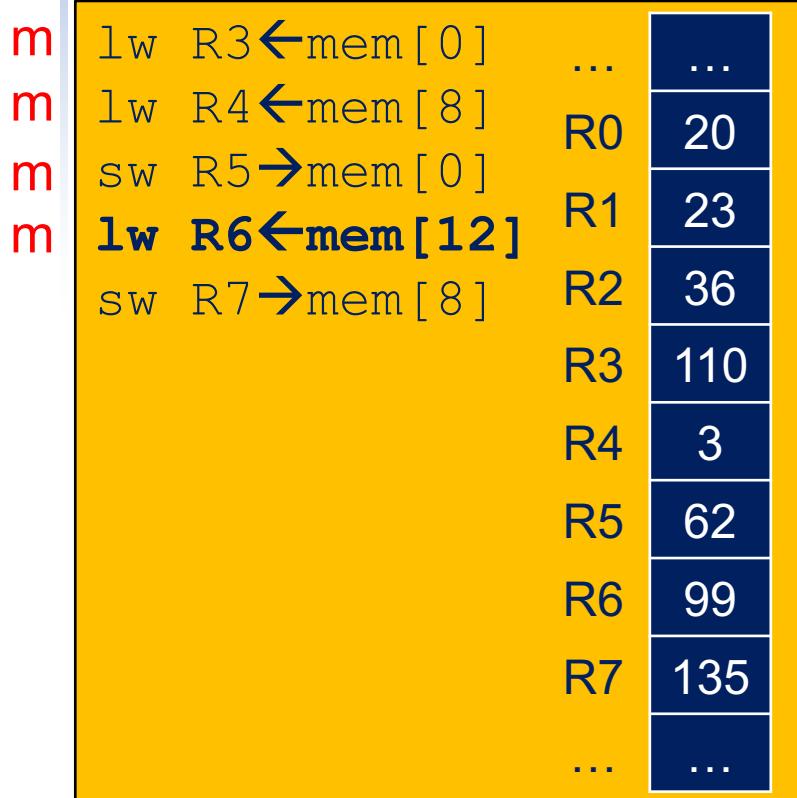


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	01 10 0	miss	10



Fetch



CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	01 10 0	hit	10

m lw R3 ← mem[0]
m lw R4 ← mem[8]
m sw R5 → mem[0]
m lw R6 ← mem[12]
m sw R7 → mem[8]

...	...
R0	20
R1	23
R2	36
R3	110
R4	3
R5	62
R6	234
R7	135
...	...

Indx	V	D	Tag	Data
00	Y	1	00	62
01	N			120
10	Y	0	01	234
11	N			912

Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

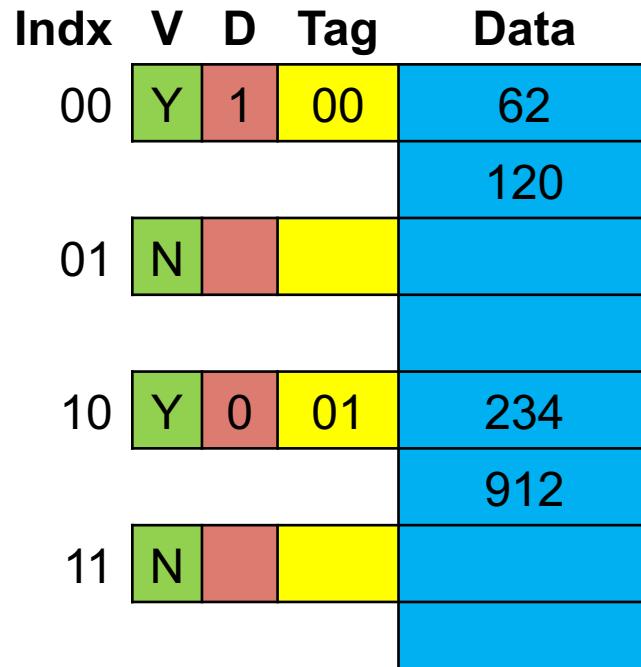
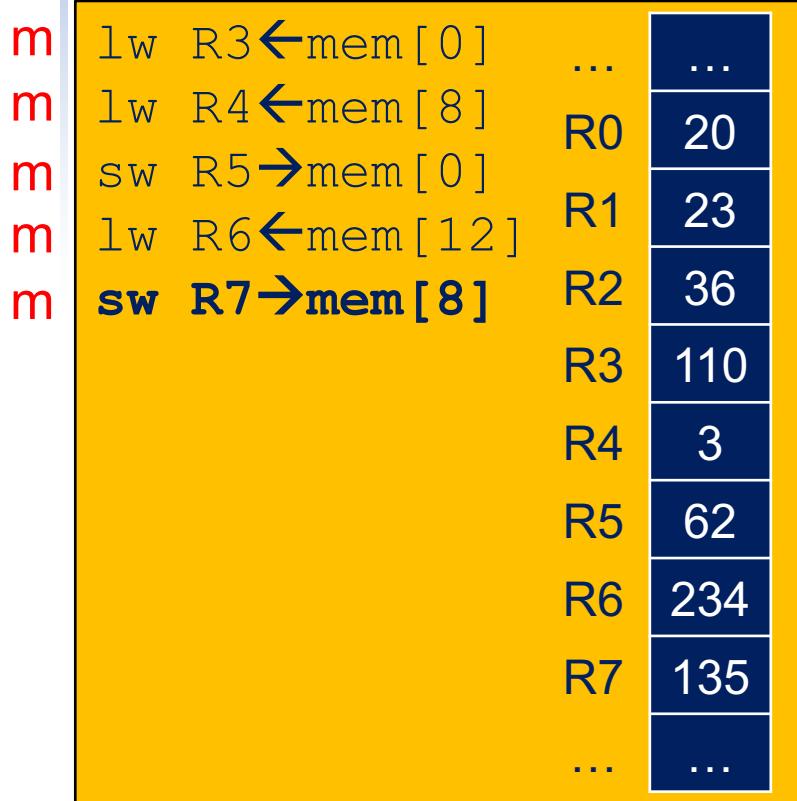


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	miss	00



Miss

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

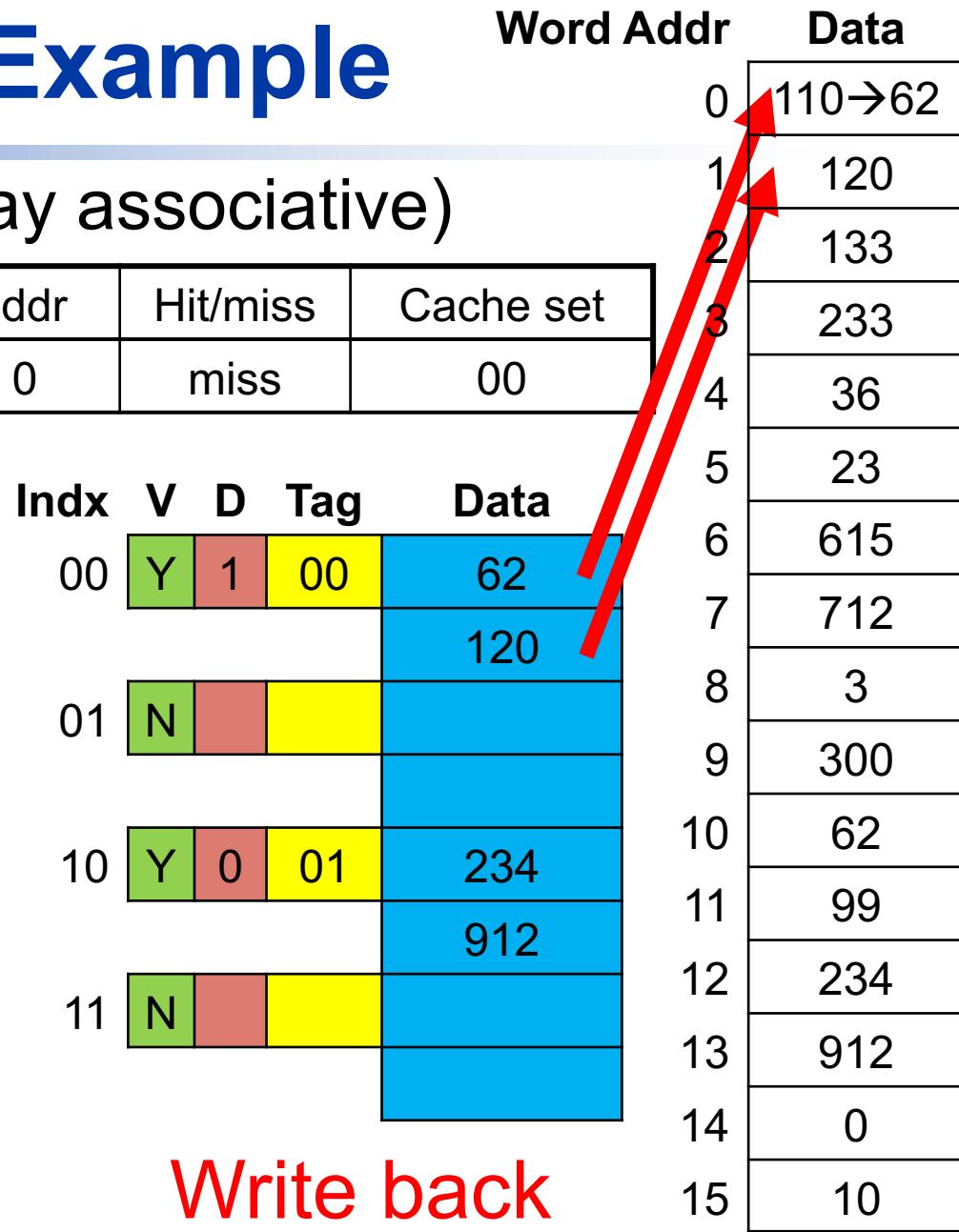
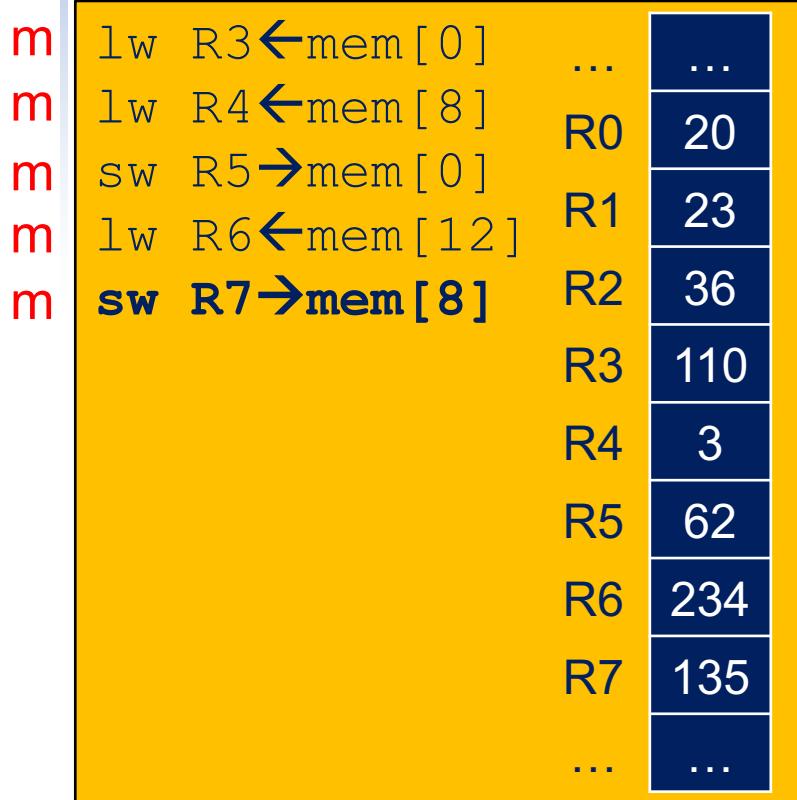


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	miss	00

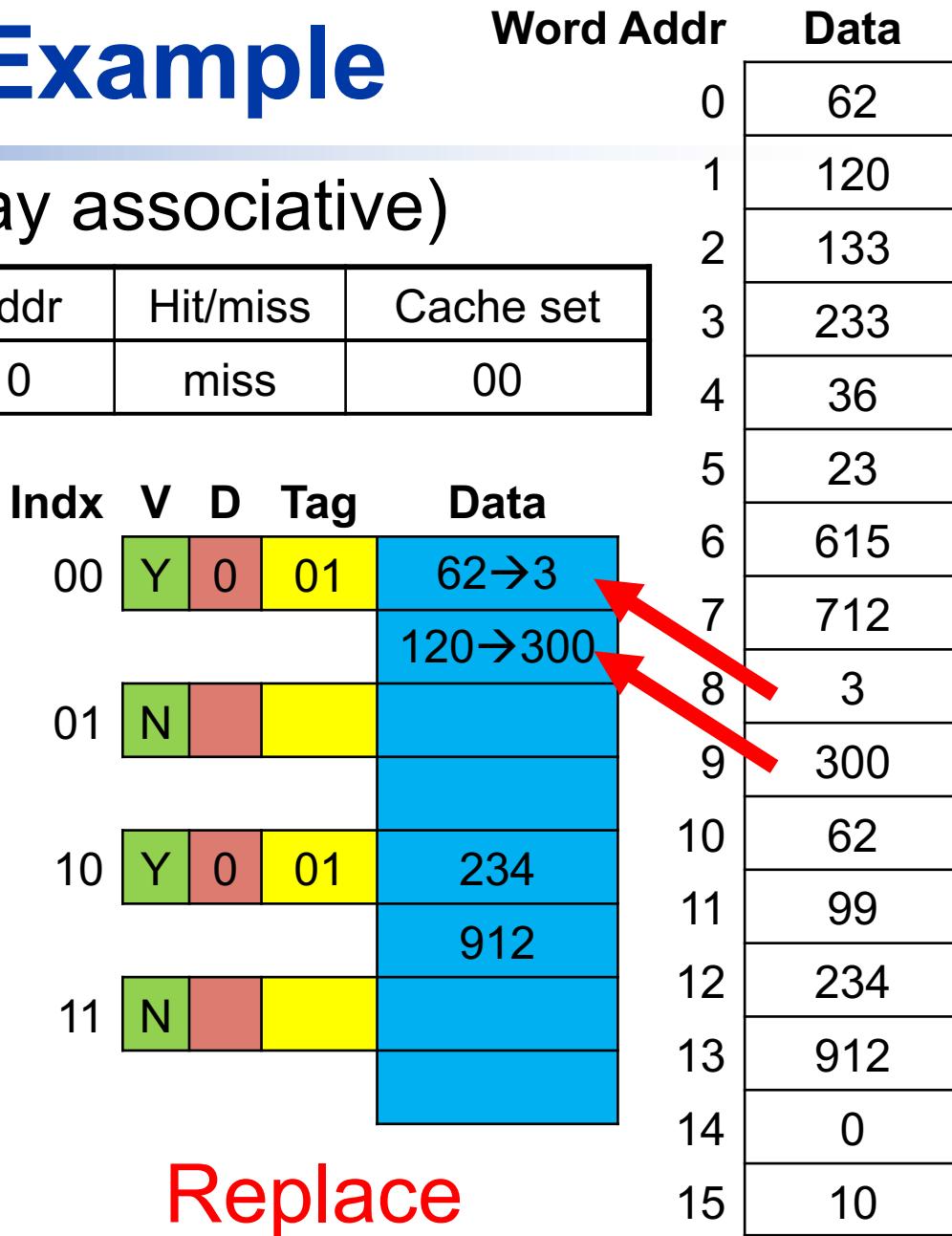
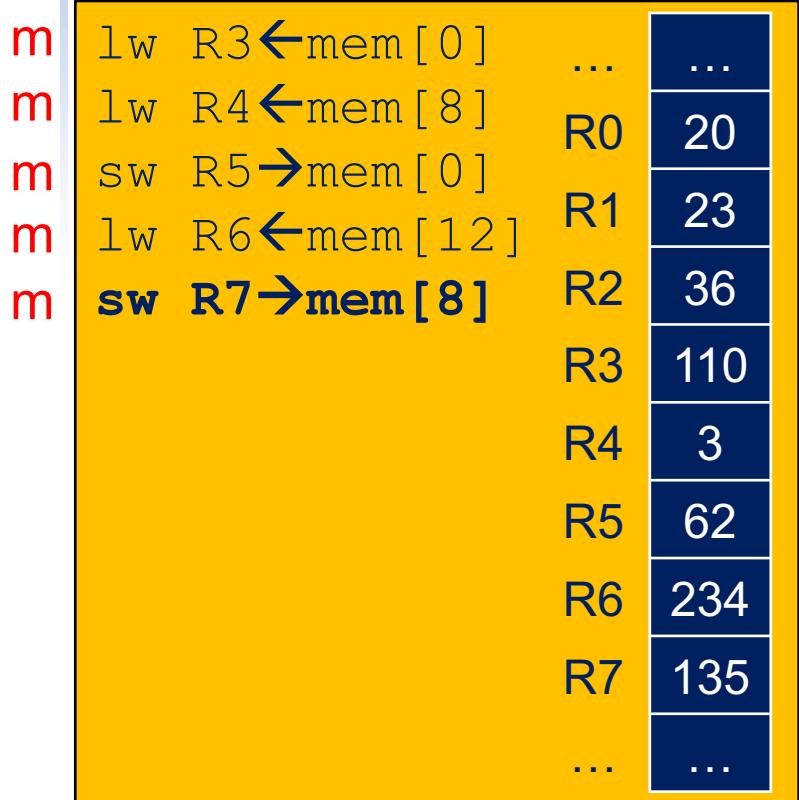


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	miss	00

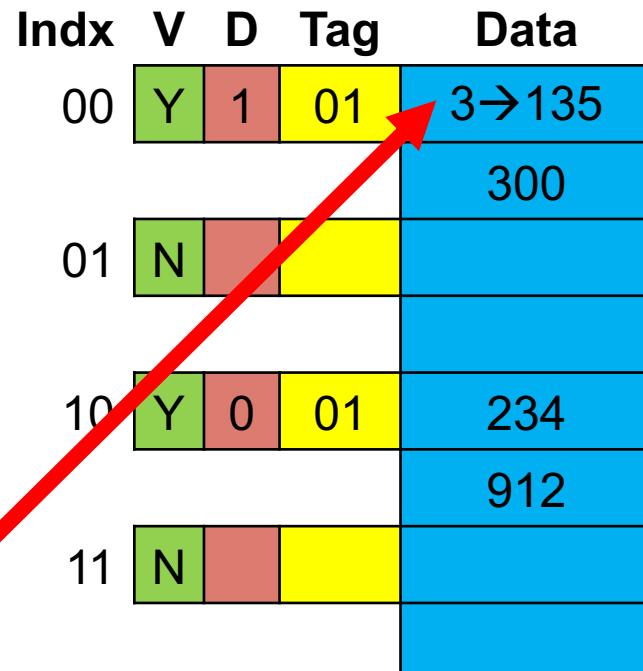
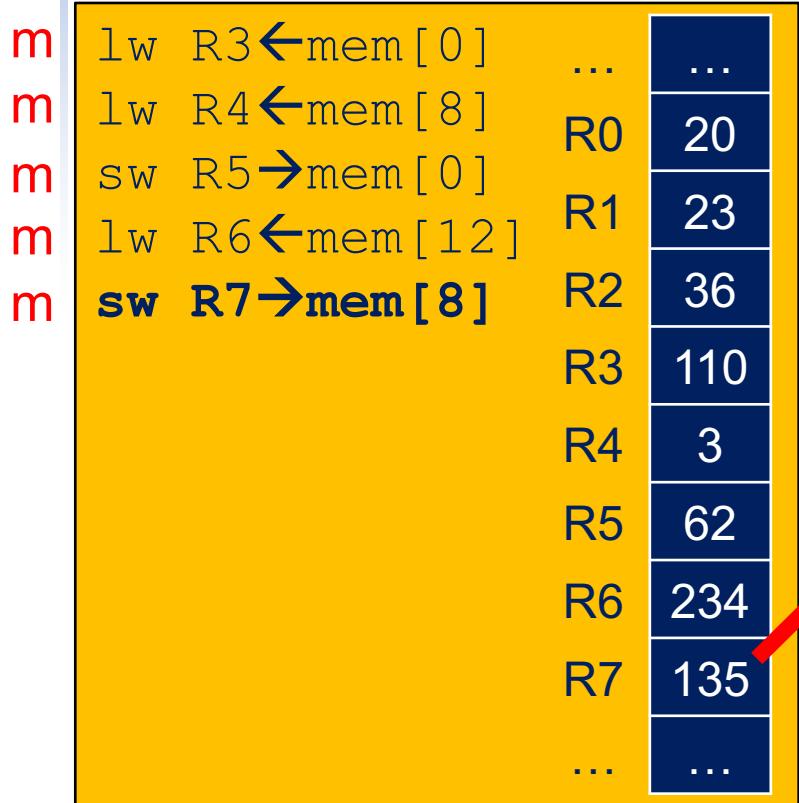


CPU

Associativity Example

- Direct mapped (1-way associative)

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	01 00 0	miss	00



Write, set dirty

Word Addr	Data
0	62
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Associativity Example

2-way associative

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	000 0 0	miss	0

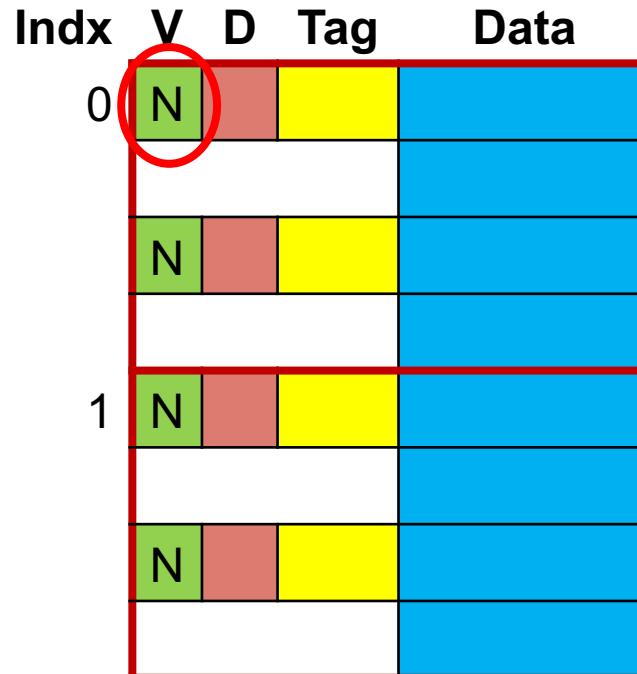
m

```

lw R3<-mem[0]
lw R4<-mem[8]
sw R5->mem[0]
lw R6<-mem[12]
sw R7->mem[8]

```

...	...
R0	20
R1	23
R2	36
R3	23
R4	87
R5	62
R6	99
R7	135
...	...



Miss

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

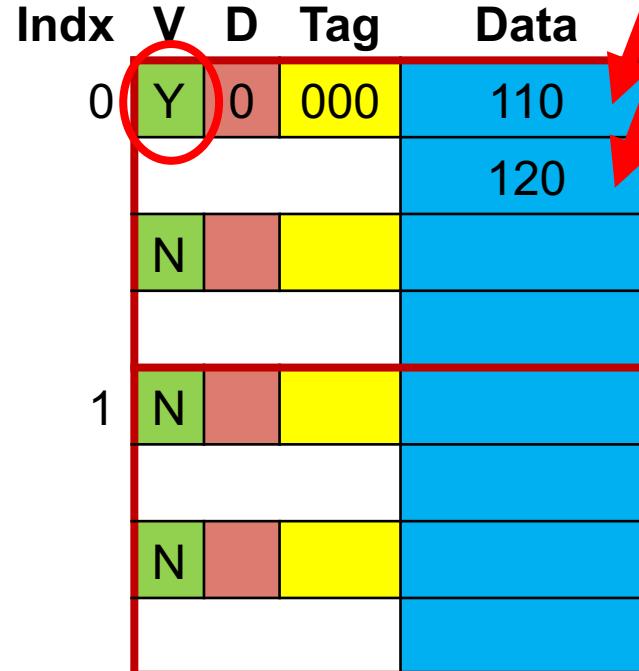
Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	000 0 0	miss	0

m

lw R3 ← mem[0]
lw R4 ← mem[8]	R0	20
sw R5 → mem[0]	R1	23
lw R6 ← mem[12]	R2	36
sw R7 → mem[8]	R3	23
	R4	87
	R5	62
	R6	99
	R7	135



Fetch



CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	000 0 0	hit	0

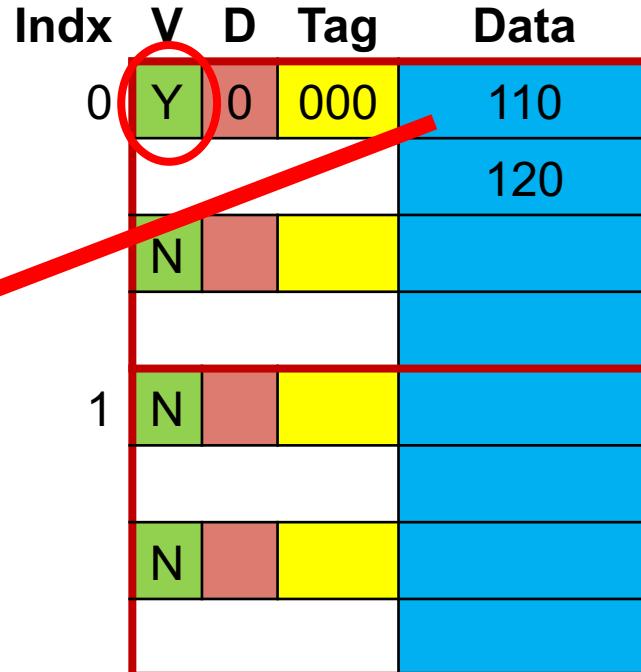
m

```

lw R3<-mem[0]
lw R4<-mem[8]
sw R5->mem[0]
lw R6<-mem[12]
sw R7->mem[8]

```

...	...
R0	20
R1	23
R2	36
R3	110
R4	87
R5	62
R6	99
R7	135
...	...



Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

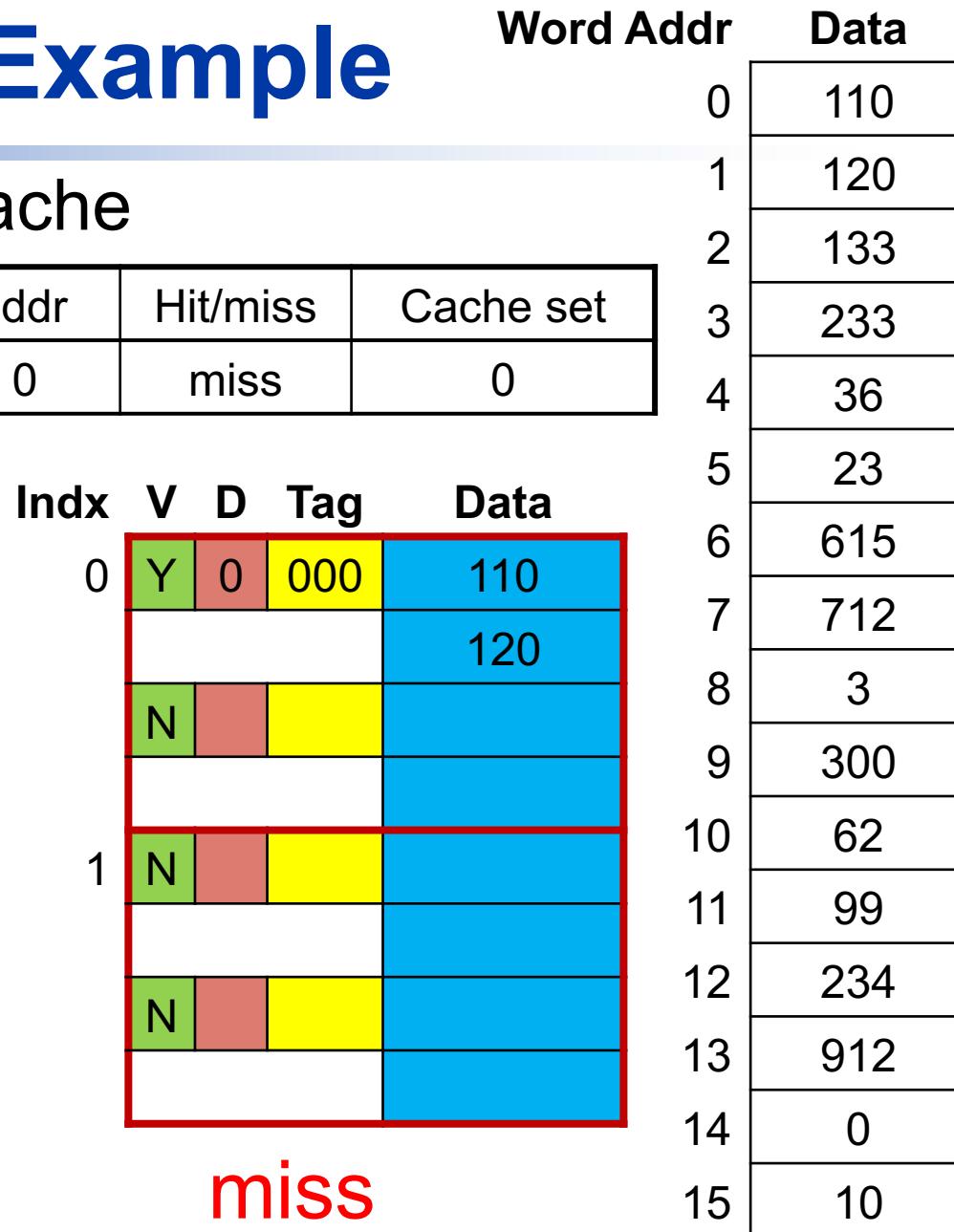
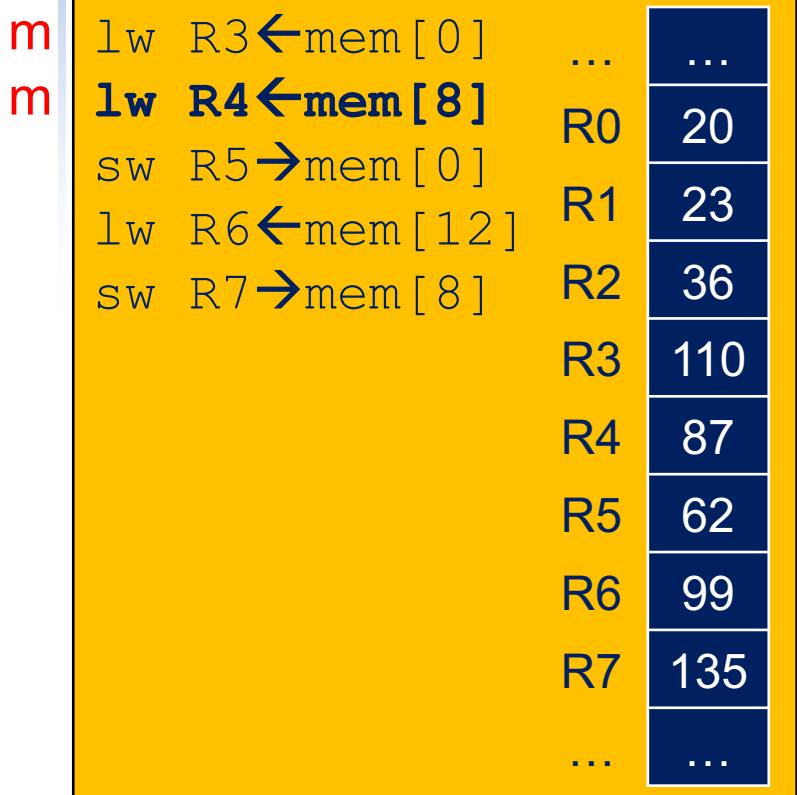


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 <u>0</u> 0	miss	0

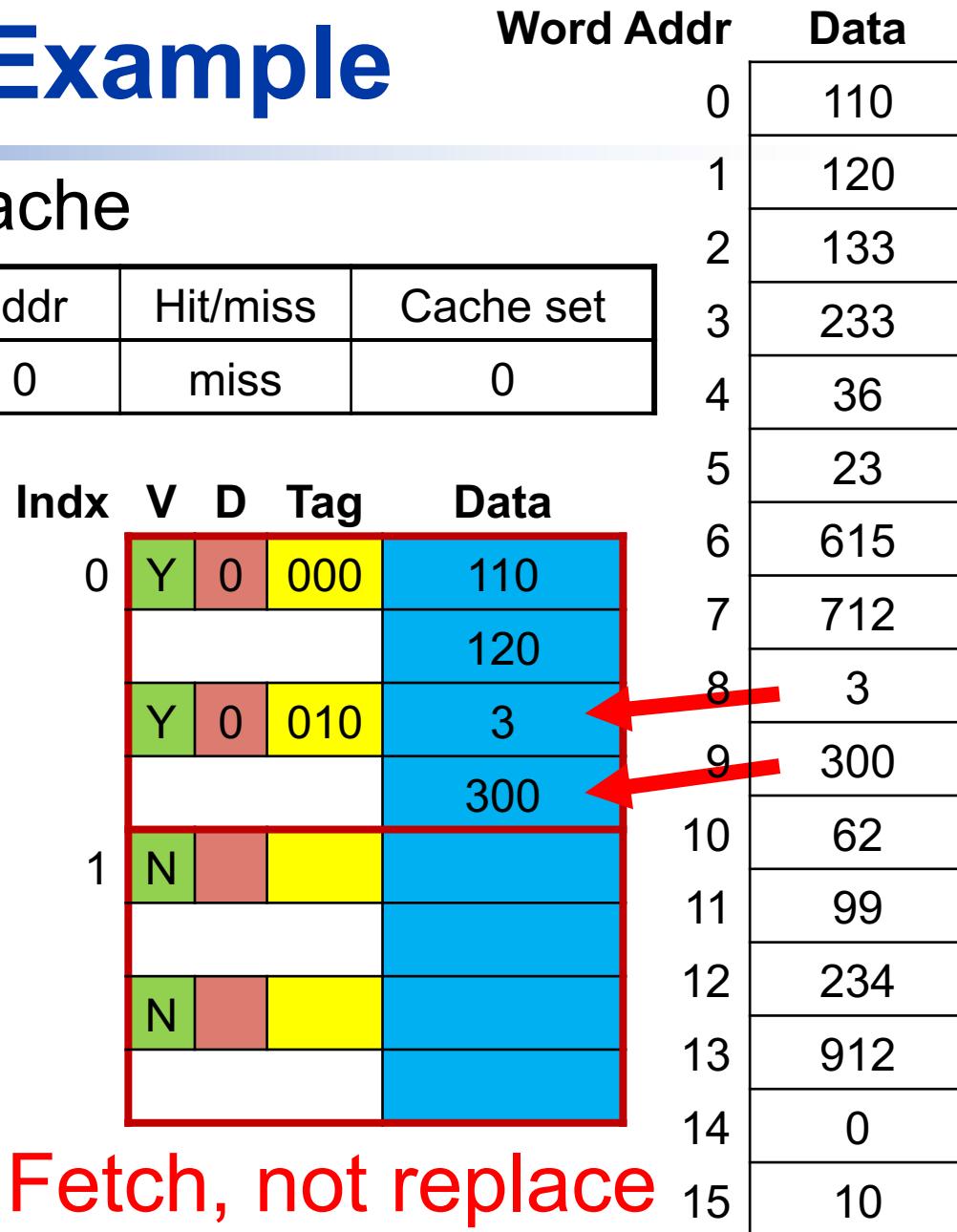
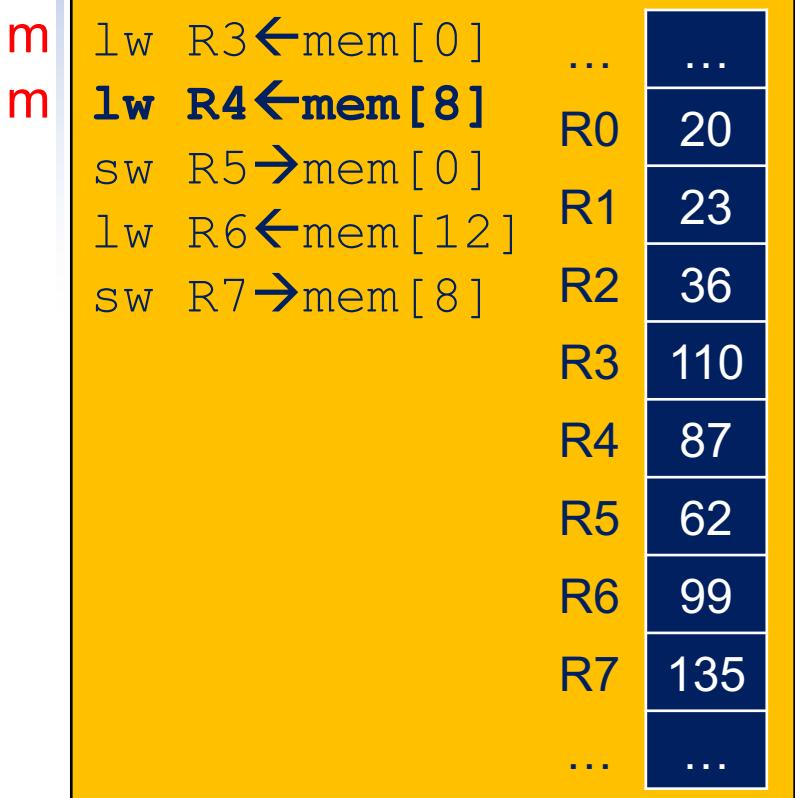


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 0 0	miss	0

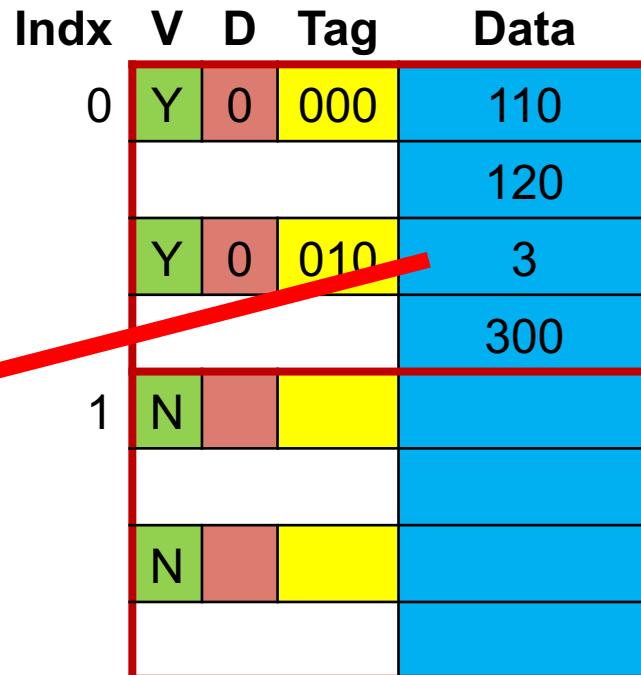
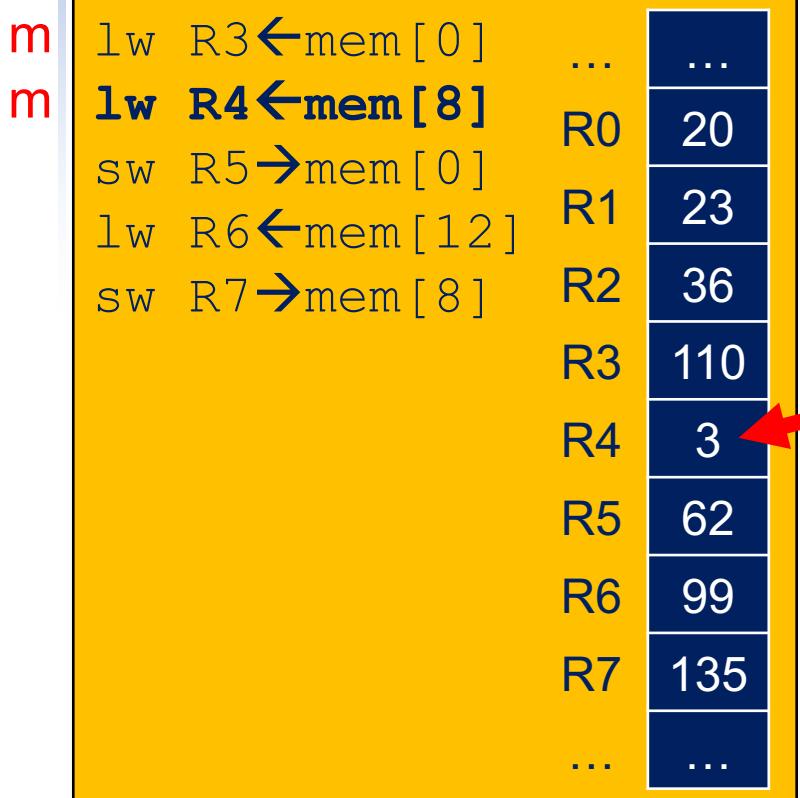


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 0 0	hit	0



Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

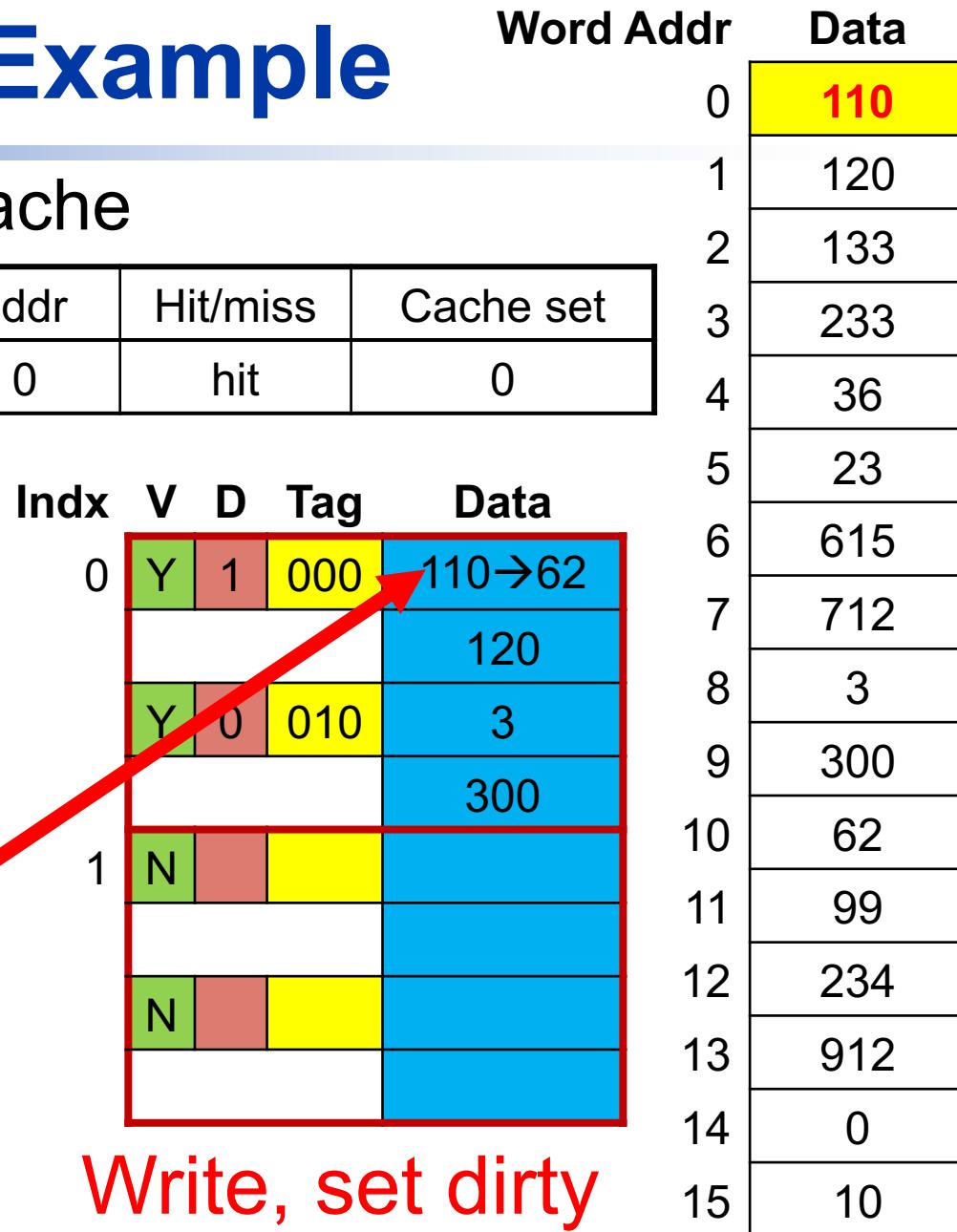
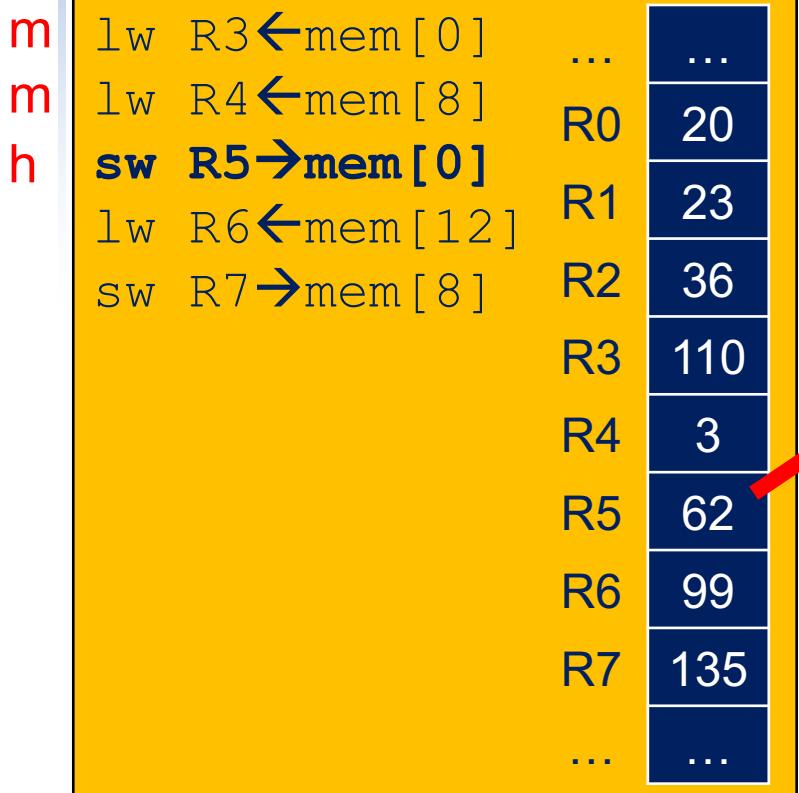


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	000 0 0	hit	0

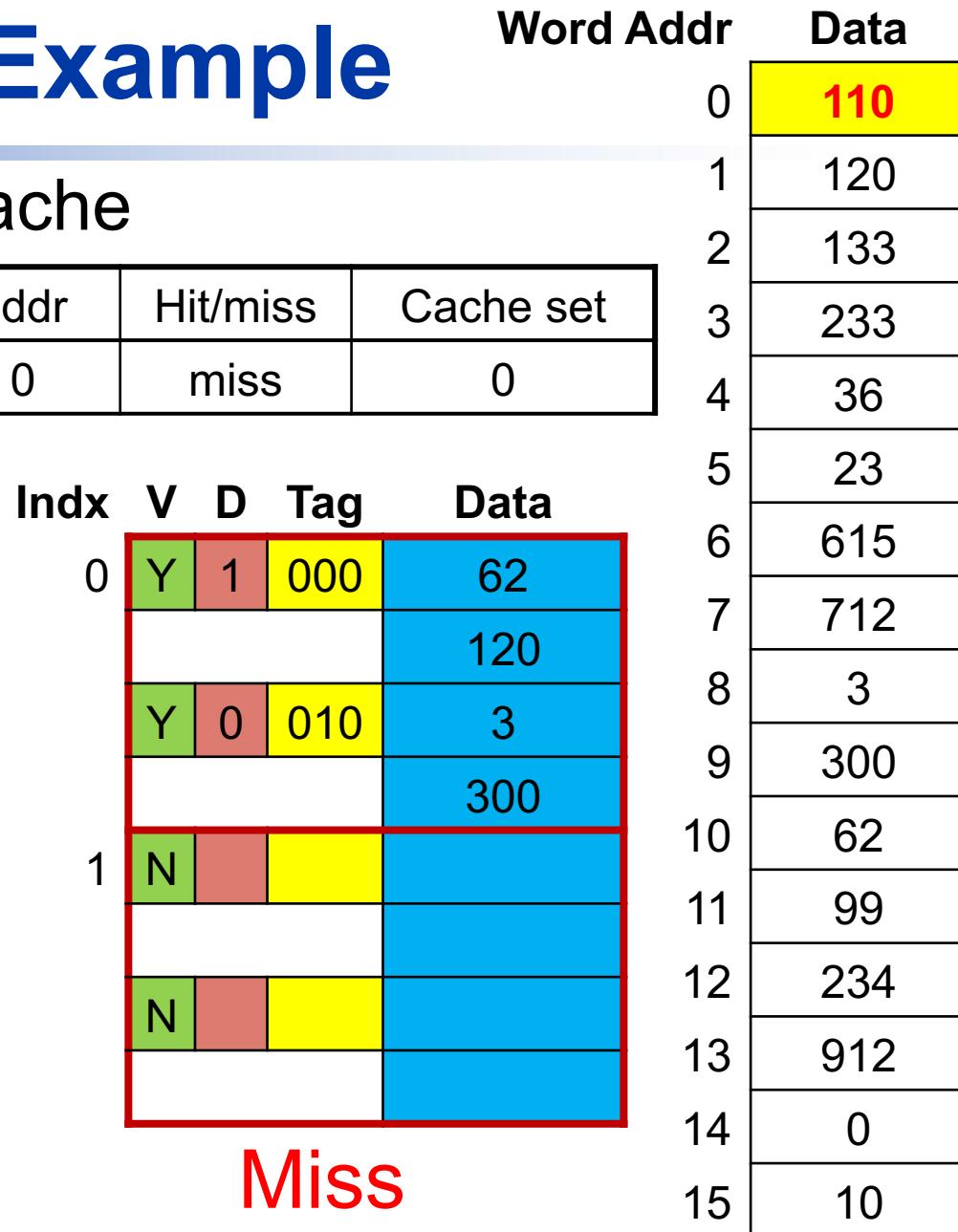
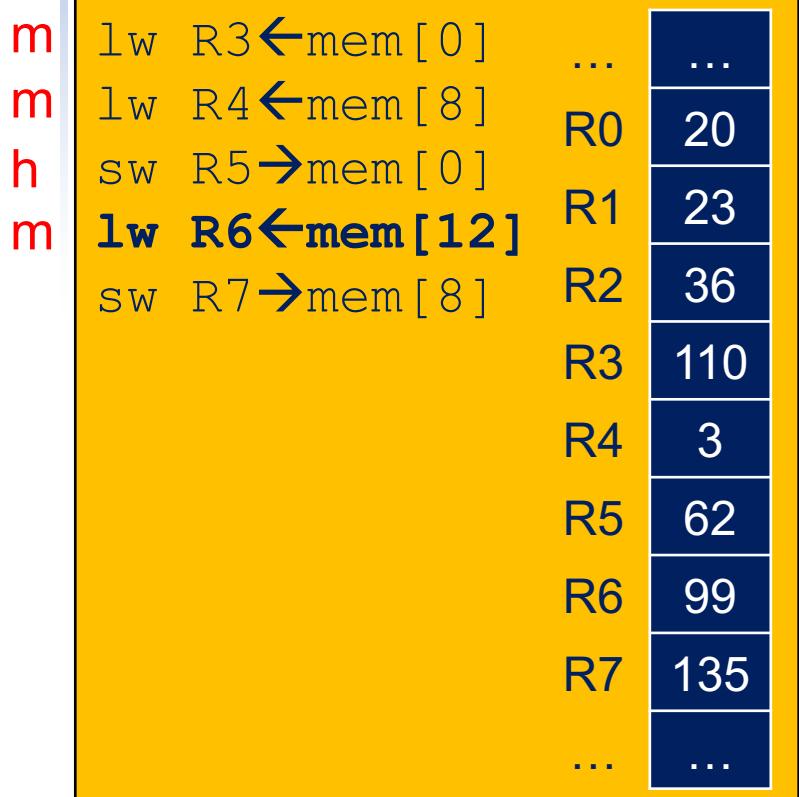


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	011 0 0	miss	0



CPU

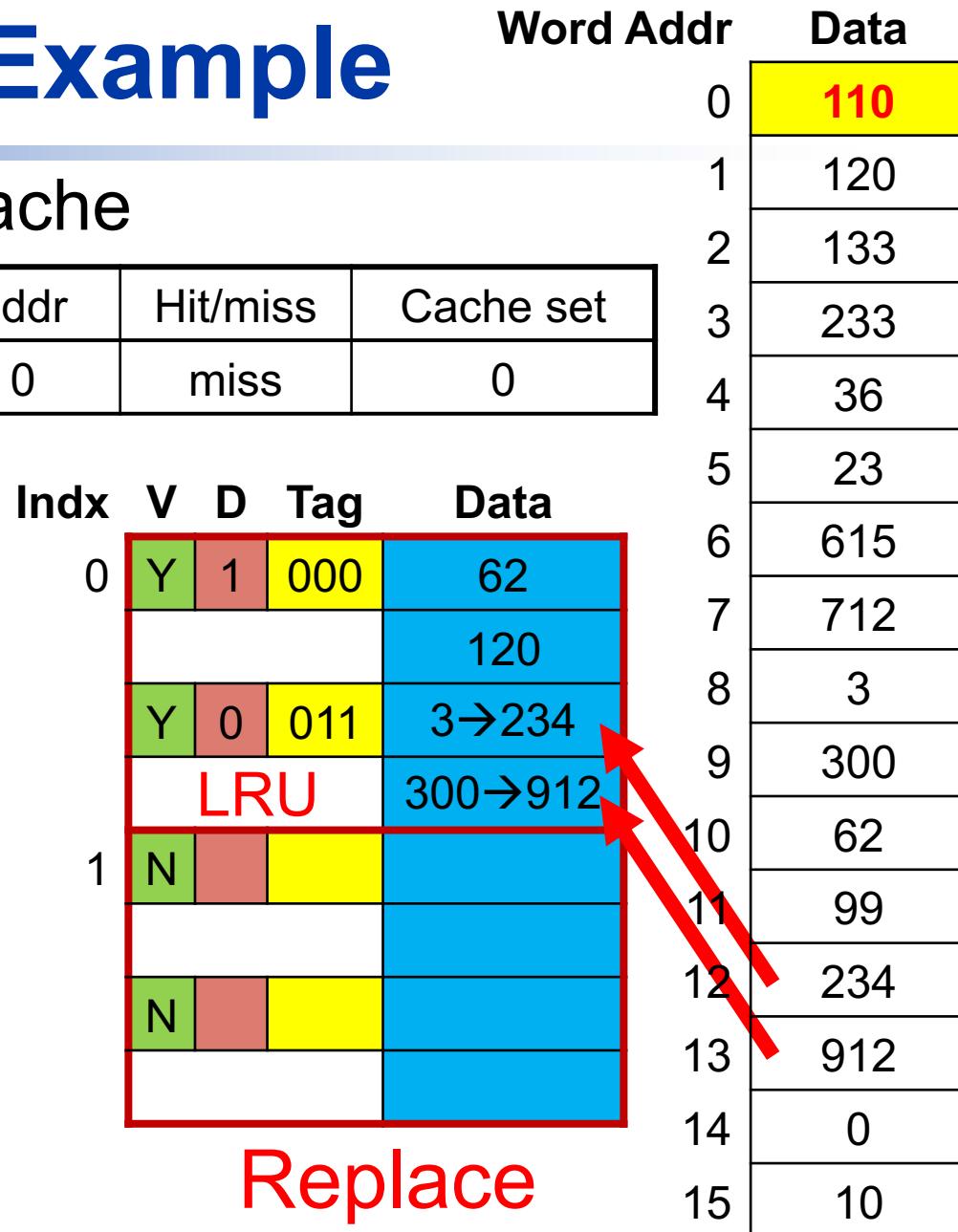
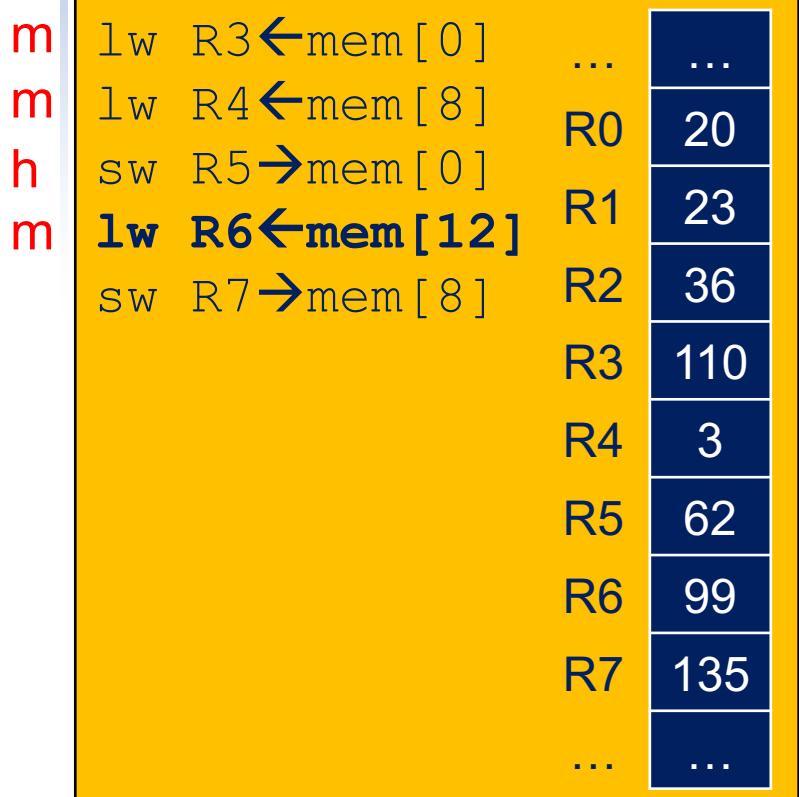
Replacement Policy

- Direct mapped: no choice
- Set associative
 - Prefer non-valid entry, if there is one
 - Otherwise, choose among entries in the set
- Choosing policy
 - ***Least-recently used (LRU)***
 - Choose the one unused for the longest time
 - Need a tracking mechanism for usage
 - Simple for 2-way, manageable for 4-way, too hard beyond that
 - Random
 - Gives approximately the same performance as LRU for high associativity

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	011 0 0	miss	0

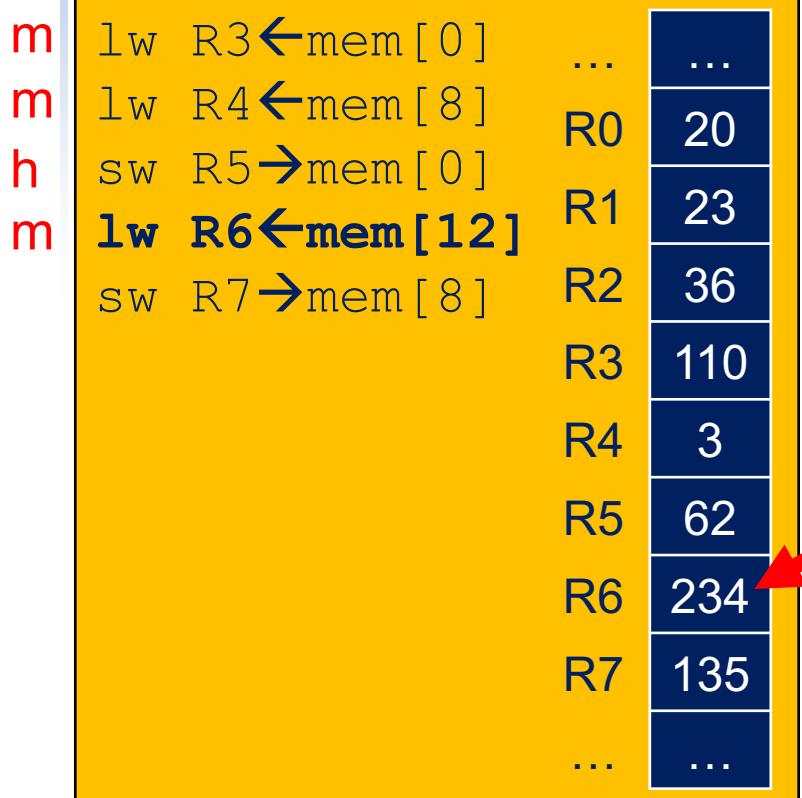


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	011 0 0	hit	0



Indx	V	D	Tag	Data
0	Y	1	000	62
				120
	Y	0	011	234
				912
1	N			
	N			

Load again

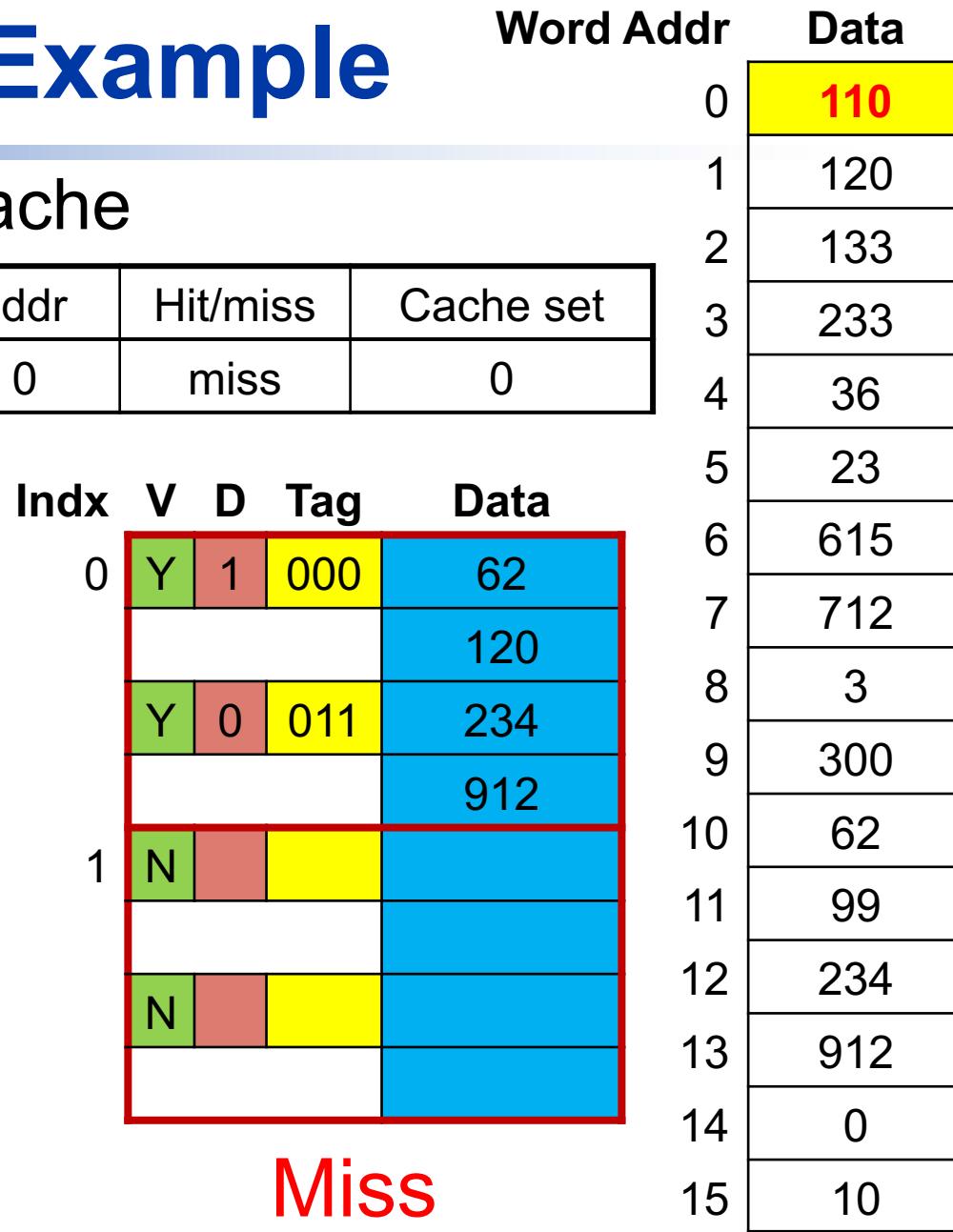
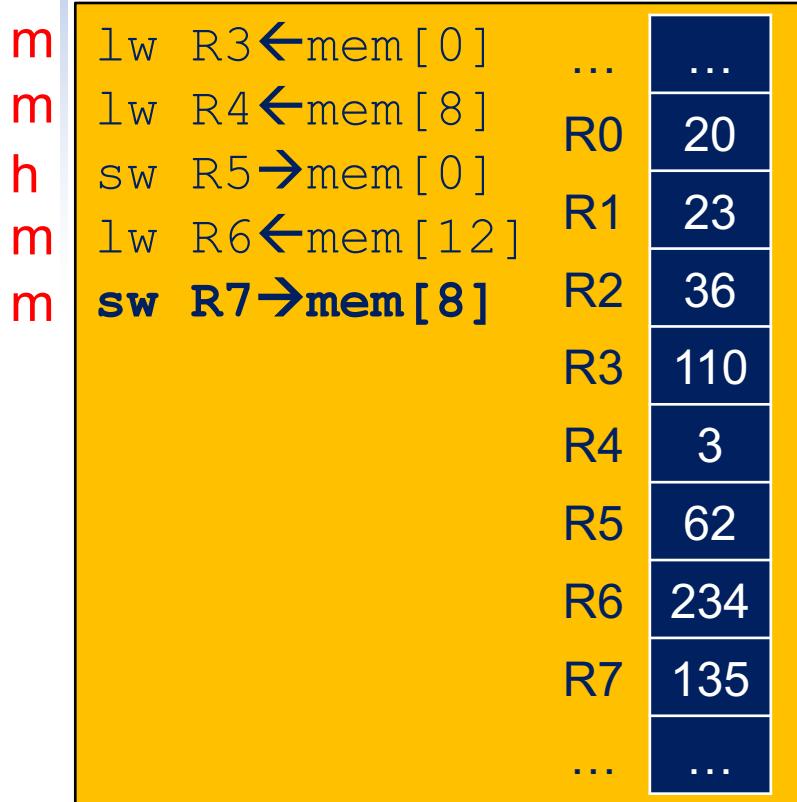
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 0 0	miss	0

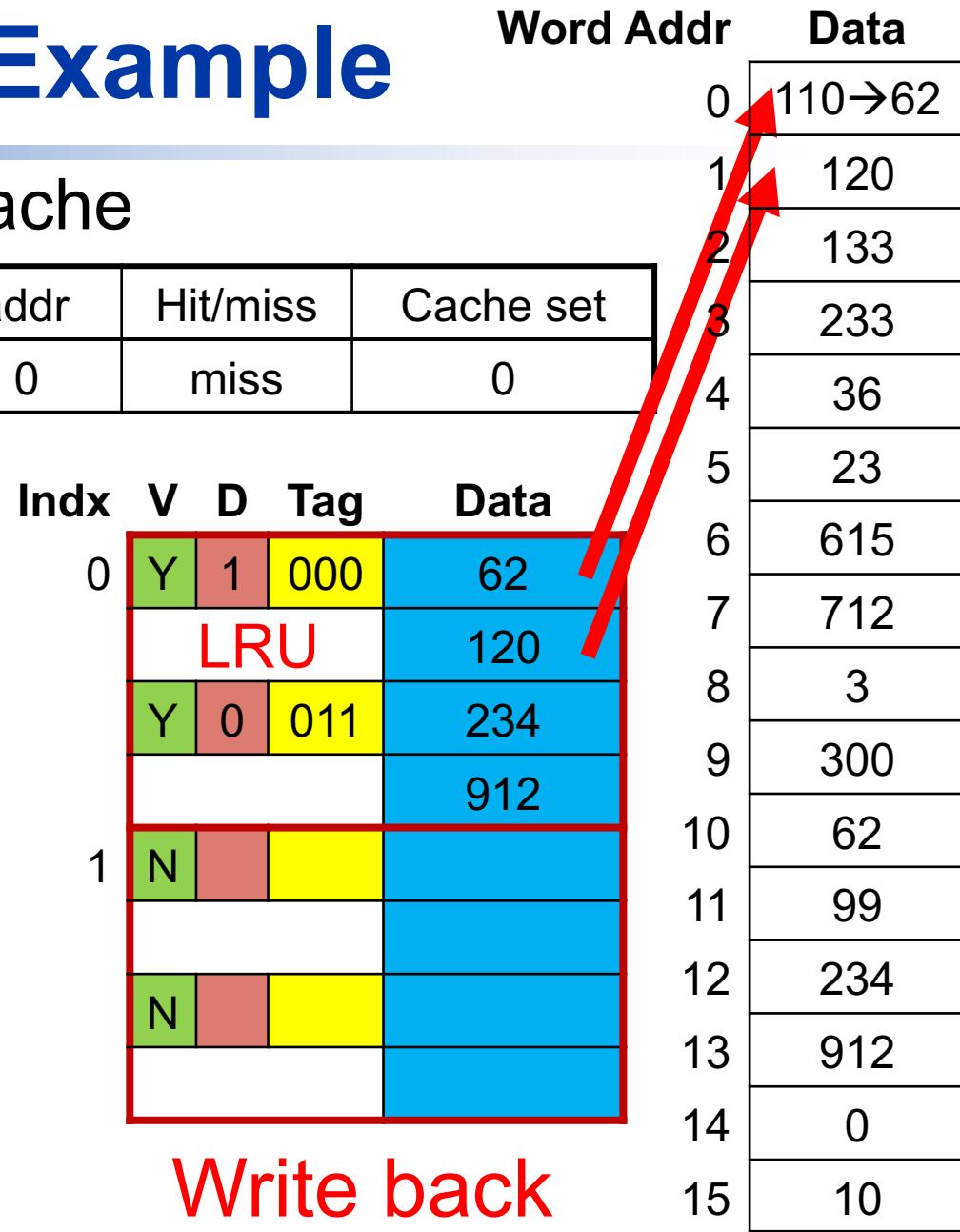
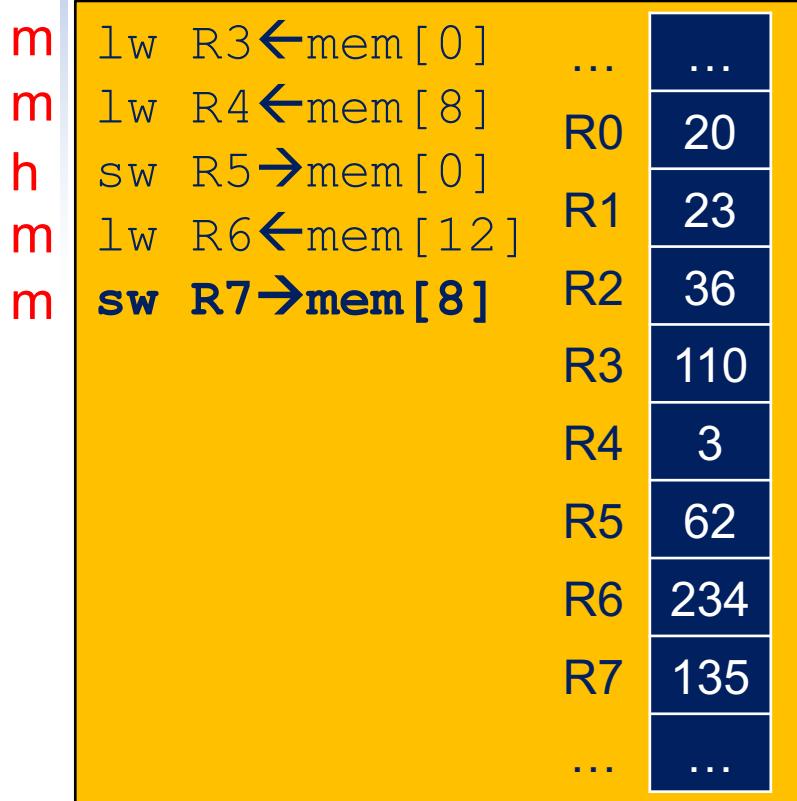


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 0 0	miss	0

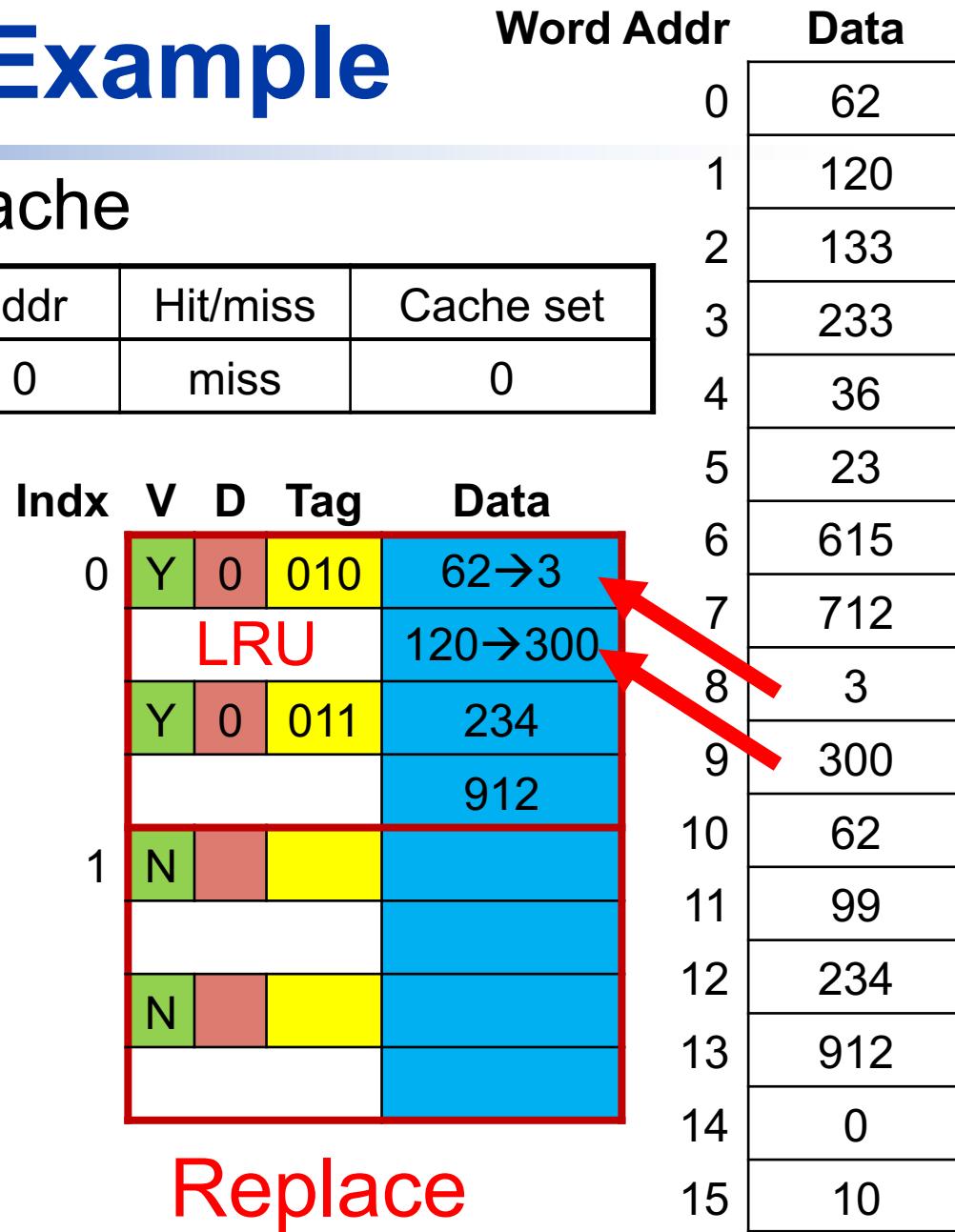
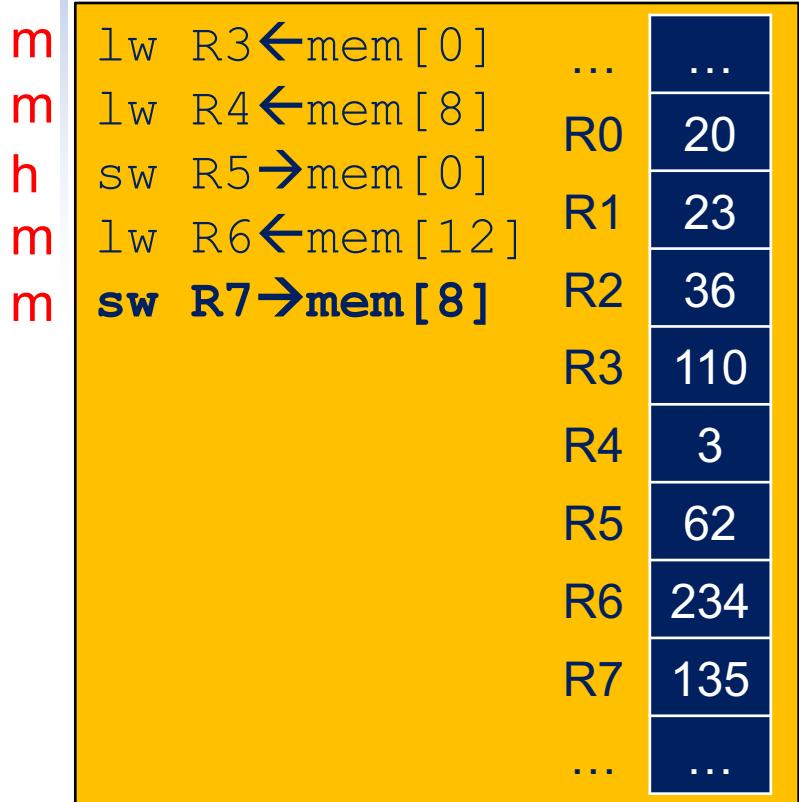


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 0 0	miss	0

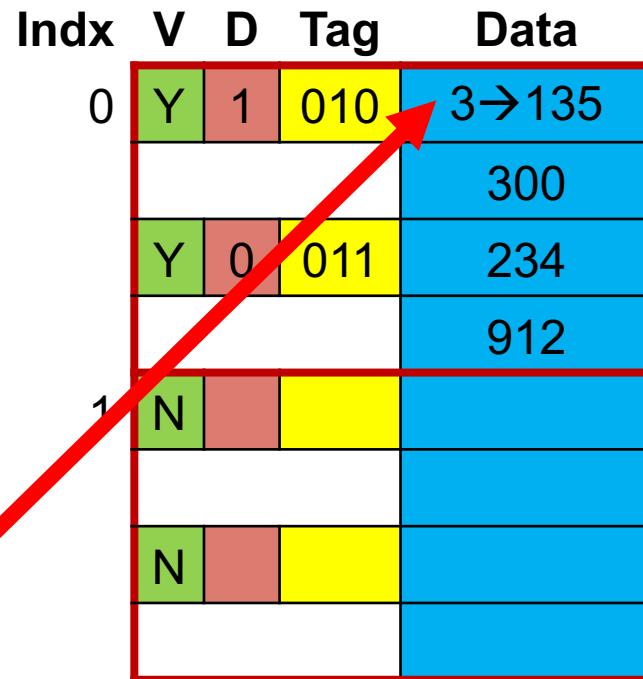
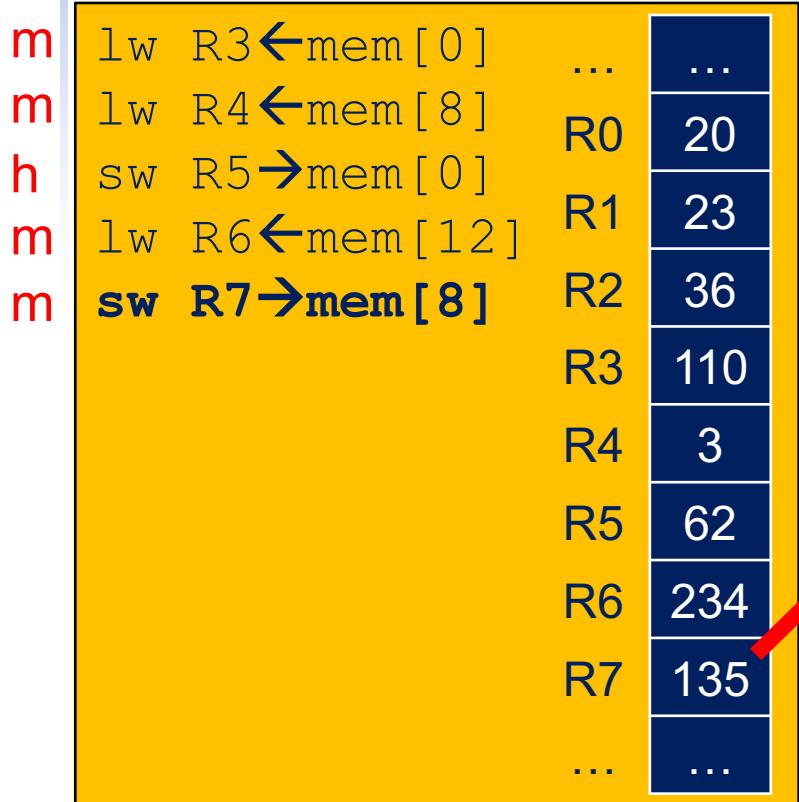


CPU

Associativity Example

- 2-way associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	010 0 0	miss	0



Write, set dirty

Word Addr	Data
0	62
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Associativity Example

Fully associative (4-way associative)

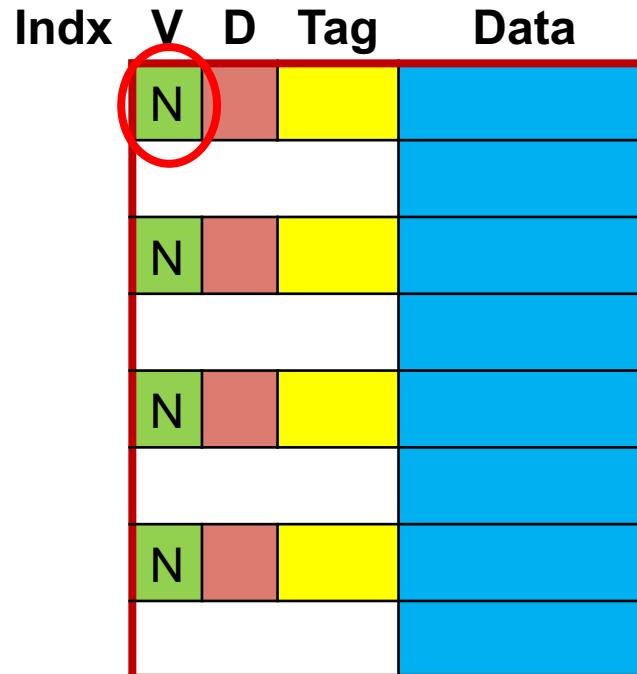
Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	0000 0	miss	-

m

lw R3 ← mem[0]
lw R4 ← mem[8]	R0	20
sw R5 → mem[0]	R1	23
lw R6 ← mem[12]	R2	36
sw R7 → mem[8]	R3	23
	R4	87
	R5	62
	R6	99
	R7	135



Miss

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	0000 0	miss	-

m

lw R3 <mem[0]
lw R4 <mem[8]	R0	20
sw R5 →mem[0]	R1	23
lw R6 <mem[12]	R2	36
sw R7 →mem[8]	R3	23
	R4	87
	R5	62
	R6	99
	R7	135

Indx	V	D	Tag	Data
0	Y	0	0000	110
1	N			120
2	N			
3	N			
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Fetch

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10



CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	0000 0	hit	-

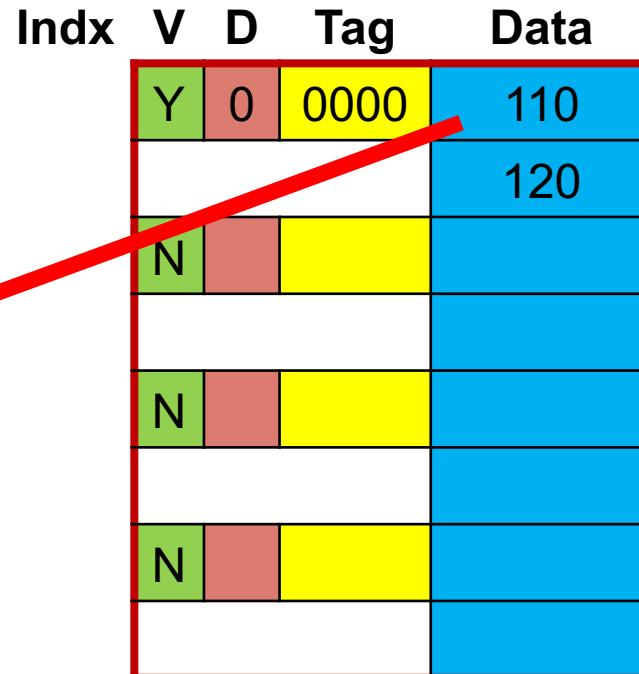
m

```

lw R3<-mem[0]
lw R4<-mem[8]
sw R5->mem[0]
lw R6<-mem[12]
sw R7->mem[8]

```

...	...
R0	20
R1	23
R2	36
R3	110
R4	87
R5	62
R6	99
R7	135
...	...



Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

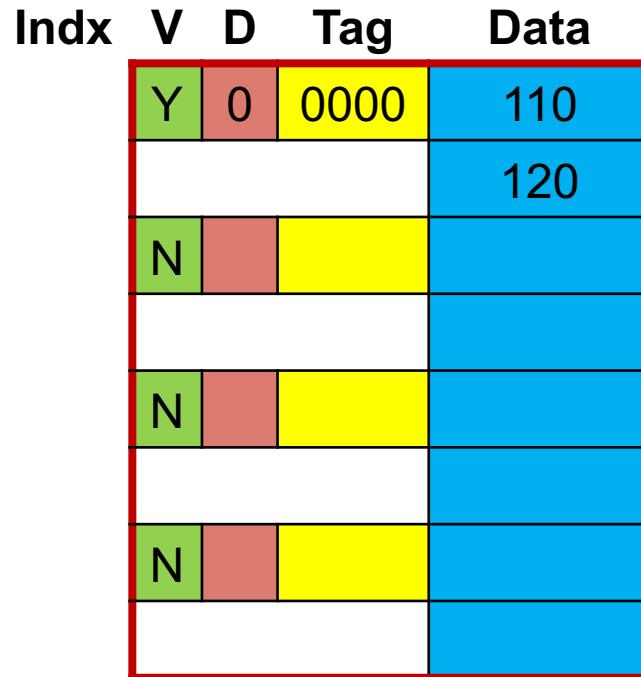
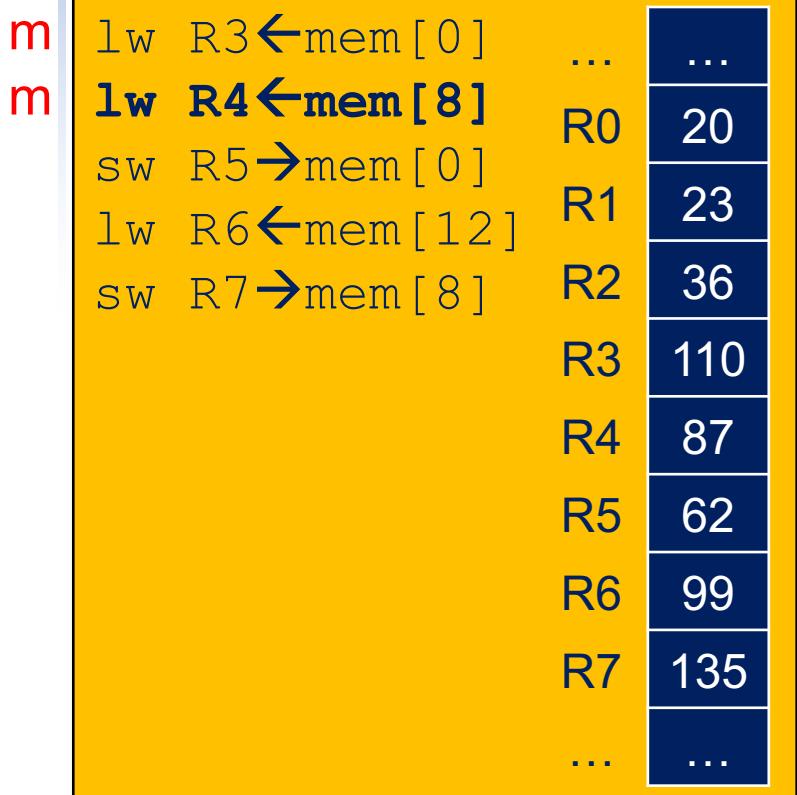


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	0100 0	miss	-



miss

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

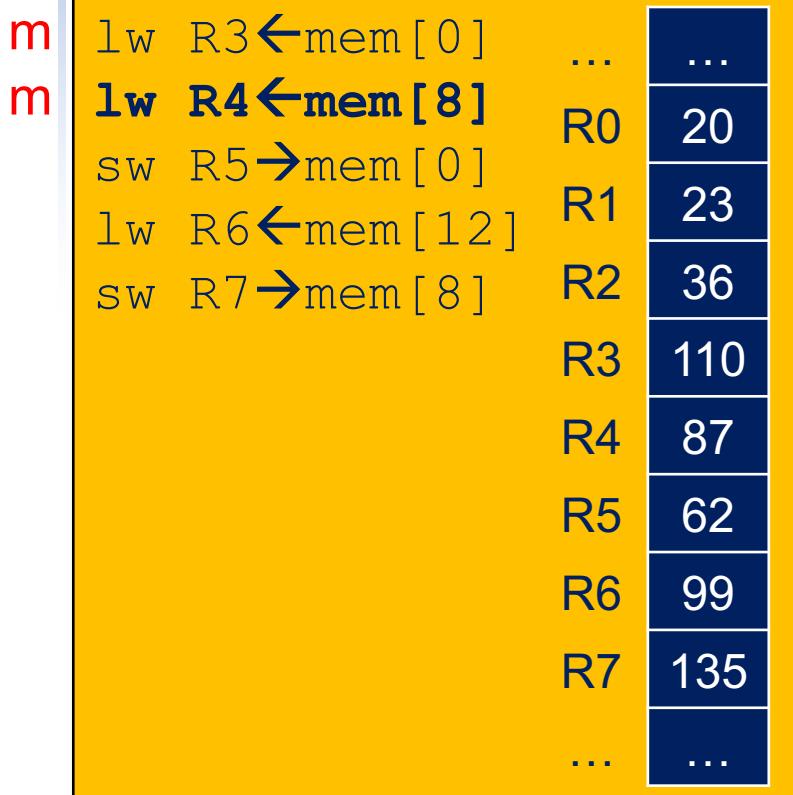


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	0100 0	miss	-



Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

Indx V D Tag Data

Y	0	0000	110
			120
Y	0	0100	3
			300
N			
N			

Fetch, not replace

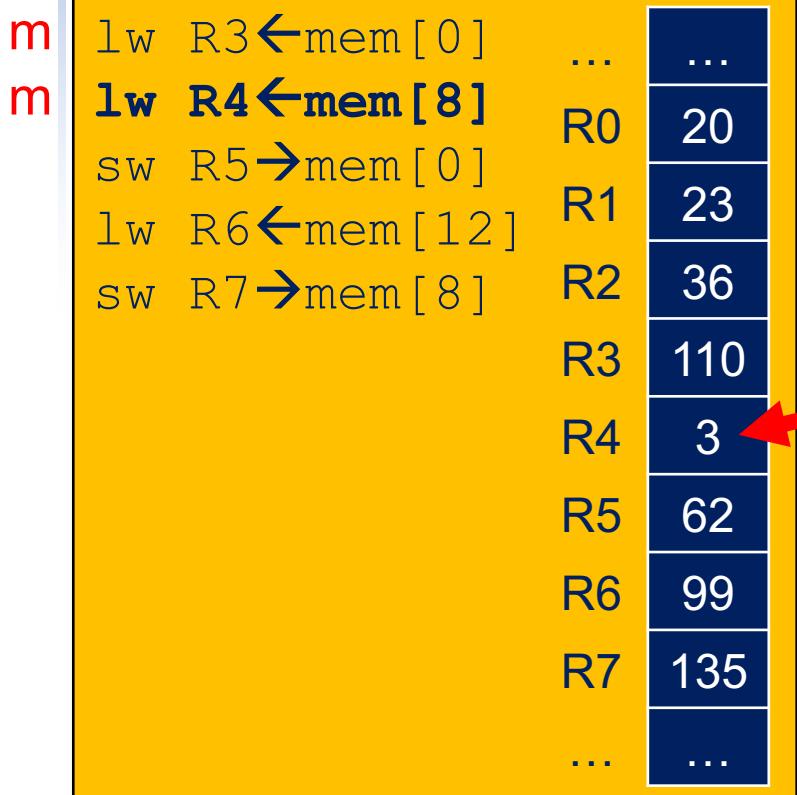


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	0100 0	hit	-



Indx	V	D	Tag	Data
Y	0	0000	110	110
			120	120
Y	0	0100	3	3
			300	300
N				
N				

Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

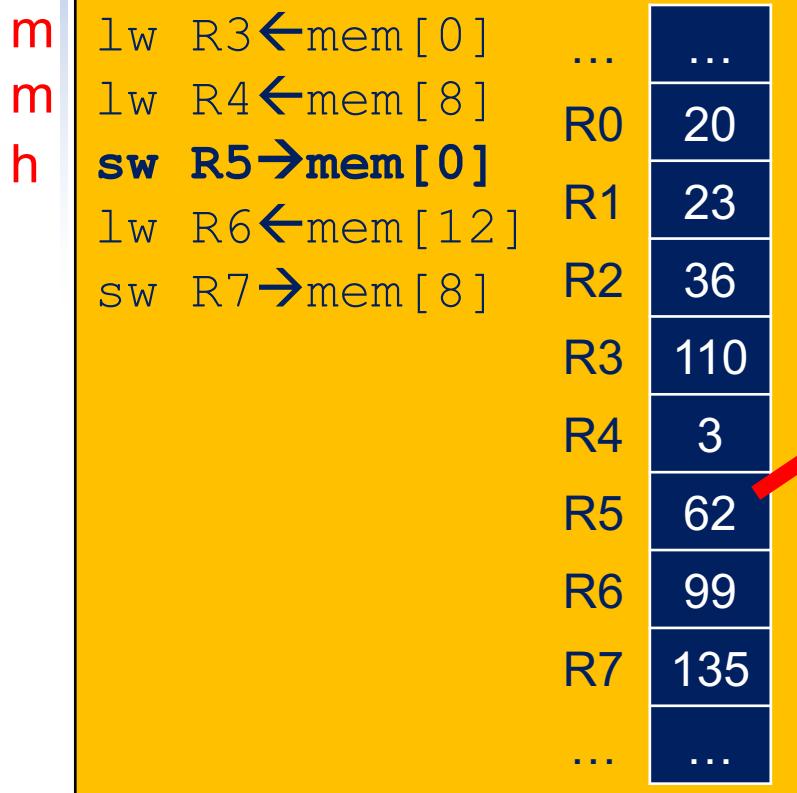


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
00000 00	0000 0	hit	-



Indx	V	D	Tag	Data
	Y	1	0000	110 → 62
	Y	0	0100	120
	N			3
	N			300

Write, set dirty

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

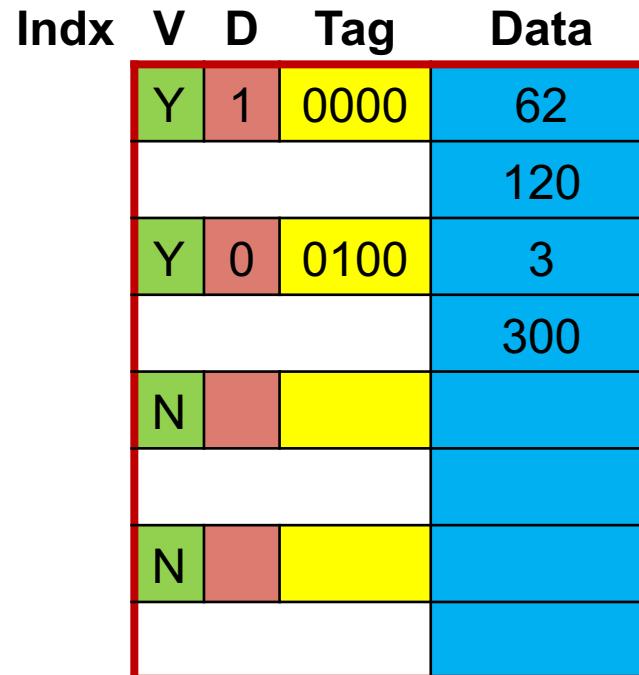
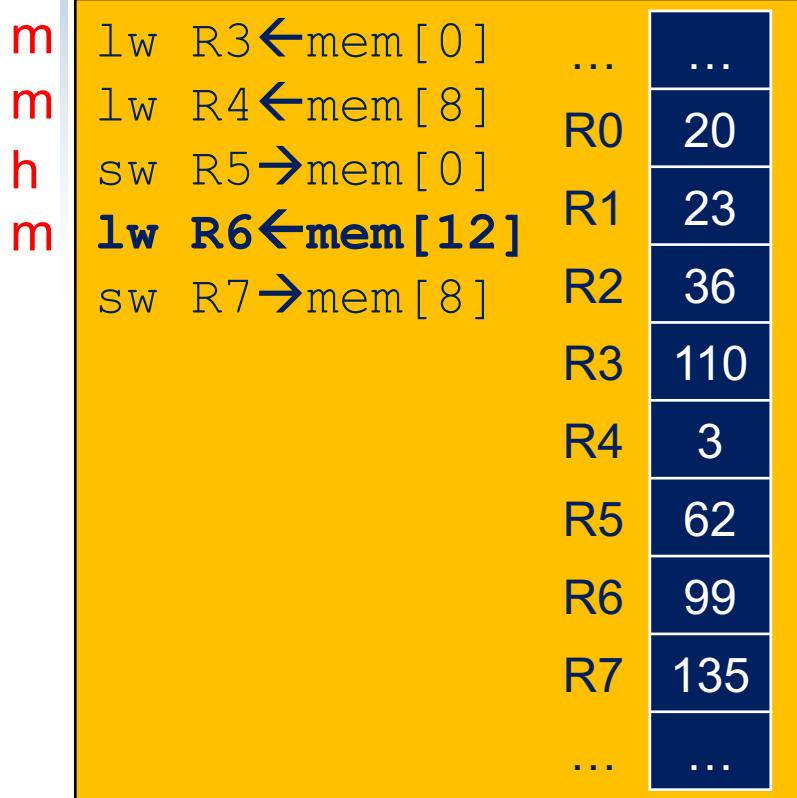


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	0110 0	miss	-



Miss

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

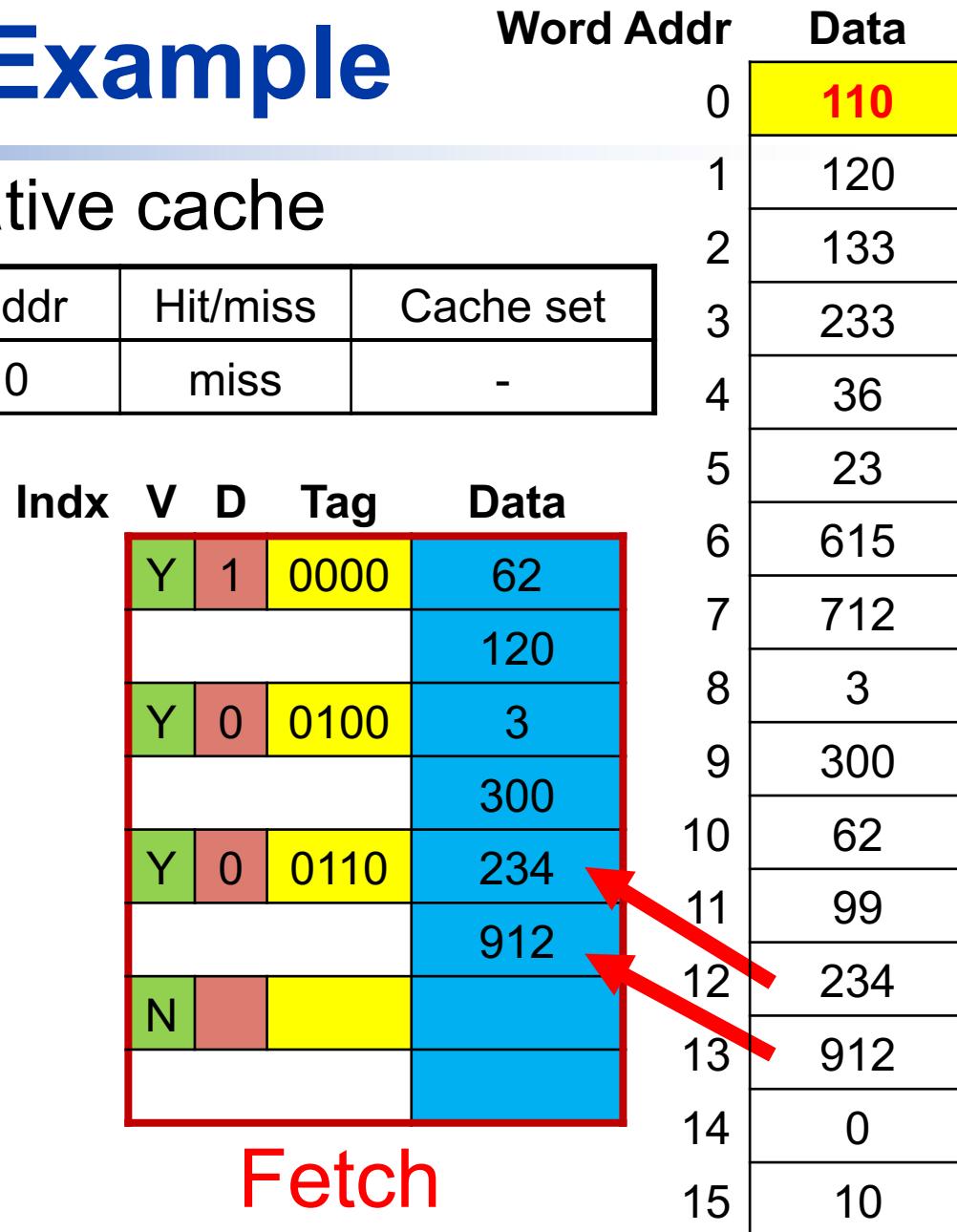
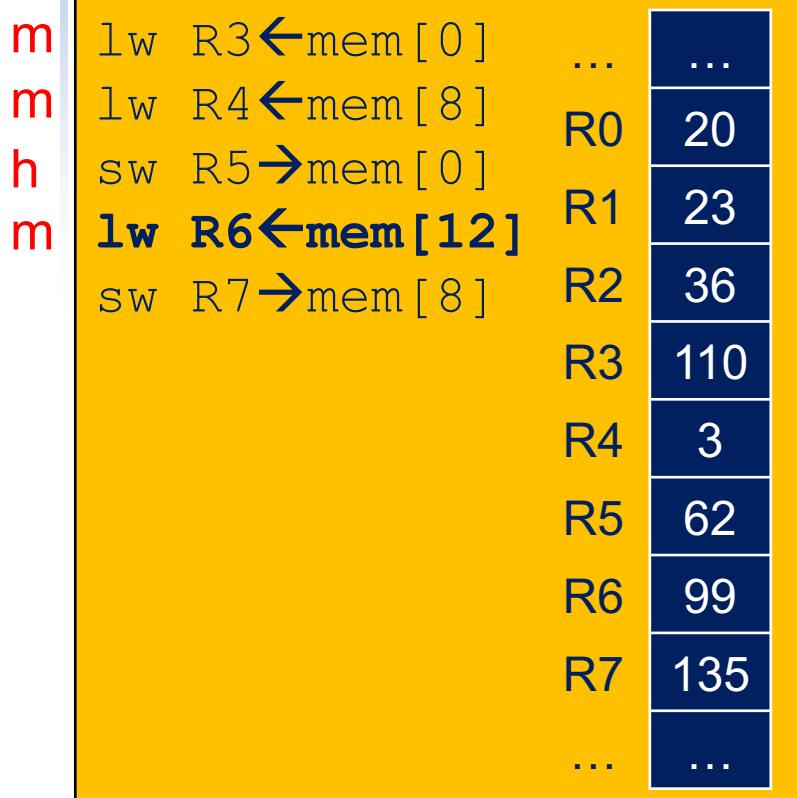


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	0110 0	miss	-

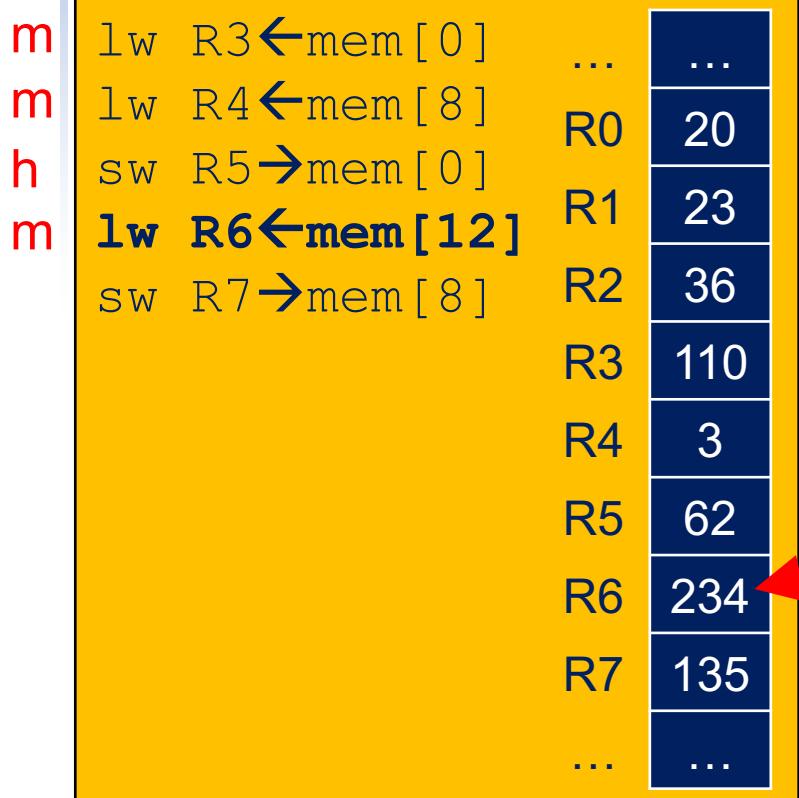


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01100 00	0110 0	hit	-



Indx	V	D	Tag	Data
	Y	1	0000	62
				120
	Y	0	0100	3
				300
	Y	0	0110	234
				912
	N			

Load again

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

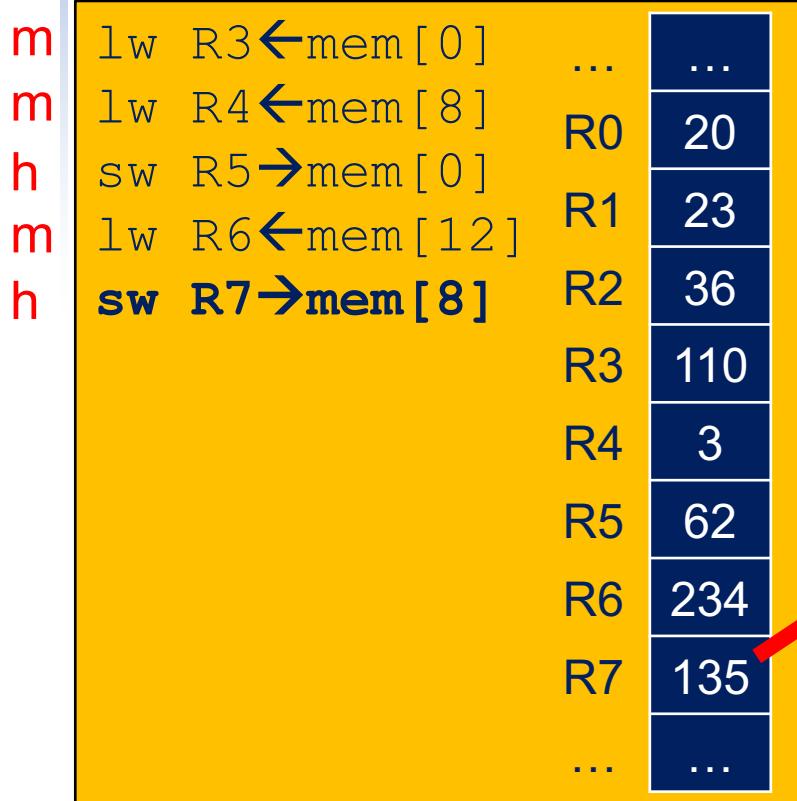


CPU

Associativity Example

- 4-way (fully) associative cache

Requested mem addr	Word addr	Hit/miss	Cache set
01000 00	0100 0	hit	-



Indx	V	D	Tag	Data
Y	1	0000	62	
			120	
Y	1	0100	3 → 135	
			300	
Y	0	0110	234	
			912	
N				

Write, set dirty

Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

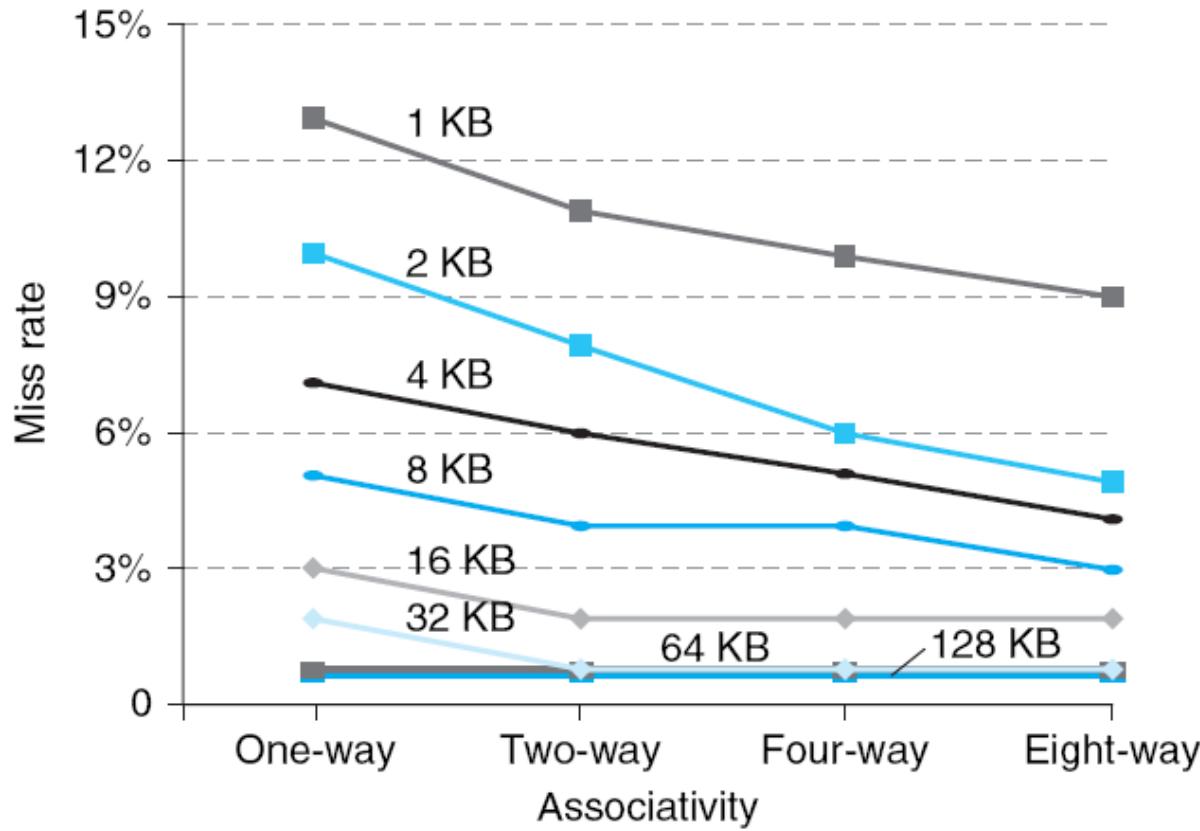


CPU

How Much Associativity

- *Increased associativity decreases miss rate*
 - But with diminishing improvement
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
 - 1-way: 10.3%
 - 2-way: 8.6%
 - 4-way: 8.3%
 - 8-way: 8.1%

How Much Associativity

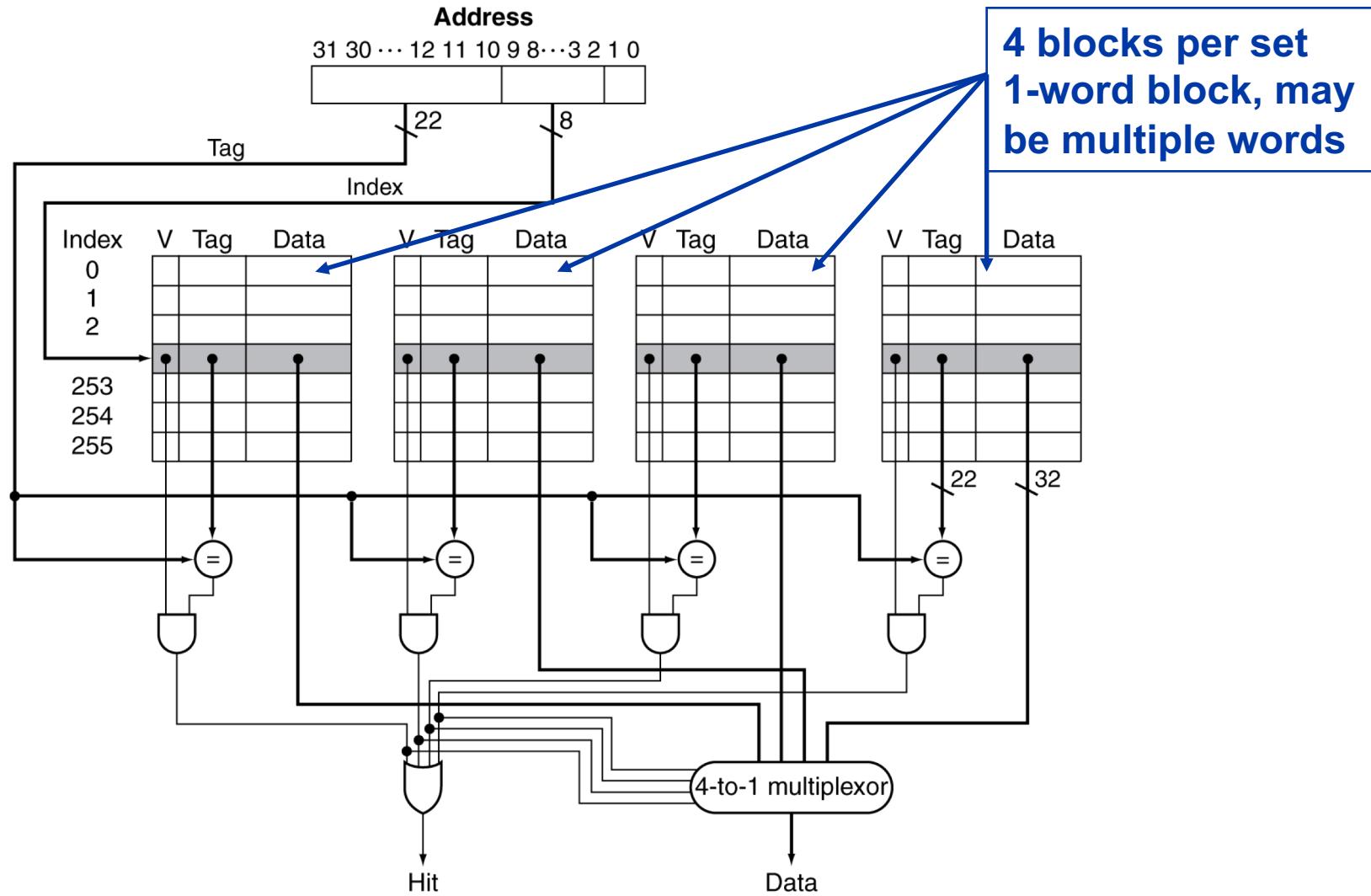


Locating a Block

Memory address	Tag	Index	Word/Byte offset
----------------	-----	-------	------------------

- Memory address decomposition
 - Index – locate a set in cache
 - Tag – upper address bits to locate block
 - Word/Byte offset – to locate a word/byte in a block
- Size of index field
 - Increasing degree of associativity decreases the number of sets, decreases number of bits for index, increases tag field
 - Doubling # of blocks by 2 halves # of set by 2
 - Reduce index bits by 1
 - Increase tag bits by 1
- All blocks in a set must be searched
 - Tag field compared in parallel
 - Extra hardware and **extra access (hit) time**

Set Associative Cache Organization



Class Exercise

- Given
 - 2K blocks in cache
 - 4-way associative
 - 8 words in each block
 - 32-bit byte address 0x810023FE requested by CPU
- Show address and organization of the target cache block

Improve Performance – Multilevel Caches

- ***Multilevel cache decreases miss penalty***
- Primary (L-1) cache attached to CPU
 - Small, but fast
- Level-2 (secondary) cache services misses from primary cache
 - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given

- CPU base CPI = 1, clock rate = 4GHz
 - Miss rate (misses/instruction) = 2%
 - Main memory access time = 100ns
 - As miss penalty, ignoring other times

- With one-level cache

- Miss penalty = $100\text{ns}/0.25\text{ns} = 400$ cycles
 - Effective CPI = $1 + 0.02 \times 400 = 9$

Example (cont.)

- Now add L-2 cache
 - Access time = 5ns (L-1 miss penalty)
 - Miss rate for L-2 = 25% of L1 misses (have to access main memory)
 - L-1 cache miss have a miss on L-2
- Primary (L-1) cache miss with L-2 hit
 - Miss penalty = $5\text{ns}/0.25\text{ns} = 20 \text{ cycles}$
- Primary cache miss with L-2 miss main memory hit
 - Extra penalty = 400 cycles
- CPI = base CPI + L-1 miss L-2 hit (cycles per instruction)
+ L-1 miss L-2 miss (cycles per instruction)
 - CPI = $1 + 0.02 \times 75\% \times 20 + 0.02 \times 25\% \times (20+400) = 3.4$
- Performance ratio = $9/3.4 = 2.6$

Multilevel Cache Considerations

- Primary cache

- Focus on minimal hit time because miss penalty is smaller
- And to reduce CPU clock cycle

- Secondary cache

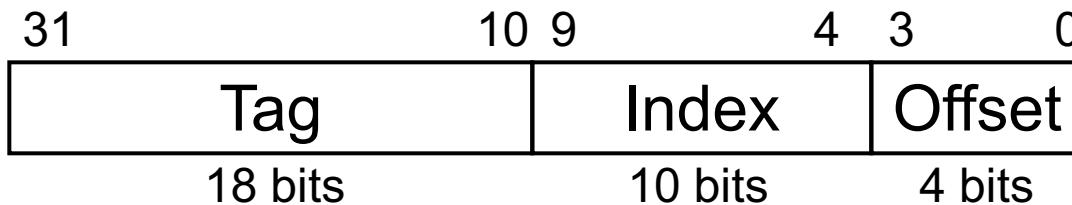
- Focus on low miss rate to avoid main memory access
- Hit time has less overall impact

Multilevel Cache Considerations

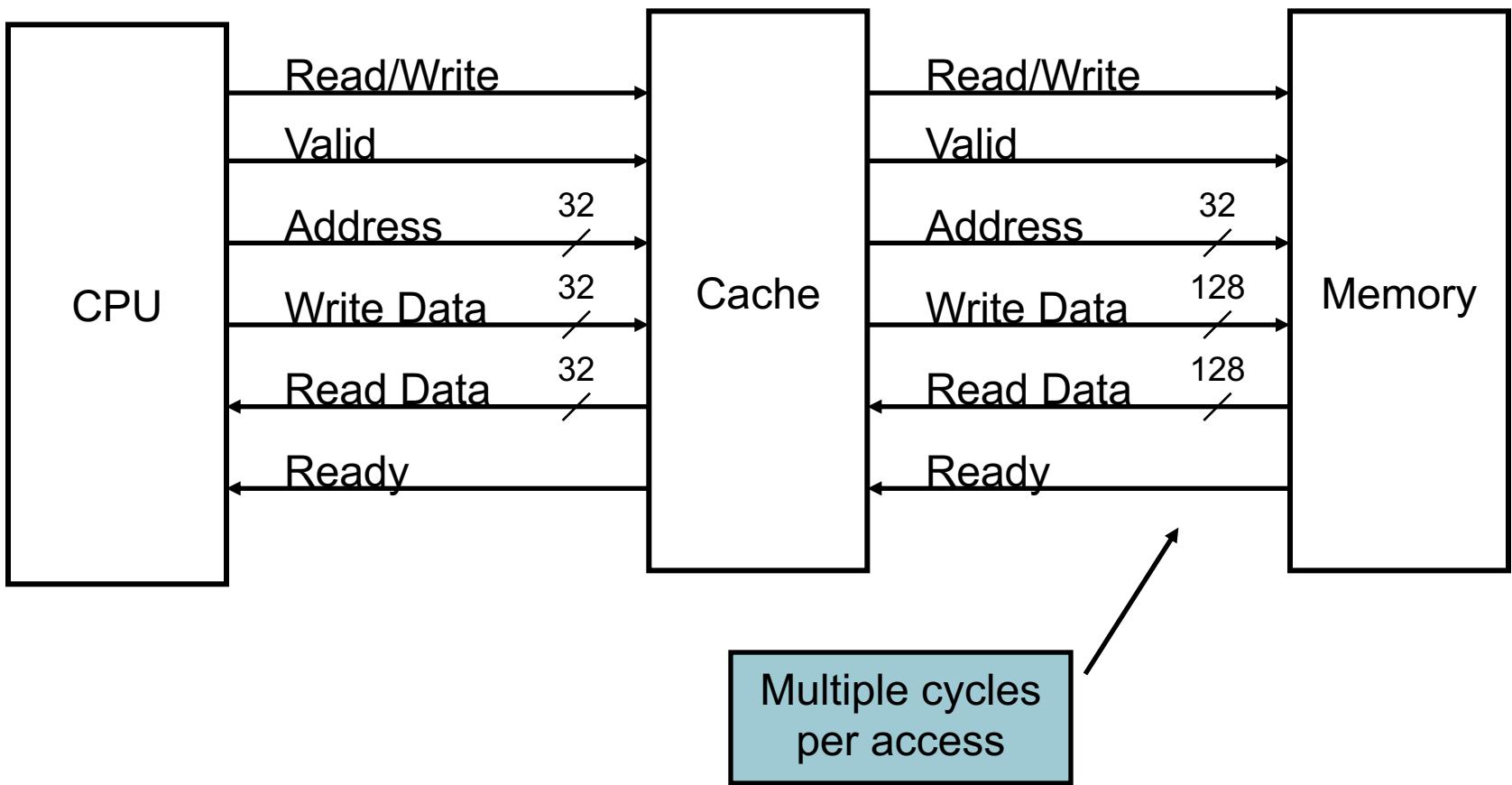
- Comparison with single level cache
 - L-1
 - Smaller cache size
 - Smaller block size, because of
 - Smaller total cache size
 - Reduced search time -> reduced hit time
 - Reduced miss penalty -> less time to fetch
 - L-2
 - Cache and block size much larger
 - because of less critical hit time
 - Higher associativity and block size to reduce miss rate
 - Because miss penalty is more severe

Cache Controller

- Example cache characteristics
 - Direct-mapped, write-back, write allocate
 - Block size: 4 words (16 bytes)
 - Cache size: 16 KB (1024 blocks)
 - 32-bit byte addresses
 - Valid bit and dirty bit per block
 - Blocking cache
 - CPU waits until access is complete

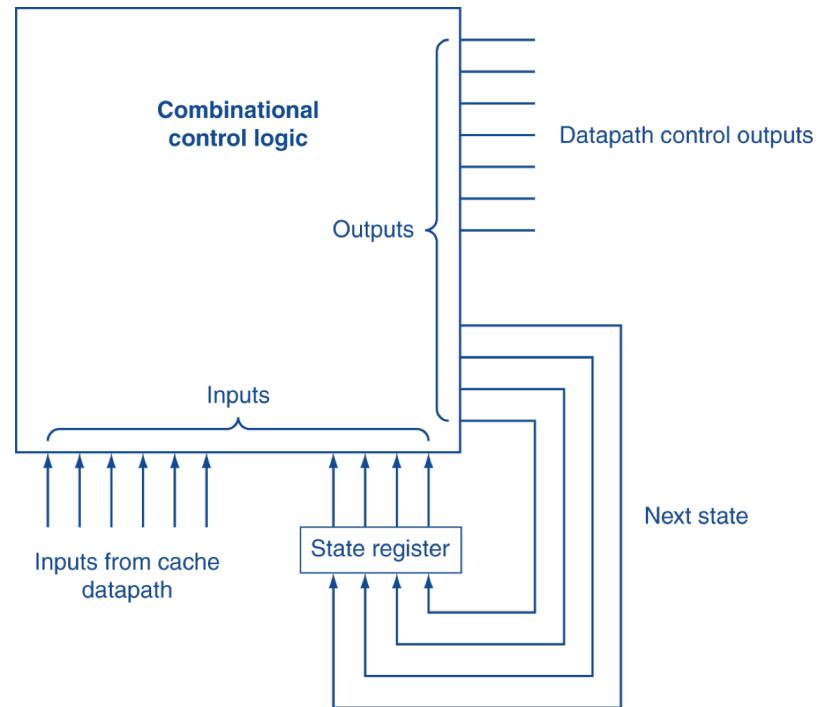


Interface Signals



Finite State Machines

- Use an FSM for sequence control steps
- Set of states, transition on each clock edge
 - State values are binary encoded
 - Current state stored in a register
 - Next state = f_n (current state, current inputs)
- Control output signals = f_o (current state)



Cache Controller FSM

