Topic 10

Memory Hierarchy

- Cache (2)

Block Size Considerations

- ***Larger blocks should reduce miss rate***
  - Due to spatial locality
- But increased miss rate 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

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

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 x0 → 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

match

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**	**00**	**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

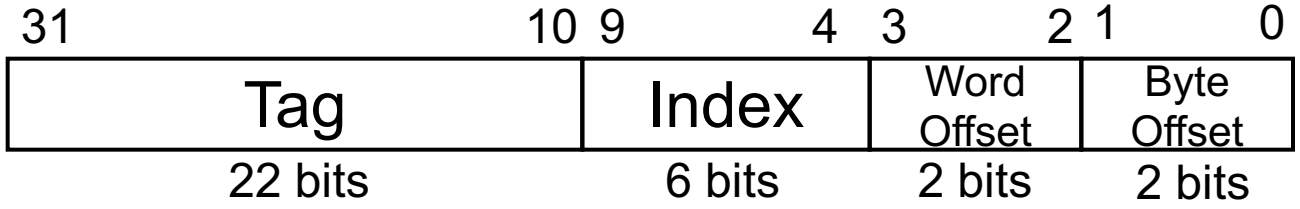
Index	V	Tag	Word 0	Word 1
00	N			
01	**Y**	**11**	**Mem[26]**	**Mem[27]**
10	N			
11	**Y**	**10**	**Mem[22]**	**Mem[23]**

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 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 = $2^n \times$ (block size + tag size + valid field size)

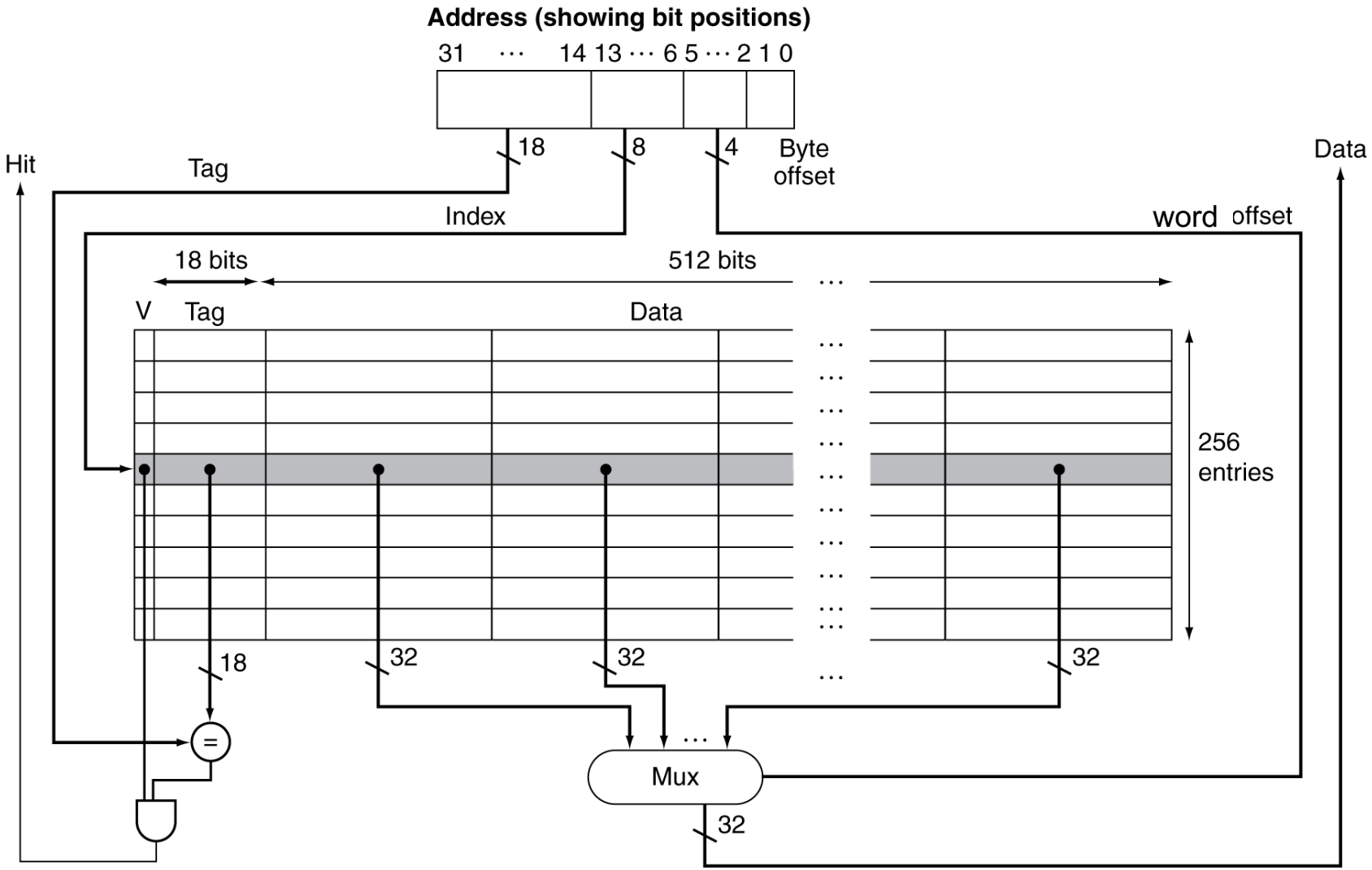
Class Exercise

- Given
  - 2K blocks in cache
  - 8 words in each block
  - 32-bit byte address 0x810023FE requested by CPU
- Show organization of the entire cache, and locate the block the address is mapped to

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



Miss in Instruction Cache

1. Send original PC 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

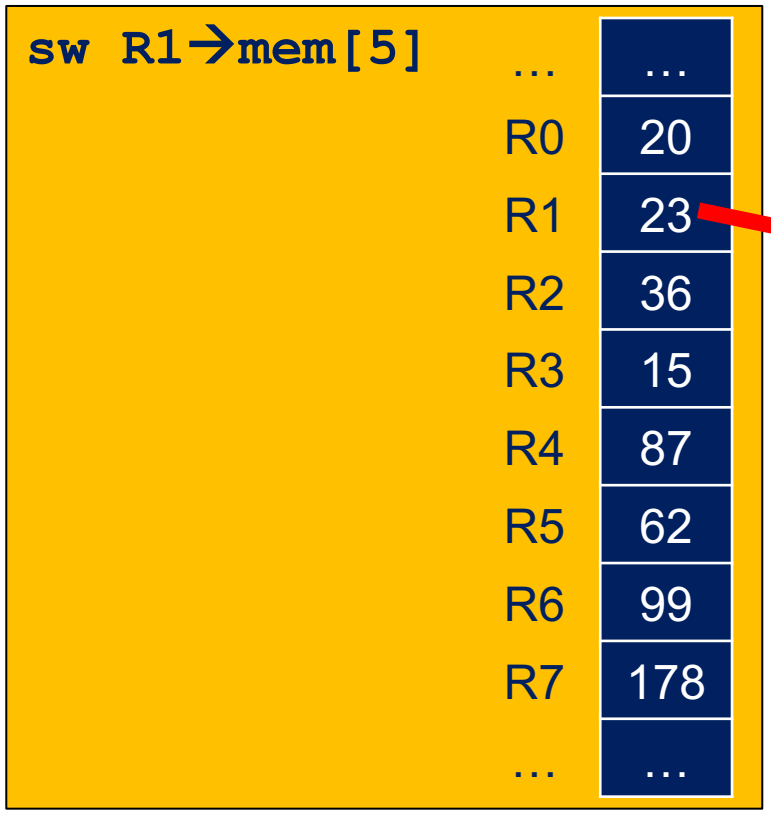
1. Hold the pipeline
2. Read main (lower level) memory and wait for data
3. Write data into cache data field from main memory, write upper bits of address into cache tag field, set valid field
4. Read cache again, proceed

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



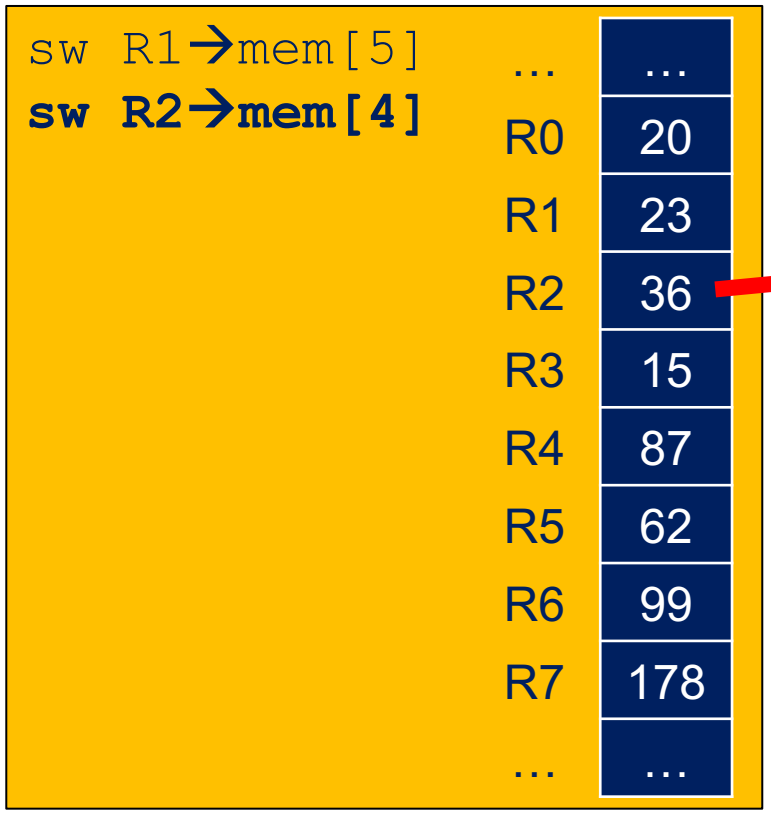
CPU

Index	V	Tag	Data
0	Y	001	140
			141→23
1	N		

Word Addr	Data
0	110
1	120
2	133
3	233
4	140
5	141→23
6	615
7	712
8	3
9	300
10	531
11	153
12	234
13	912
14	0
15	10

Write Through Example

Requested mem addr	Word addr	Hit/miss	Cache block
00100 00	001 0 0	hit	0



CPU

Index	V	Tag	Data
0	Y	001	140→36
			23
1	N		

Word Addr	Data
0	110
1	120
2	133
3	233
4	140→36
5	23
6	615
7	712
8	3
9	300
10	531
11	153
12	234
13	912
14	0
15	10

Handling Data Writes – Write Through

- But makes writes take longer time
  - Must wait till the update finishes
  - e.g., if base CPI = 1 (when everything is normal), 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = base CPI + write time (cycles) per instruction
    - Effective CPI = 1 + 10%×100 = 11

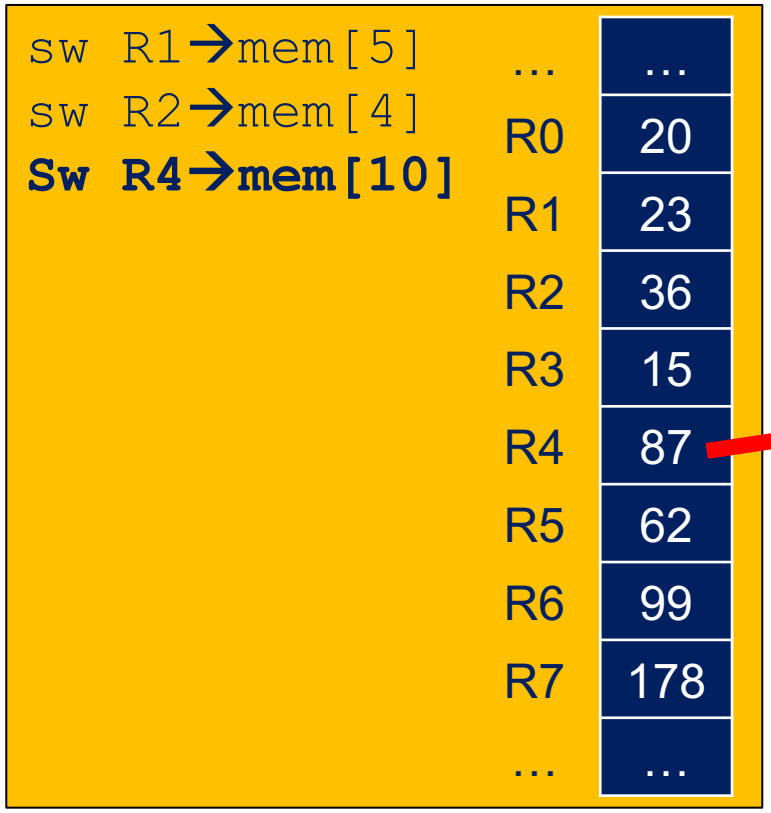
Handling Data Writes – Write Through

- 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

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



CPU

Index	V	Tag	Data
0	Y	001	36
			23
1	N		

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	531
11	153
12	234
13	912
14	0
15	10

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
…	…

CPU

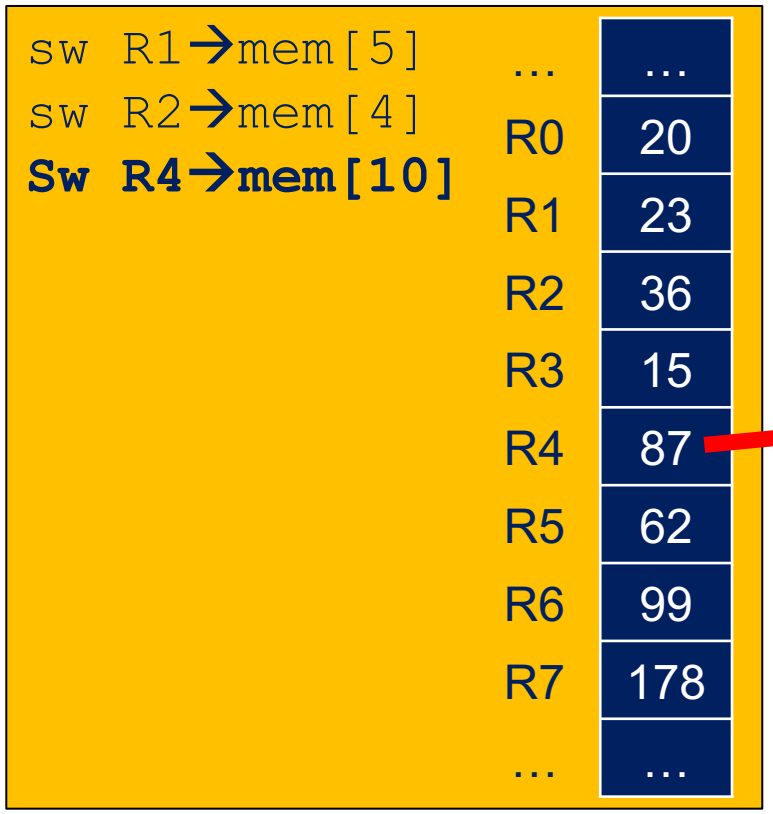
Index	V	Tag	Data
0	Y	001	36
			23
1	Y	010	531
			135

Fetch block

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

Write Through Example

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



CPU

Index	V	Tag	Data
0	Y	001	36
			23
1	Y	010	531→87
			135

Write Through

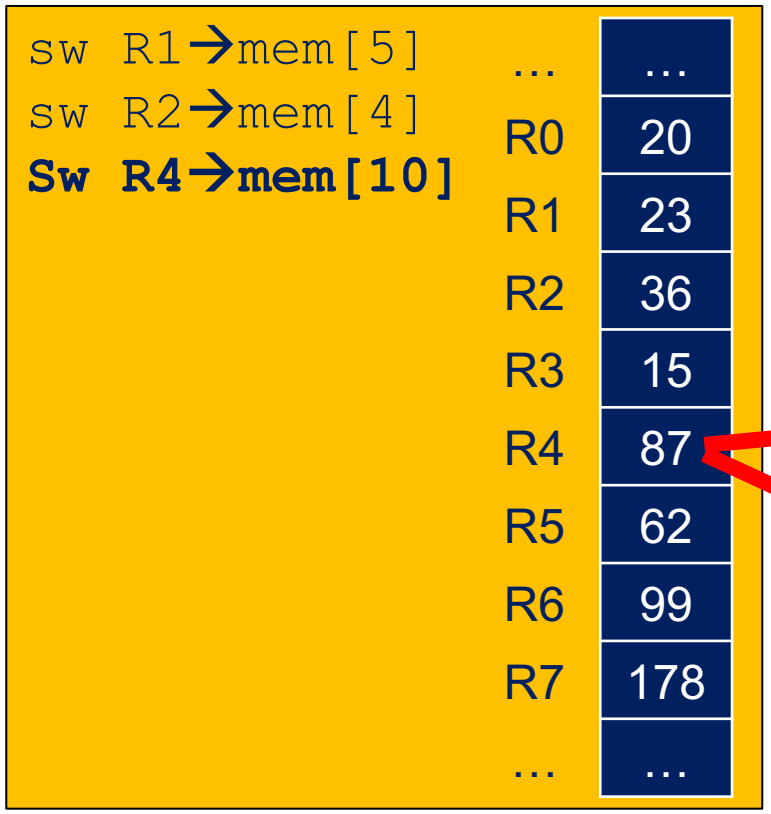
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	531→87
11	135
12	234
13	912
14	0
15	10

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

Write Through with Buffer

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



CPU

Index	V	Tag	Data
0	Y	001	36
			23
1	Y	010	531→87
			135

87

Buffer

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

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

```
lw R3←mem[5]
lw R7←mem[11]
sw R6→mem[11]
sw R5→mem[10]
sw R7→mem[6]
sw R0→mem[13]
```

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

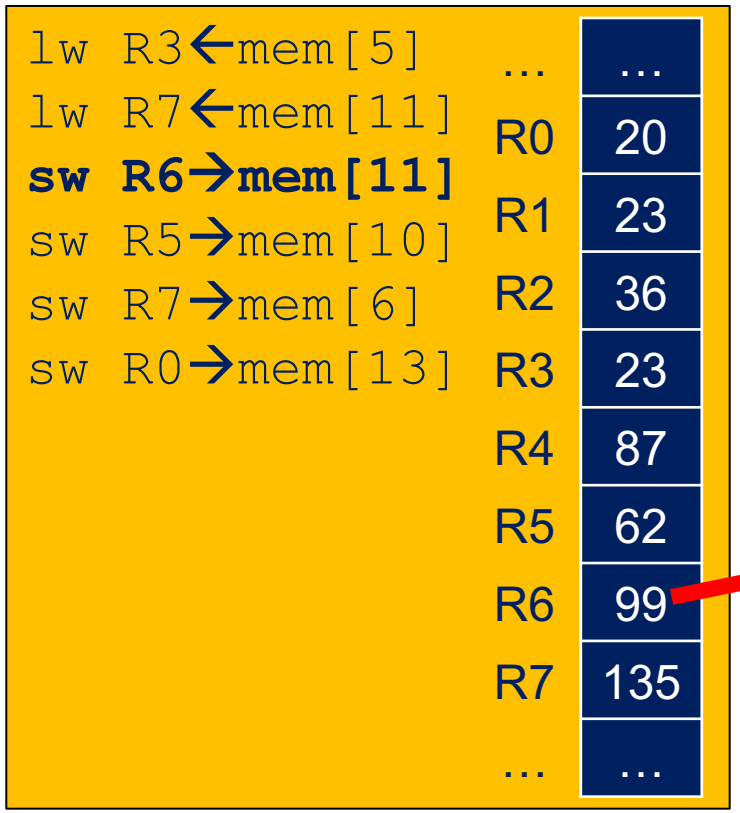
CPU

Indx	V	D	Tag	Data
0	Y	0	001	36
				23
1	Y	0	010	87
				135

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 Back Example

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



CPU

Indx	V	D	Tag	Data
0	Y	0	001	36
				23
1	Y	1	010	87
				135→99

Write cache, 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	87
11	135
12	234
13	912
14	0
15	10

Write Back Example

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

```
lw  R3←mem[5]
lw  R7←mem[11]
sw  R6→mem[11]
sw  R5→mem[10]
sw  R7→mem[6]
sw  R0→mem[13]
```

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

CPU

Indx	V	D	Tag	Data
0	Y	0	001	36
				23
1	Y	1	010	87→62
				99

Write cache, 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	87
11	135
12	234
13	912
14	0
15	10

Write Back Example

Requested mem addr	Word addr	Hit/miss	Cache block
00110 00	0011 0	miss	1

```
lw  R3←mem[5]
lw  R7←mem[11]
sw  R6→mem[11]
sw  R5→mem[10]
sw  R7→mem[6]
sw  R0→mem[13]
```

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

CPU

Miss match

Indx	V	D	Tag	Data
0	Y	0	001	36
				23
1	Y	1	010	62
				99

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

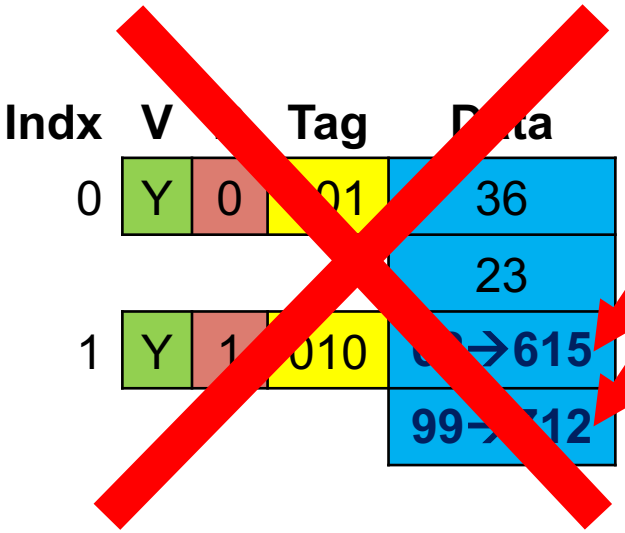
Write Back Example

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

```
lw  R3←mem[5]
lw  R7←mem[11]
sw  R6→mem[11]
sw  R5→mem[10]
sw  R7→mem[6]
sw  R0→mem[13]
```

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

CPU

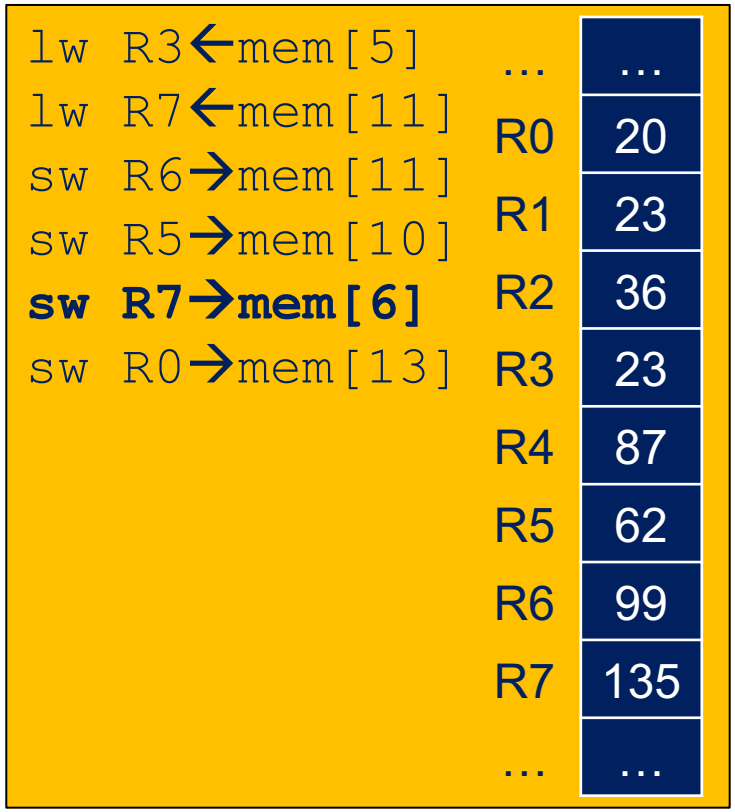


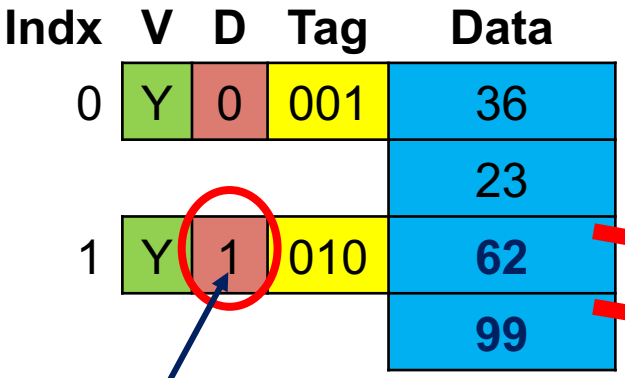
If replaced, content overwritten

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 Back Example

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



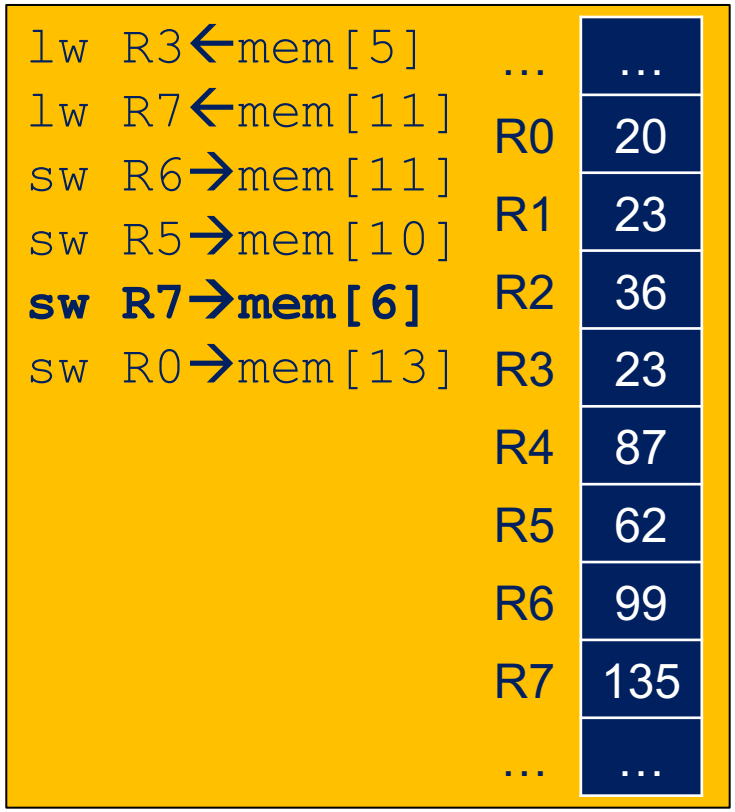


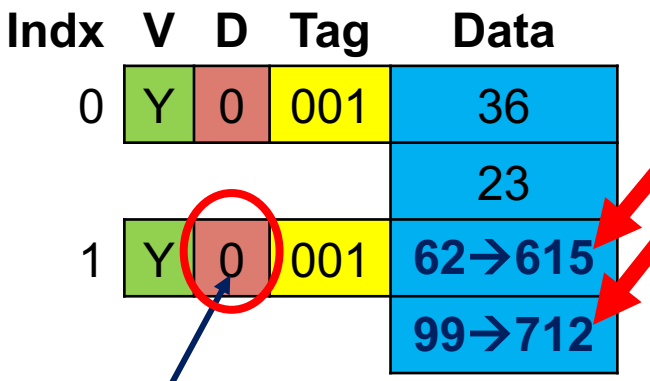
Dirty
Write back first!

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→62
11	135→99
12	234
13	912
14	0
15	10

Write Back Example

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



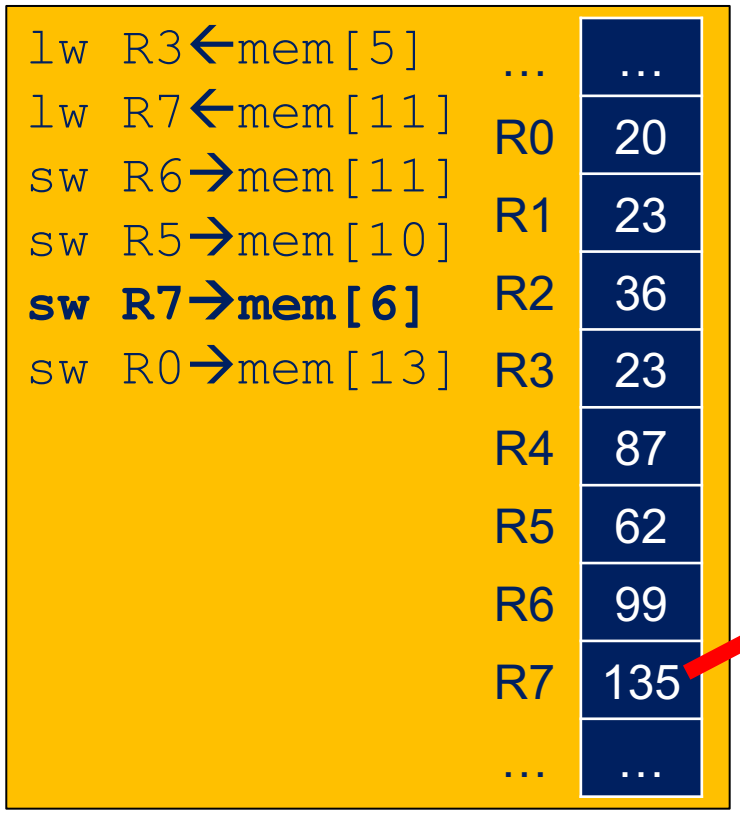


Replace now,
Reset dirty

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

Write Back Example

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



Indx	V	D	Tag	Data
0	Y	0	001	36
				23
1	Y	1	001	615→135
				712

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

Write Back Example

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

```
lw  R3←mem[5]
lw  R7←mem[11]
sw  R6→mem[11]
sw  R5→mem[10]
sw  R7→mem[6]
sw  R0→mem[13]
```

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

CPU

Miss match

Indx	V	D	Tag	Data
0	Y	0	001	36
				23
1	Y	1	001	135
				712

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

Write Back Example

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

```
lw R3←mem[5]
lw R7←mem[11]
sw R6→mem[11]
sw R5→mem[10]
sw R7→mem[6]
sw R0→mem[13]
```

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

CPU

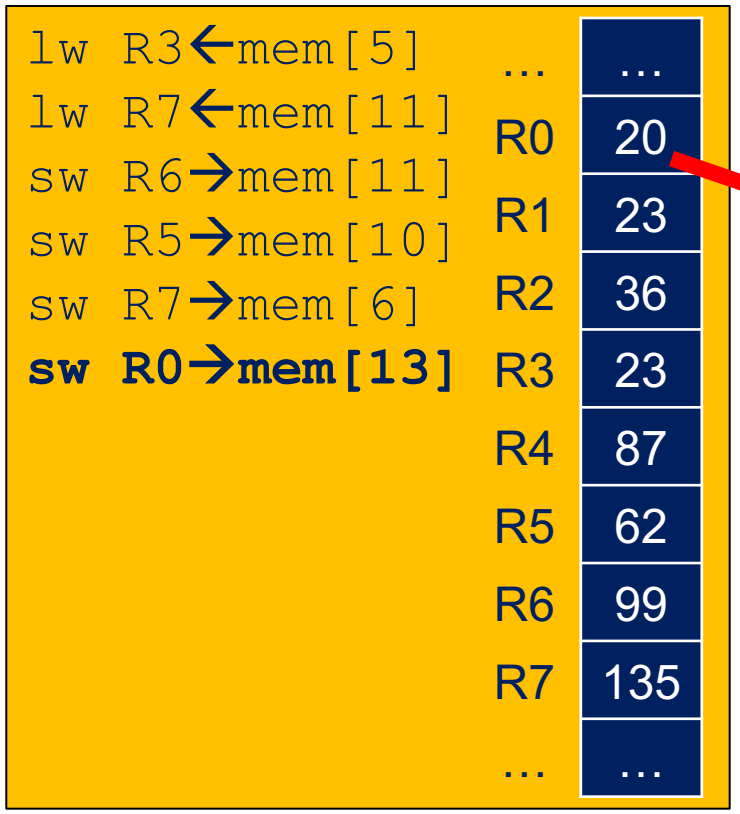
Indx	V	D	Tag	Data
0	Y	0	011	36→234
				23→912
1	Y	1	001	135
				712

Not Dirty,
Replace directly

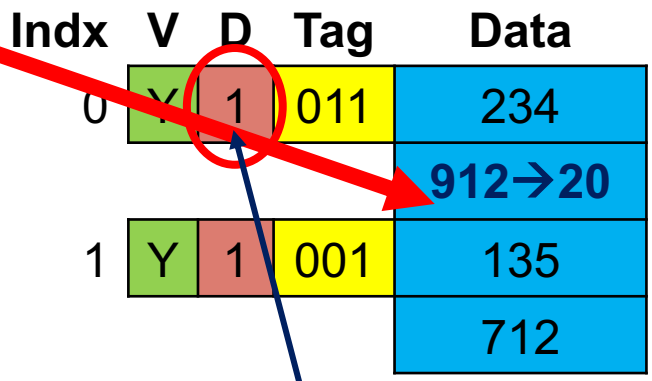
Word Addr	Data
0	110
1	120
2	133
3	233
4	36
5	23
6	615
7	712
8	3
9	300
10	62
11	99
12	234
13	912
14	0
15	10

Write Back Example

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



CPU



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

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

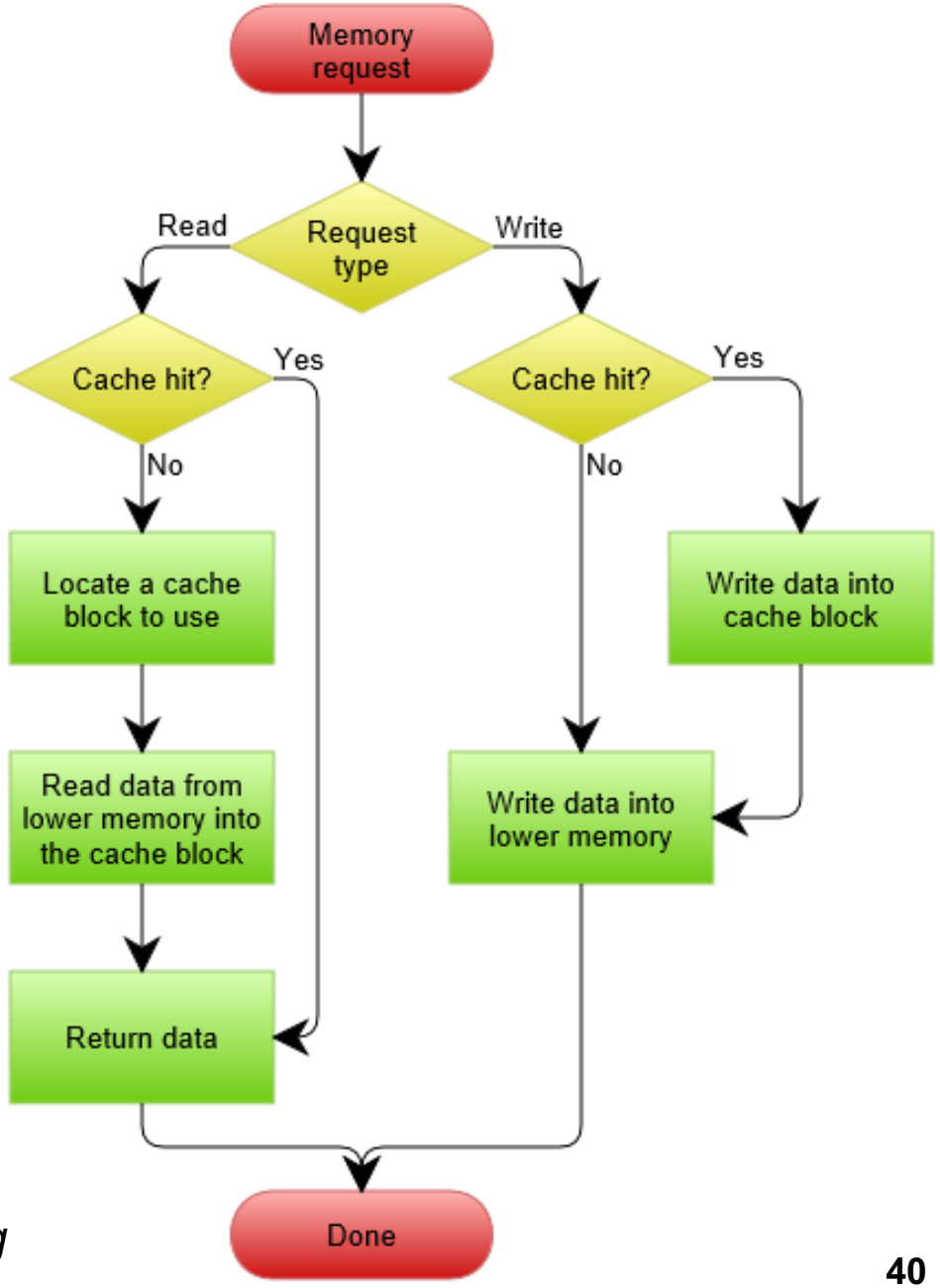
  } ***Simultaneously in one step***

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

Write Allocation on Miss

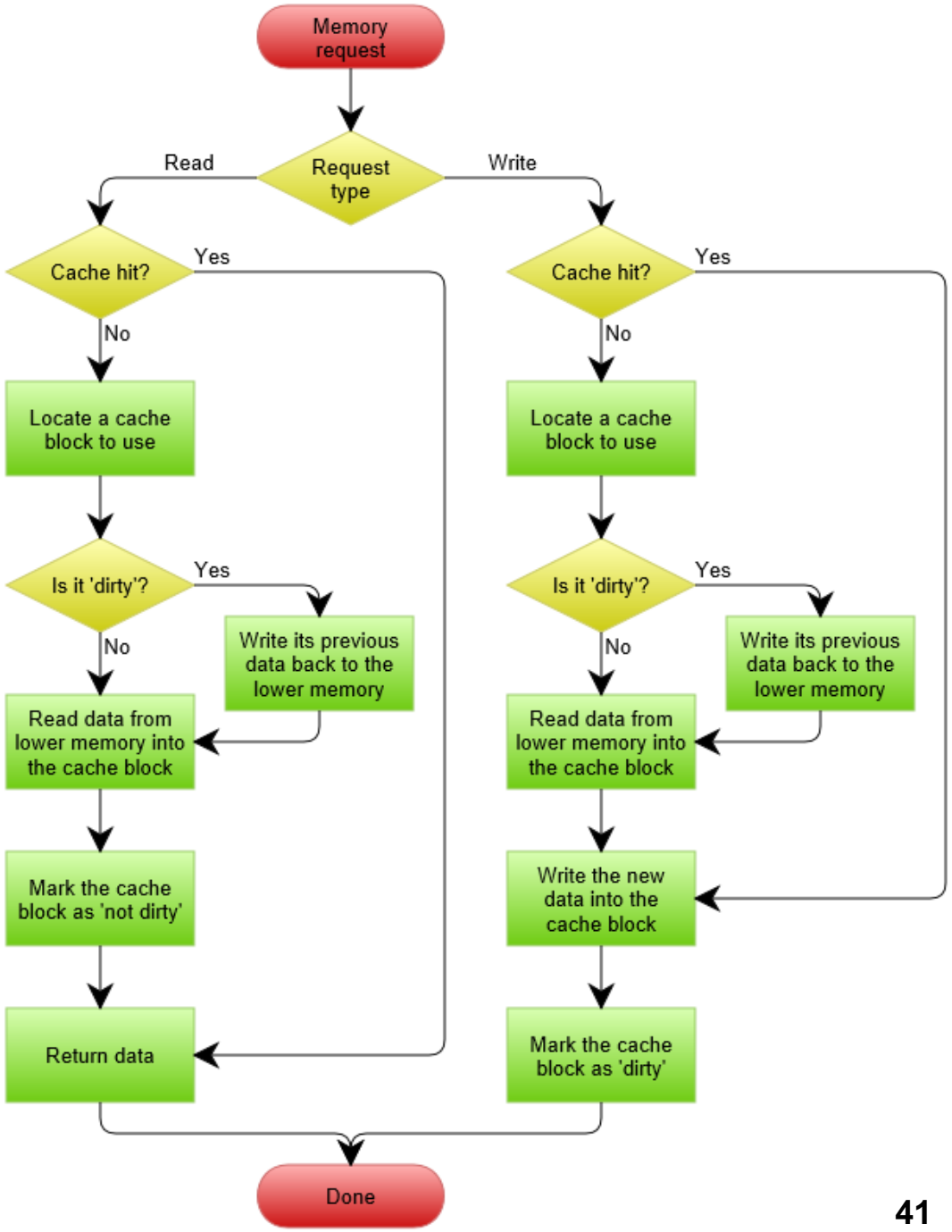
- Ways of cache handlings for write-through
  - Write Allocate
    - Allocate cache block on miss by fetching corresponding memory block
    - Update cache block
    - Update memory block
  - No Write Allocate
    - Write around: write directly to memory
    - Then fetch from memory to cache
- For write-back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

Write Through with no Write Allocation



Source: Wikipedia.org

Write Back with Write Allocation



Source: Wikipedia.org

Measuring Cache Performance

- CPU time consists of
  - Program execution cycles without cache miss (including cache hit time), plus
  - memory stall (miss) cycles, caused by cache misses
- With simplified assumptions:

$$Memory\ Stall\ Cycles$$

$$= \frac{Memory\ Accesses}{Program} \times Miss\ Rate \times Miss\ Penalty$$

$$= \frac{Instructions}{Program} \times \frac{Memory\ Access}{Instructions} \times \frac{Misses}{Memory\ Accesses} \times Miss\ Penalty$$

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 (without cache miss) = 2
  - Load & stores are 36% of instructions
- What is the overall execution time?
- What is the total CPI?

Cache Performance Example

- Total execution time (assume 100 instructions)
  - Ideal execution time
    = 100 (instructions) × 2 (base CPI) = 200 cycles
  - I-cache stall time
    = 100 (instructions) × 100% (memory access rate)
    × 2% (miss rate) × 100 (cycles, miss penalty)
    = 200 cycles
  - D-cache stall time
    = 100 (instructions) × 36% (memory access rate)
    × 4% (miss rate) × 100 (cycles, miss penalty)
    = 144 cycles
  - Total memory stall cycles = I-cache stall time + D-cache stall time
    = 200 + 144 = 344 cycles
  - Total execution time = ideal execution time + memory stall cycles
    = 544 cycles

Cache Performance Example

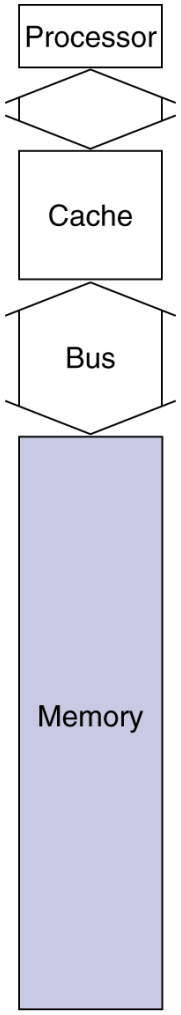
- Total CPI (actual CPI)
- Method 1:
  - Actual CPI
    = Total execution time (cycles) / instructions
    = 544 / 100 = 5.44
- Method 2:
  - Stalled cycles per instruction (CPI caused by stalls)
    I-cache stalled CPI = 100% × 2% × 100 = 2
    D-cache stalled CPI = 36% × 4% × 100 = 1.44
  - Actual CPI
    = base CPI + Miss (stalled) CPI
    = 2 + 2 + 1.44 = 5.44
- Ideal CPU is 5.44/2 =2.72 times faster

Reducing Miss Penalty by Main Memory organization

- For less memory stall cycles
- 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

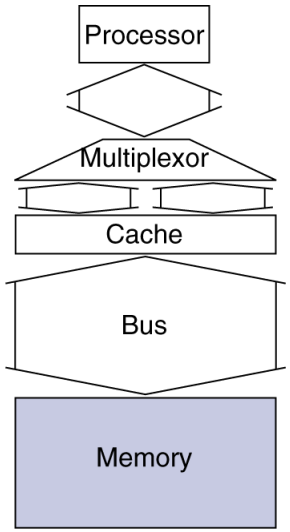


a. One-word-wide memory organization

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

Reducing Miss Penalty

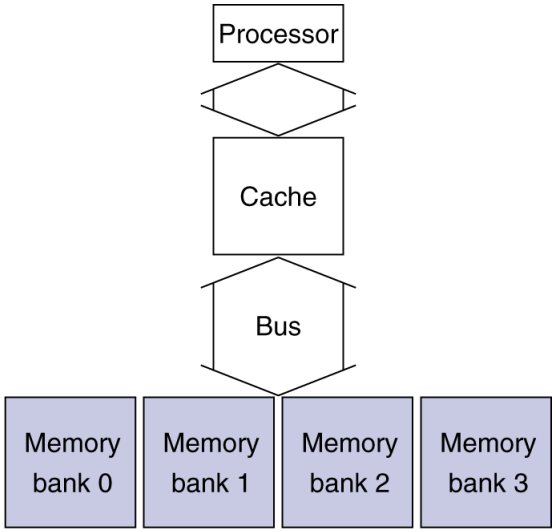
by Increasing Memory Bandwidth



b. Wider memory organization

- 2-word wide memory
  - Miss penalty = 1 + 2×15 + 2×1 = 33 bus cycles
  - Bandwidth = 16 bytes / 33 cycles = 0.48 B/cycle

- 4-bank ***interleaved memory***
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle



c. Interleaved memory organization