Memory

Part 2

Final Exam Review Session

- Final Exam Review session
- A) August 4th
- B) August 5th
- C) August 6th
- D) August 7th
- E) August 10th

Direct mapped cache

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100

Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	11	Mem[28]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

Main Memory

Direct mapped cache

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100
20	**10100**	**Miss**	**100**

Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	11	Mem[28]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

Both Addresses map
To the same cache location

something needs
To be kicked out of cache

Main Memory

Direct mapped cache

Replace

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100
20	10100	Miss	100

Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	10	Mem[20]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

Main Memory

Direct mapped cache

Replace

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100
20	**10100**	**Miss**	**100**
28	**11100**	**Miss**	**100**

Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	10	Mem[20]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

Main Memory

Direct mapped cache

Replace Again….

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100
20	10100	Miss	100
28	11100	Miss	100

Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	11	Mem[28]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

Main Memory

Direct mapped cache

Replace Again….

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100
20	10100	Miss	100
28	11100	Miss	100

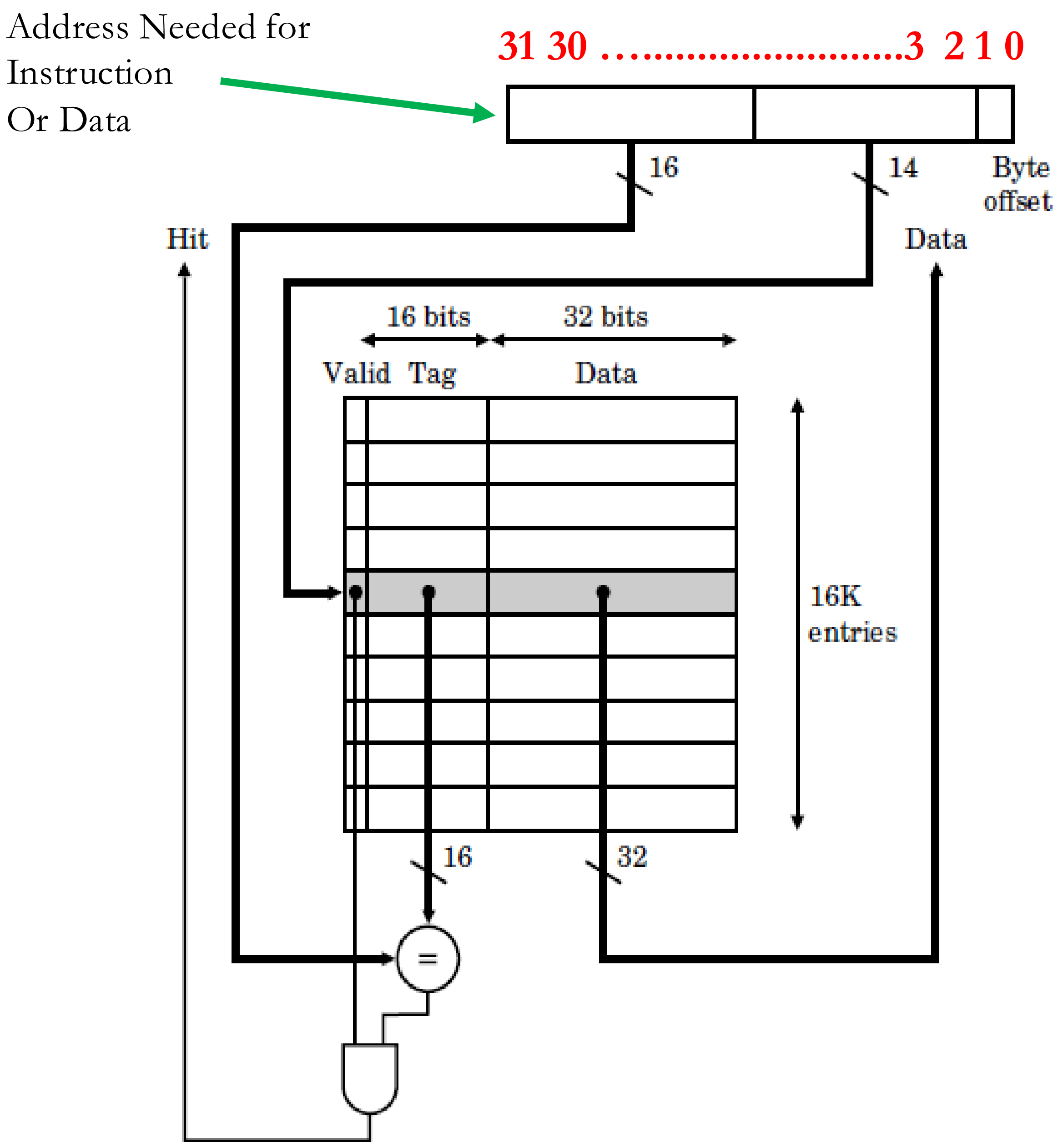
Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	11	Mem[28]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

NOTE: Block Size –One Word.
So whenever we go to RAM, we bring One Word Only

Main Memory

Testing For Cache Hit/Miss



Byte offset bits of the Address
are always 00 because words are addressable
As Multiples of 4

Use 14 Index bits to directly know where
to look in Cache (check Valid Bit first)

Then check if 16 bit Tag Field is a Match.
If **Hit** : Get the 32 bits of Data Needed

If **Miss**: Go to RAM (CPU waits)

Direct mapped cache

Memory Access
As needed by CPU

Dec	Binary	Hit/miss	Cache block
20	10100	MISS	100
18	10010	MISS	010
20	10100	Hit	100
18	10010	Hit	010
22	10110	MISS	110
7	00111	MISS	111
22	10110	Hit	110
28	11100	MISS	100
20	**10100**	**Miss**	**100**
28	**11100**	**Miss**	**100**

Cache

Index	V	Tag	Data
000	0		
001	0		
010	1	10	Mem[18]
011	0		
100	1	11	Mem[28]
101	0		
110	1	10	Mem[22]
111	1	00	Mem[7]

This Problem
Of Multiple address
Mapping to the same
Cache location :
~~leads to Many Misses.~~

Instead of One Location
Where the Entry can map
Into the cache: Allow
It to Map to a Set of locations
In Cache.

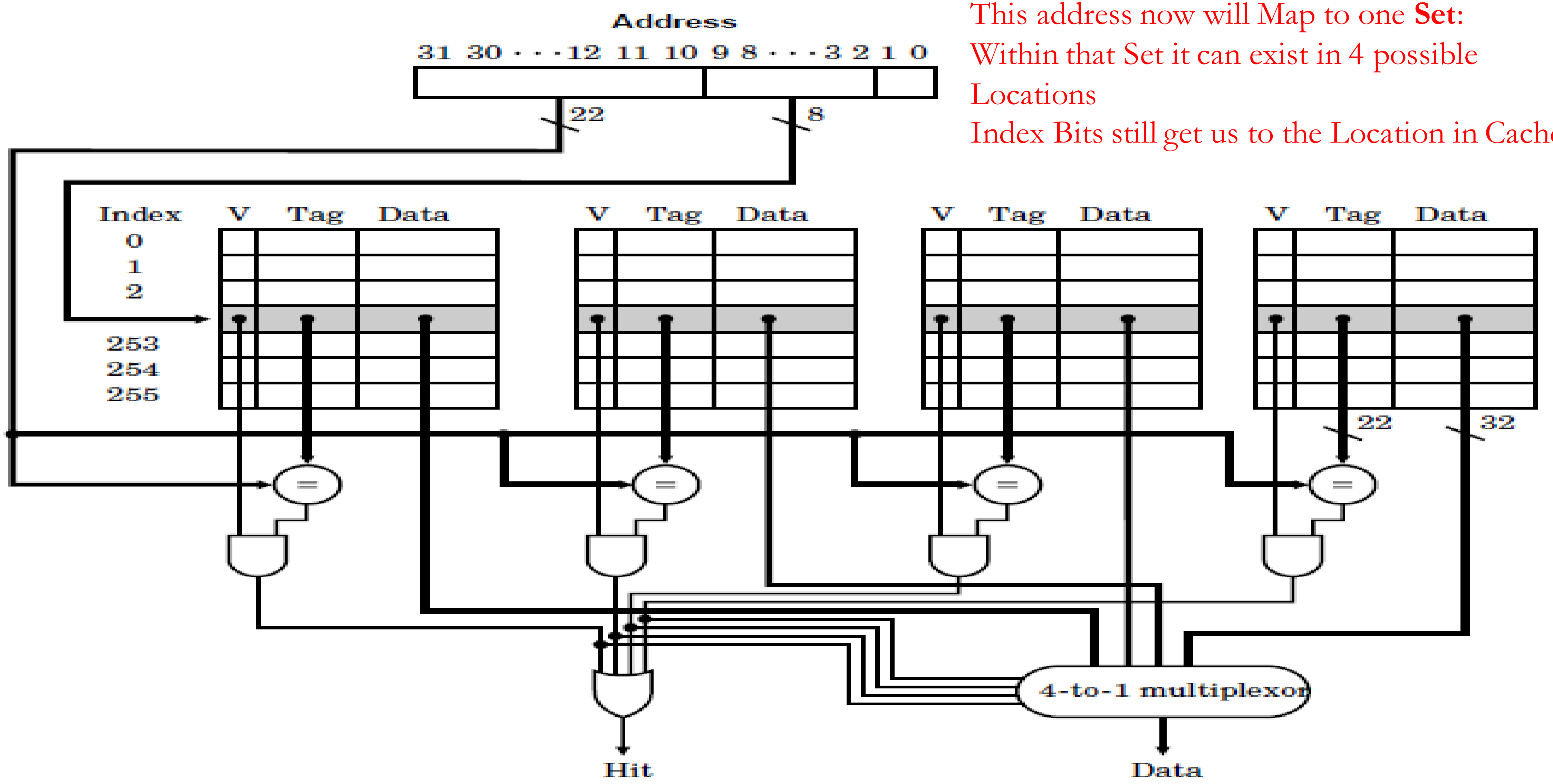
This will increase the Hit Rate

Main Memory

Alternative Placement Schemes

A 4-way Set-Associative Cache

This address now will Map to one **Set:**
Within that Set it can exist in 4 possible Locations
Index Bits still get us to the Location in Cache



2-way set associative cache

Still 8 Addressable Locations

Memory Access

Dec	Binary	Hit/miss
20 18	10100 10010	
20 18	10100 10010	
22 7	10110 00111	
22 28	10110 11100	

Cache

Index	Tag0	Tag1
00 01		
10 11		

4-way set associative cache

Memory Access

Dec	Binary	Hit/miss
20	10100	
18	10010	
20	10100	
18	10010	
22	10110	
7	00111	
22	10110	
28	11100	

Still 8 Addressable Locations

Cache

Index	Tag0	Tag1	Tag2	Tag3
0				
1				

Fully associative cache

Memory Access

Dec	Binary	Hit/miss
20	10100	
18	10010	
20	10100	
18	10010	
22	10110	
7	00111	
22	10110	
28	11100	

Still 8 Addressable Locations

Cache

Tag0	Tag1	Tag2	Tag3	Tag4	Tag5	Tag6	Tag7

Replacement Scheme

- What to replace in cache when there is no more room.
- One location needs to be overwritten
- Replace Least Recently Used: Or the Location that

was used least recently : The oldest Reference to it

Addresses :   **ONLY 4 Locations in Cache**

8

12

16

20

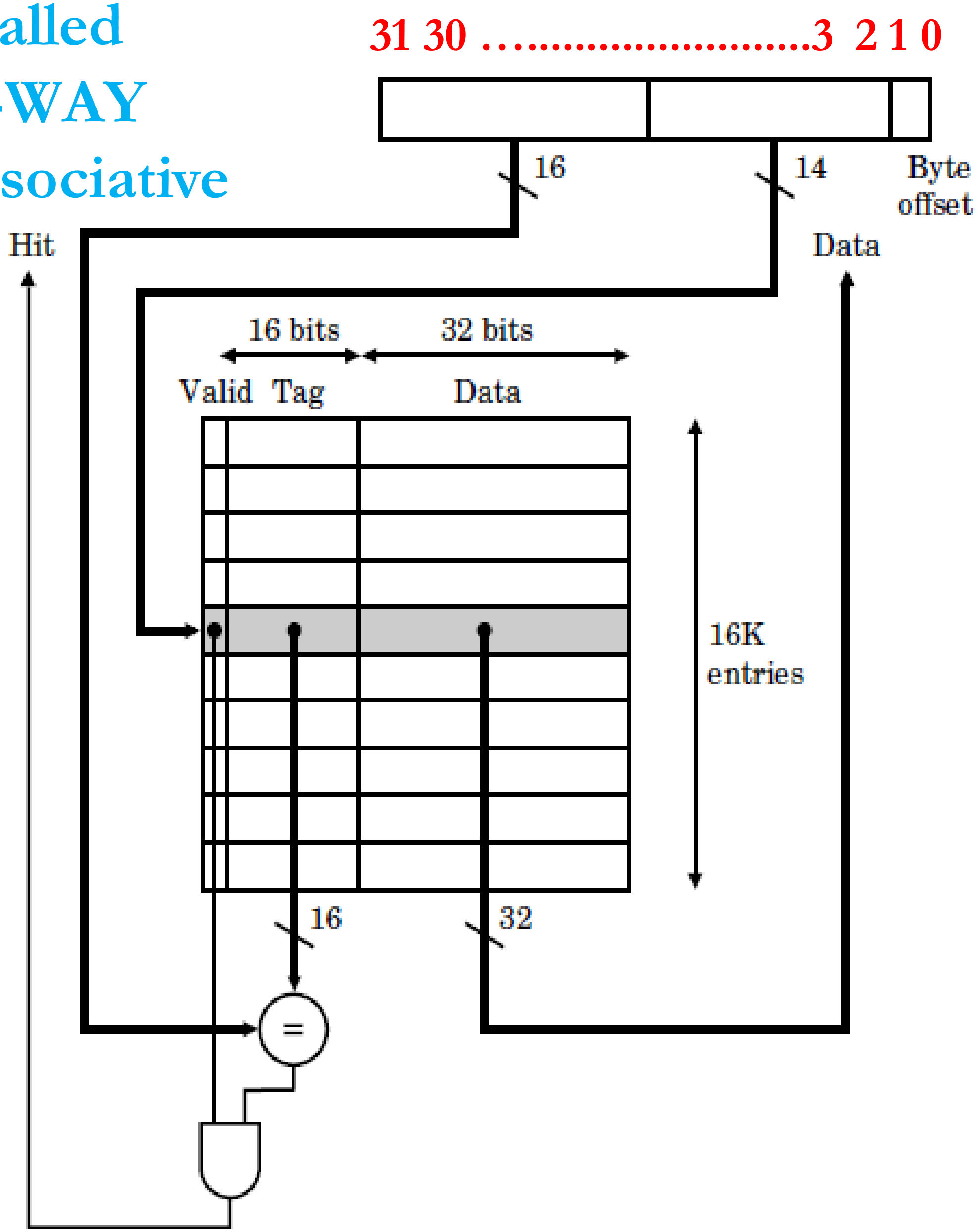
8

16

24 * Something in cache will need to be replaced : Replace Location 12: LRU

Testing For Cache Hit/Miss

Also called
ONE-WAY
Set Associative

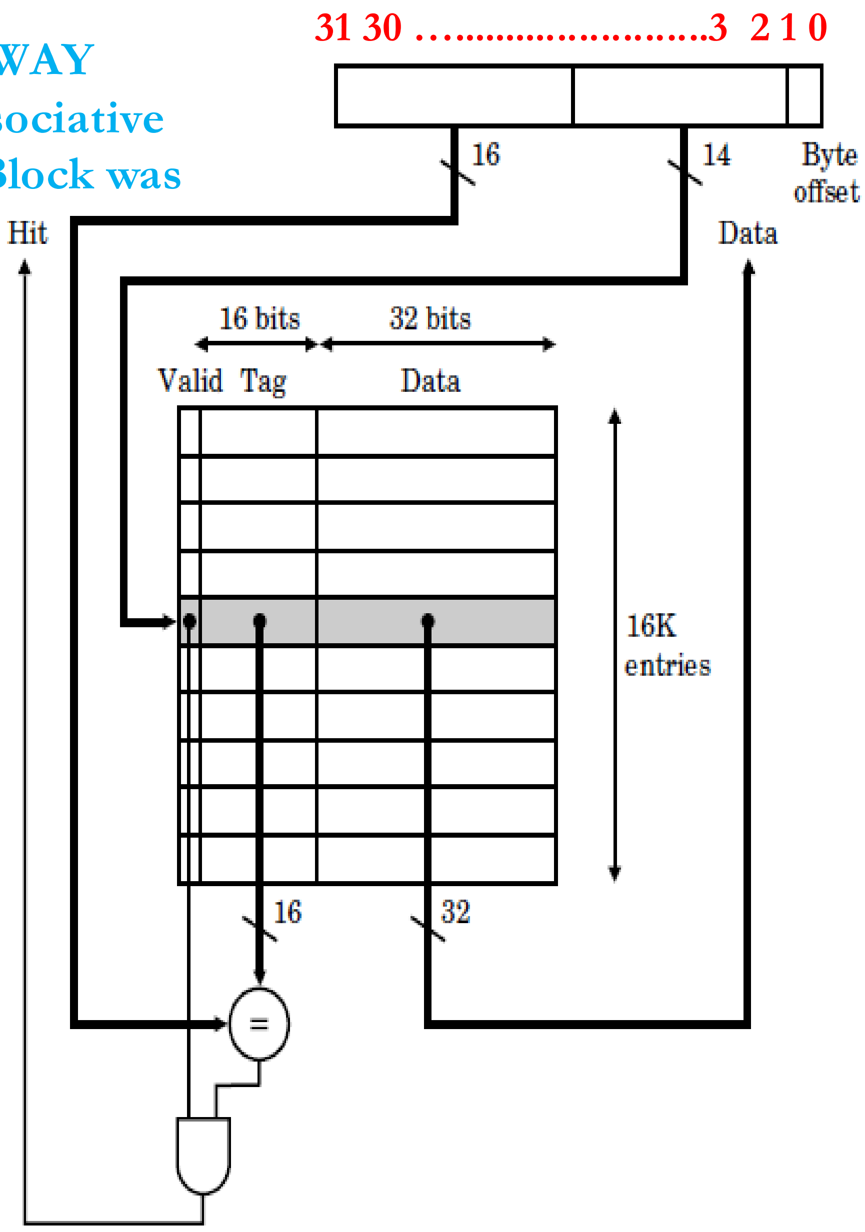


- **Out of all the Associative Caches.**
- **Which has the *longest look up time*,**
- **To check if the Address exists in Cache**
- **Or Not?**
- **(assume cache size of >= 16K)**

- A) 4 way set associative
- B) All the same
- C) 2 way set associative
- D) Fully Associative
- E) One way Set Associative

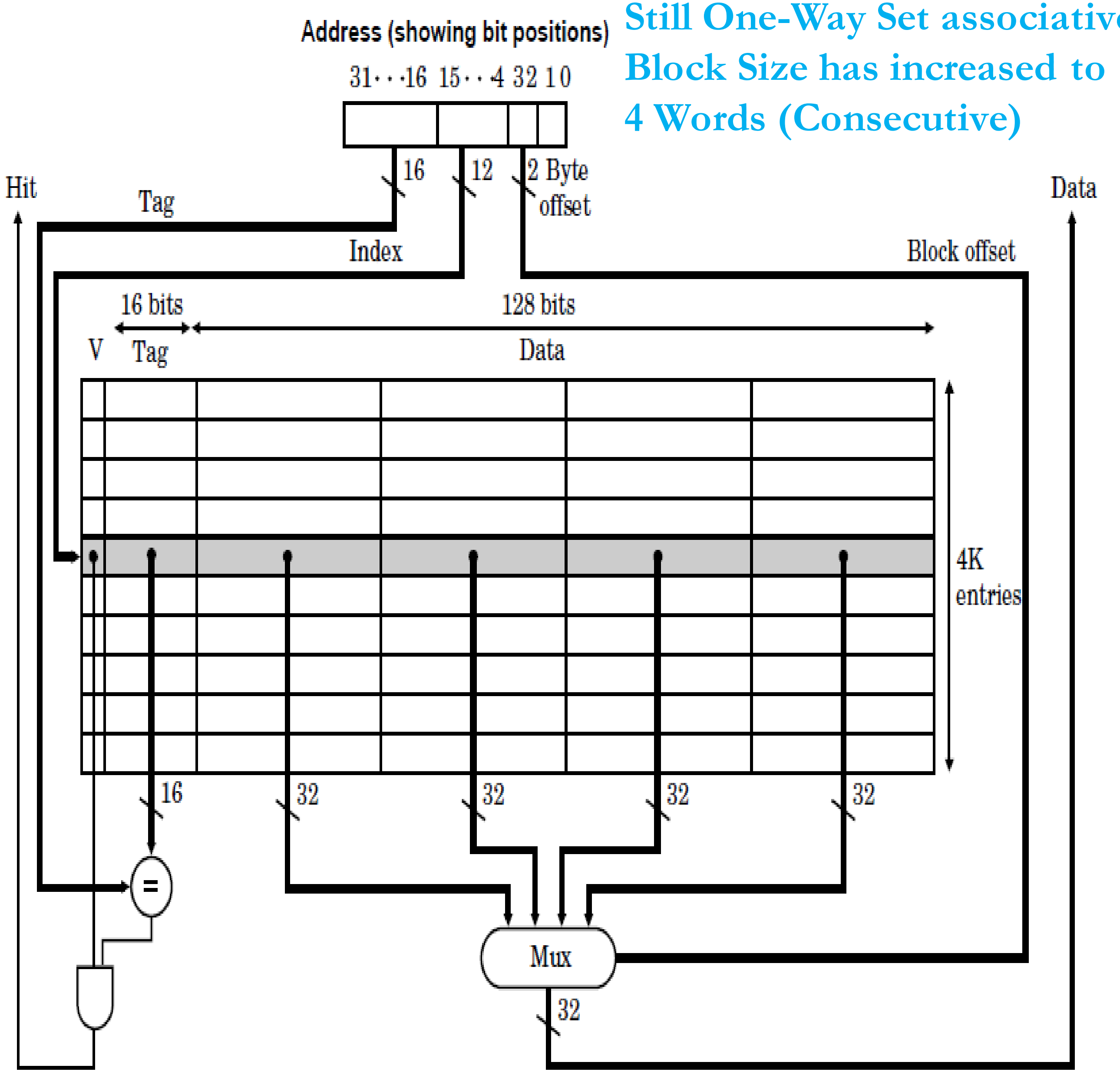
Testing For Cache Hit/Miss

ONE-WAY
Set Associative
Each Block was
1 word

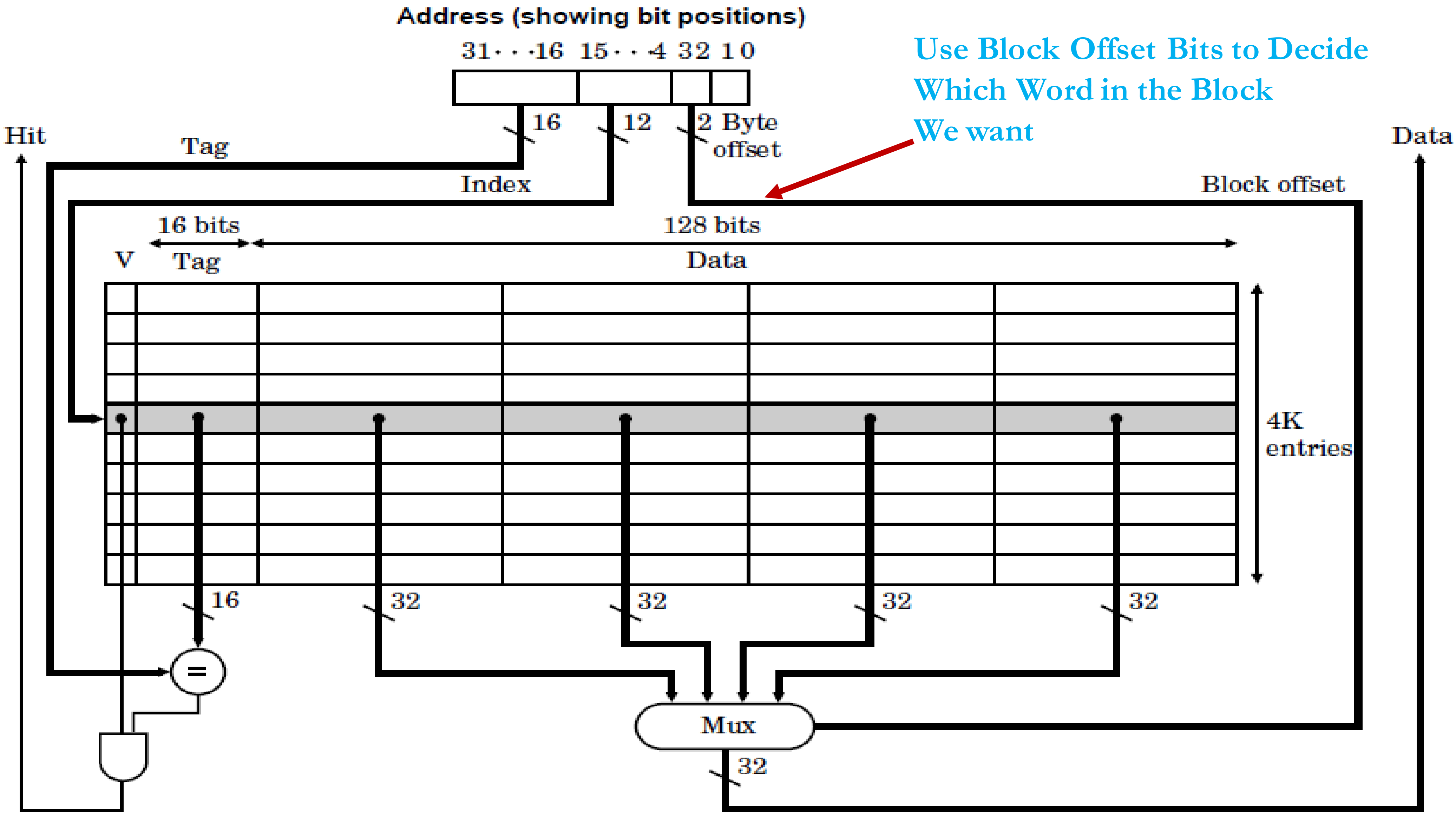


Spatial Locality: Larger Block Sizes

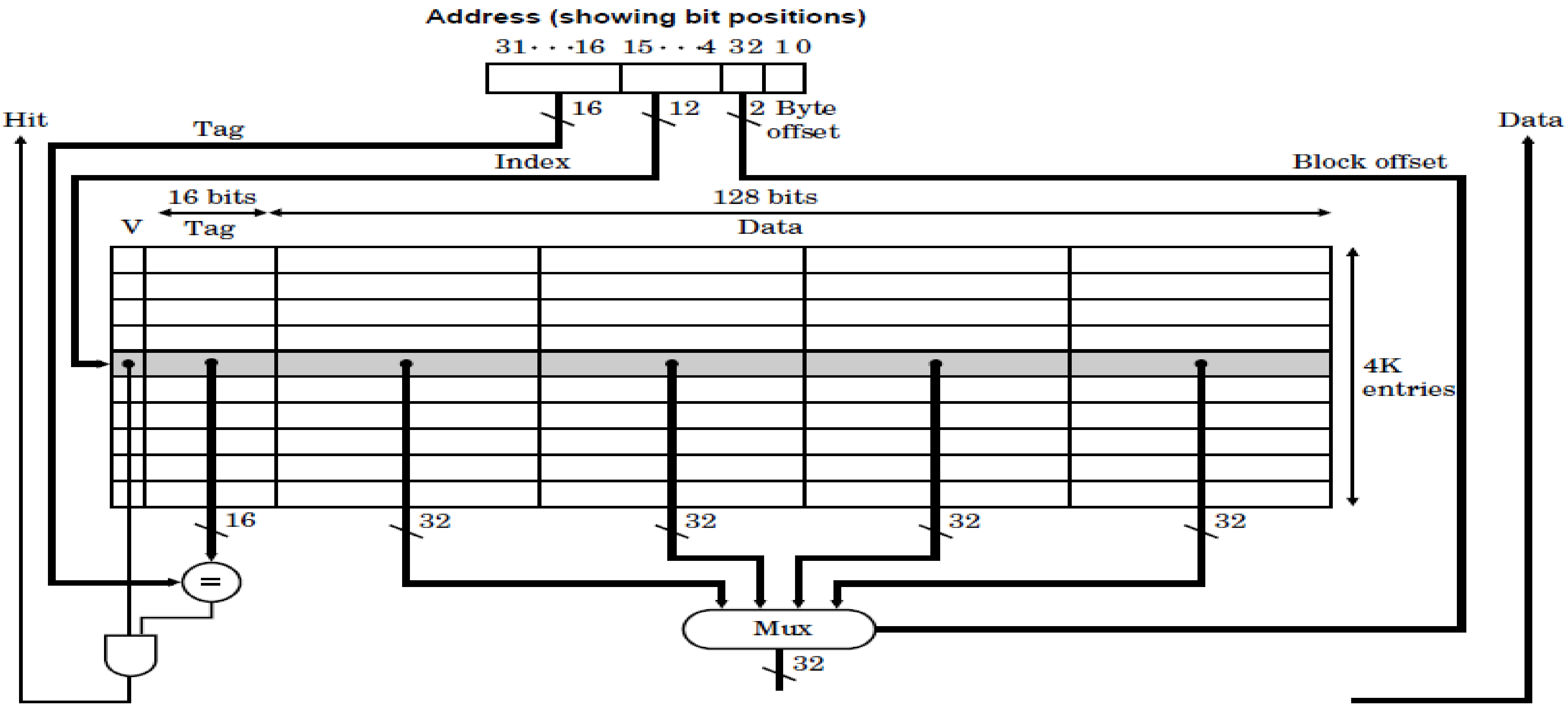
Still One-Way Set associative
Block Size has increased to
4 Words (Consecutive)



Spatial Locality: Larger Block Sizes



Use Block Offset Bits to Decide Which Word in the Block We want



Tradeoffs in Choosing Block Size

- Smaller blocks mean more misses for local references
- Larger blocks mean fewer blocks in cache, premature bumping
- Larger blocks increase miss penalty
- Memory must be read on write miss if block size > 1

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011					
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	
lw	52	000	011	01	00	
lw	56	000	011	10	00	
lw	60	000	011	11	00	

Lw instructions

Data for LW is at Given Decimal Address

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011					
100					
101					
110					
111					

4 different lw instructions
Data at different Addresses

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	
lw	52	000	011	01	00	
lw	56	000	011	10	00	
lw	60	000	011	11	00	

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011					
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	
lw	56	000	011	10	00	
lw	60	000	011	11	00	

Now
We will get this
Entire Block of
4 Words from
RAM: 20cc extra

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	000	Mem[48]	Mem[52]	Mem[56]	Mem[60]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	56	000	011	10	00	Hit
lw	60	000	011	11	00	Hit

Accessing these Consecutive locations Now leads to HITS

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	000	Mem[48]	Mem[52]	Mem[56]	Mem[60]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	56	000	011	10	00	Hit
lw	60	000	011	11	00	Hit

Total Clock Cycles:
20cc RAM
+ 4 instructions
= 24 cc
~~Assuming Pipeline~~
Already running ☺

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	000	Mem[48]	Mem[52]	Mem[56]	Mem[60]
100					
101					
110					
111					

Sw and Lw Instructions Accessing Data in Non Consecutive locations

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	
lw	188	001	011	11	00	
lw	48	000	011	00	00	

Another Example

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	000	Mem[48]	Mem[52]	Mem[56]	Mem[60]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	
lw	188	001	011	11	00	
lw	48	000	011	00	00	

All the instructions
Have the same
Index field

SW:
Another Cache Miss
Bring in Entire Block
20cc

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	001	Mem[176]	Mem[180]	Mem[184]	Mem[188]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	MISS
lw	188	001	011	11	00	
lw	48	000	011	00	00	

Brought in
Entire 4 word BLOCK
That contained
Address 184

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	001	Mem[176]	Mem[180]	Mem[184]	Mem[188]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	MISS
lw	188	001	011	11	00	Hit
lw	48	000	011	00	00	

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	001	Mem[176]	Mem[180]	Mem[184]	Mem[188]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	MISS
lw	188	001	011	11	00	Hit
lw	48	000	011	00	00	

NOTE:
SW Modifies
Memory Location

Write Back:
Only cache is updated
~~RAM and Cache~~
Temporarily
Inconsistent data

Write Through:
This Write to Cache
Also requires
A write to RAM
Takes additional
Time:
Using Write Buffer will
Decrease delay to RAM

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	001	Mem[176]	Mem[180]	Mem[184]	Mem[188]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	MISS
lw	188	001	011	11	00	Hit
lw	48	000	011	00	00	MISS

NOW:
LW 48 belongs
To same index in Cache

Need to Swap Out

SW instruction

Block Size Read Example

	Tag	00	01	10	11
000					
001					
010					
011	000	Mem[48]	Mem[52]	Mem[56]	Mem[60]
100					
101					
110					
111					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	MISS
lw	52	000	011	01	00	Hit
lw	48	000	011	00	00	Hit
sw	184	001	011	10	00	MISS
lw	188	001	011	11	00	Hit
lw	48	000	011	00	00	MISS

NOW:
LW 48 belongs
To same index in Cache

Need to Swap Out

SW instruction:

If Write Through
Was used: We are okay
Do not need to updated
RAM

Write Back Scheme
Will require writing data
To RAM NOW
Adding delay

- Write Back:
- Do not update RAM right away: Wait until this word is kicked out of cache
- At this point →update RAM with the new value of data
- ~~Entire Block gets written to RAM (good if many sw instructions)~~
- Slower on Cache Misses: Increases miss penalty

  Usually Modified Data Block will have a Dirty Bit indicating if this data was updated
  - Not only does data need to be loaded from Ram into Cache
  - But what is being replaced in Cache must be updated to RAM (entire Block)
- Write Through : Update RAM right away
  - Every Time there is a Sw instruction: Write not only to Cache but also to RAM
  - Keep data consistent
  - Having a write buffer will improve this, writing only the new WORD not the entire Block

Handling Cache Misses

- On miss: stall entire processor until item fetched
- On write: usually written item goes into cache
- Write-through: immediately write item back into memory
- Write-back: write item only into cache

  Cache and memory temporarily inconsistent

  Write back cache block only when it must be replaced
- Could have separate instruction and data caches (increases bandwidth)

- If Write Through takes 10cc, and Write Back takes 20cc
- Which method would be more efficient for the following code fragment

Block Size Read/Write Example

	Tag	00	01	10	11
000					
001					
010					
011					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	
sw	52	000	011	01	00	
lw	48	000	011	00	00	
sw	184	001	011	10	00	
lw	188	001	011	11	00	
lw	48	000	011	00	00	

- If Write Through takes 10cc, and Write Back takes 20cc
- Loading from RAM to Cache on Miss: 20cc for 4 word blocks
- Assume 1cc per instruction : FOR ALL INSTRUCTIONS (We do not know if LW followed by use)

Memory Hierarchies

Block Size Read/Write Example

	Tag	00	01	10	11
000					
001					
010					
011					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	
sw	52	000	011	01	00	
lw	48	000	011	00	00	
sw	184	001	011	10	00	
lw	188	001	011	11	00	
lw	48	000	011	00	00	

- If Write Through takes 10cc, and Write Back takes 20cc
- Loading from RAM to Cache on Miss: 20cc for 4 word blocks
- Which is correct execution time ? (Based on either method)
- Assume 1cc per instruction : FOR ALL INSTRUCTIONS (We do not know if LW followed by use)

Block Size Read/Write Example

	Tag	00	01	10	11
000					
001					
010					
011					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	
sw	52	000	011	01	00	
lw	48	000	011	00	00	
sw	184	001	011	10	00	
lw	188	001	011	11	00	
lw	48	000	011	00	00	

A) Write Back: First Miss 20cc , plus 3 Miss: 20cc plus 20 writing back, +2 Miss 20cc plus 20cc writing back +1: Total: 106 cc

B) Write Through : First Miss 20cc +3 +10 Miss: 20cc + 10cc + 2 Miss: 20cc + 1 : Total 86 cc

C) Both A and B

D) Write Back: 103 : Write Through 83

E) All of the above

- Write Back may seem much worse :However if there are many sw instructions in one block (ie
- Many sw instructions working on every word in an array ➔ Write Back would be much more useful
- Think of an instruction mix where Write Back would be the best suited policy

Block Size Read/Write Example

	Tag	00	01	10	11
000					
001					
010					
011					

	Dec	Tag	Indx	Block	Byte	Hit/Miss
lw	48	000	011	00	00	
sw	52	000	011	01	00	
lw	48	000	011	00	00	
sw	184	001	011	10	00	
lw	188	001	011	11	00	
lw	48	000	011	00	00	

A) Write Back: First Miss 20cc , plus 3 Miss: 20cc plus 20 writing back, +2 Miss 20cc plus 20cc writing back +1: Total: 106 cc

B) Write Through : First Miss 20cc +3 +10 Miss: 20cc + 10cc + 2 Miss: 20cc + 1 : Total 86 cc

C) Both A and B

D) Write Back: 103 : Write Through 83

E) All of the above

Radix Sort:

- https://www.youtube.com/watch?v=YXFI4osELGU
- Sorting A list of Numbers:
  - based on successively examining each digit.

Why does Quicksort have less cache misses than Radix sort

- A)Radix sort needs to pass the entire list repeatedly therefore a big list cannot be stored all in cache
- B) Quicksort needs only to work on smaller portions of the original list,
- Successively working on subsets of the original list.
- C) Quicksort we need to pass again and again through the entire list , similar to radix sort, however this is done recursively
- D) Radix sort only examines n times the number of digits therefore this makes cache misses rise
- E) both A and B