EE116C/CS151B
Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang

Homework Discussion

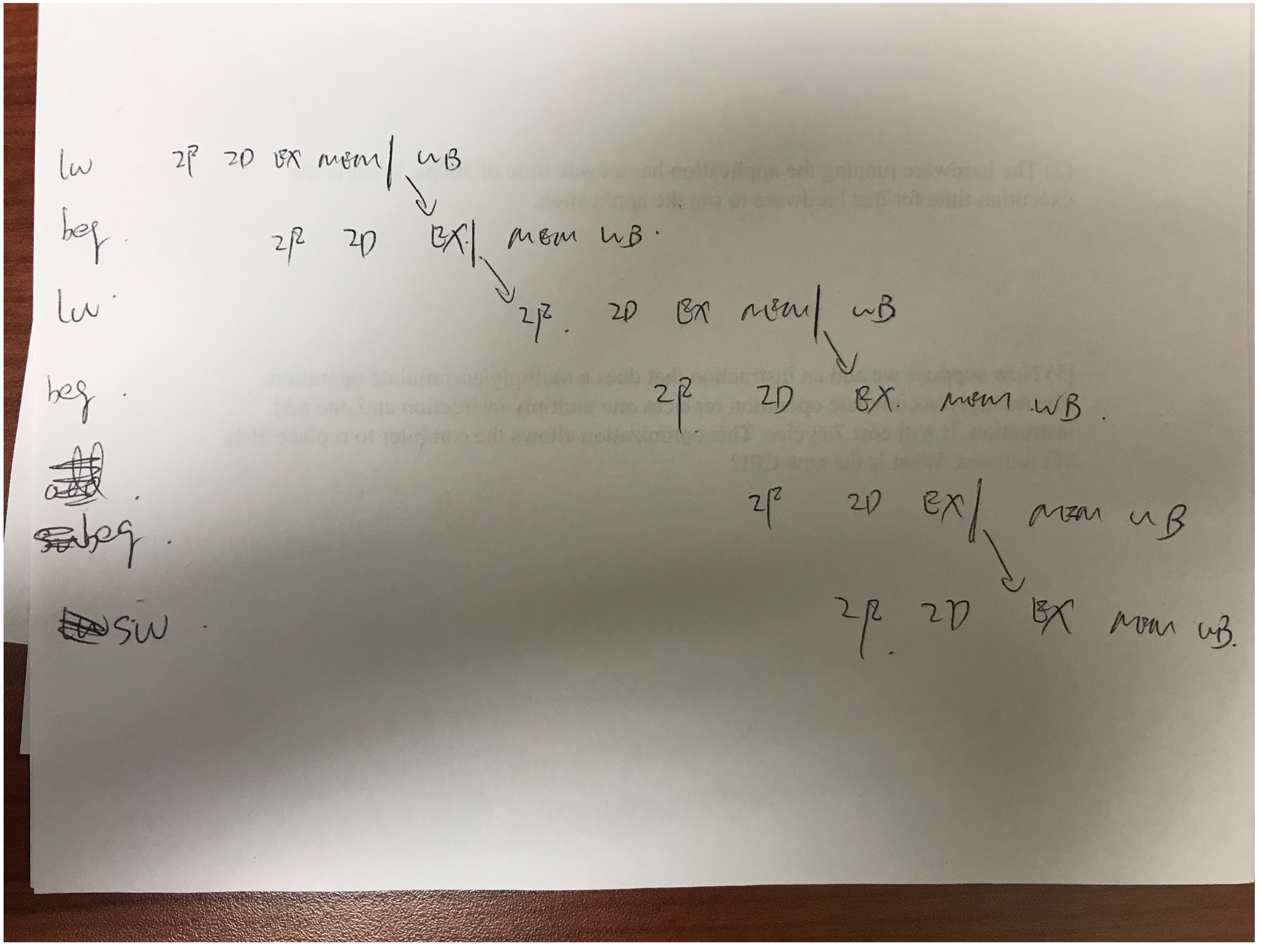
Problem 1

This is exercise is intended to help you understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
lw r2,0(r1)
label1: beq r2,r0,label2 # not taken once, then taken
lw r3,0(r2)
beq r3,r0,label1 # taken
add r1,r3,r1
label2: sw r1,0(r2)
```

Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

Homework Discussion



Homework Discussion

Problem 2

This is exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT

1. What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

2. What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.63 (predict not taken)?

3. What is the accuracy of the two-bit predictor if this pattern is repeated forever?

Homework Discussion

1. What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

Always-taken predictor:
Actual: T, NT, T, T, NT
Predicted: T, T, T, T, T
Accuracy = 3/5 = 60%

Always-not-taken predictor:
Actual: T, NT, T, T, NT
Predicted: NT, NT, NT, NT, NT
Accuracy = 2/5 = 40%

2. What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.63 (predict not taken)?

Actual: T, NT. T. T
Predicted: **NT, NT, NT, NT**
Accuracy = number of hits / total braches = 1/4 = 25%

3. What is the accuracy of the two-bit predictor if this pattern is repeated forever?

1st round: 0, 1, 0, 1, 2

2nd round: 1, 2, 1, 2, 3

3rd round: 2, 3, 2, 3, 3

4th round: 2, 3, 2, 3, 3

We can see that the accuracy = 3/5 = 60%.

here the numbers are states!

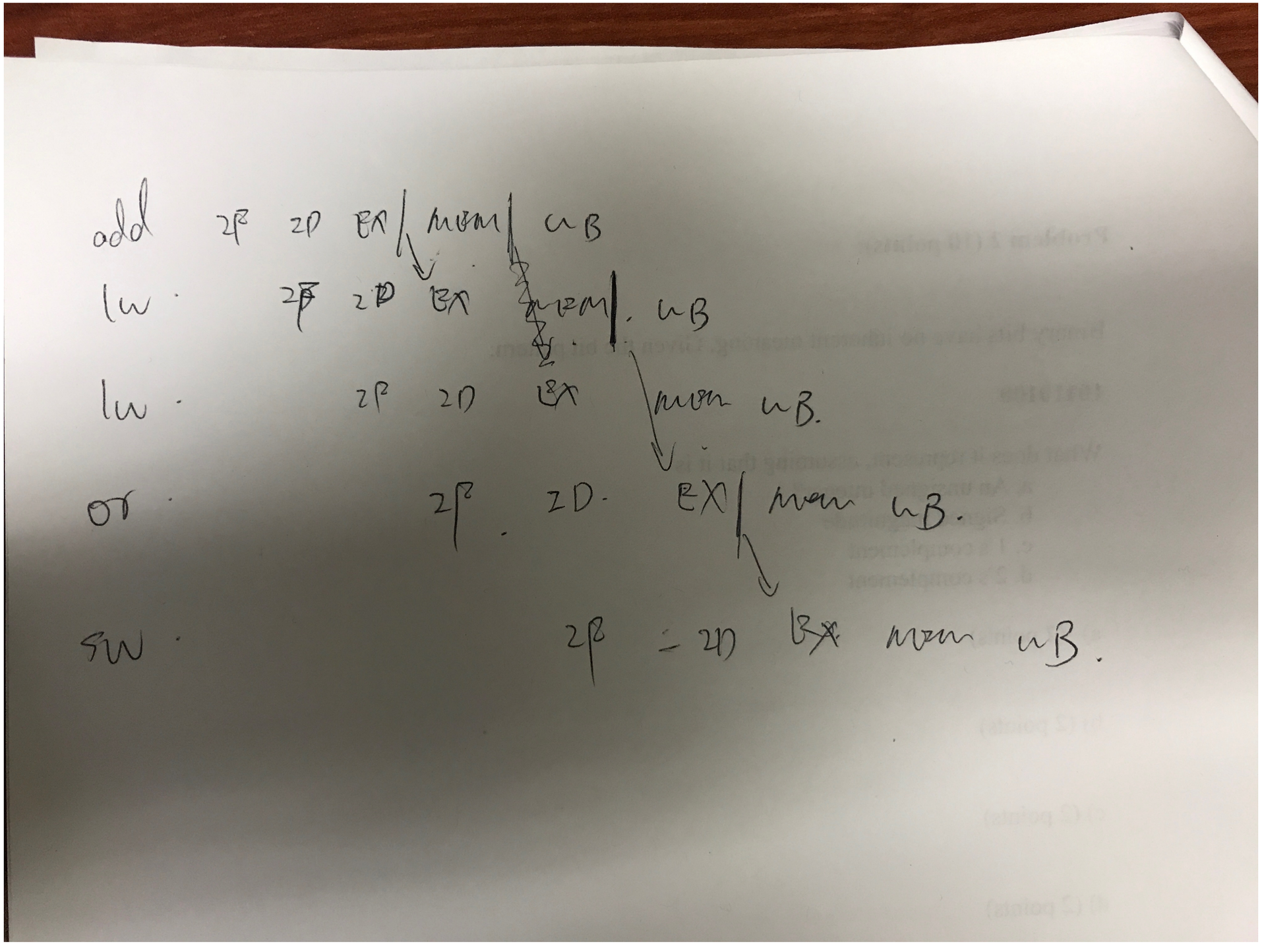
Homework Discussion

For the code sequence below, how many cycles would it take to execute this code on the 5-stage pipelined data path with forwarding logic?

```
add r5,r2,r1
lw r3,4(r5)
lw r2,0(r2)
or r3,r5,r3
sw r3,0(r5)
```

Draw the pipeline diagram to answer.

Homework Discussion



Homework Discussion

Problem 4

For the code sequence below, show the pipeline diagram for the following cases:

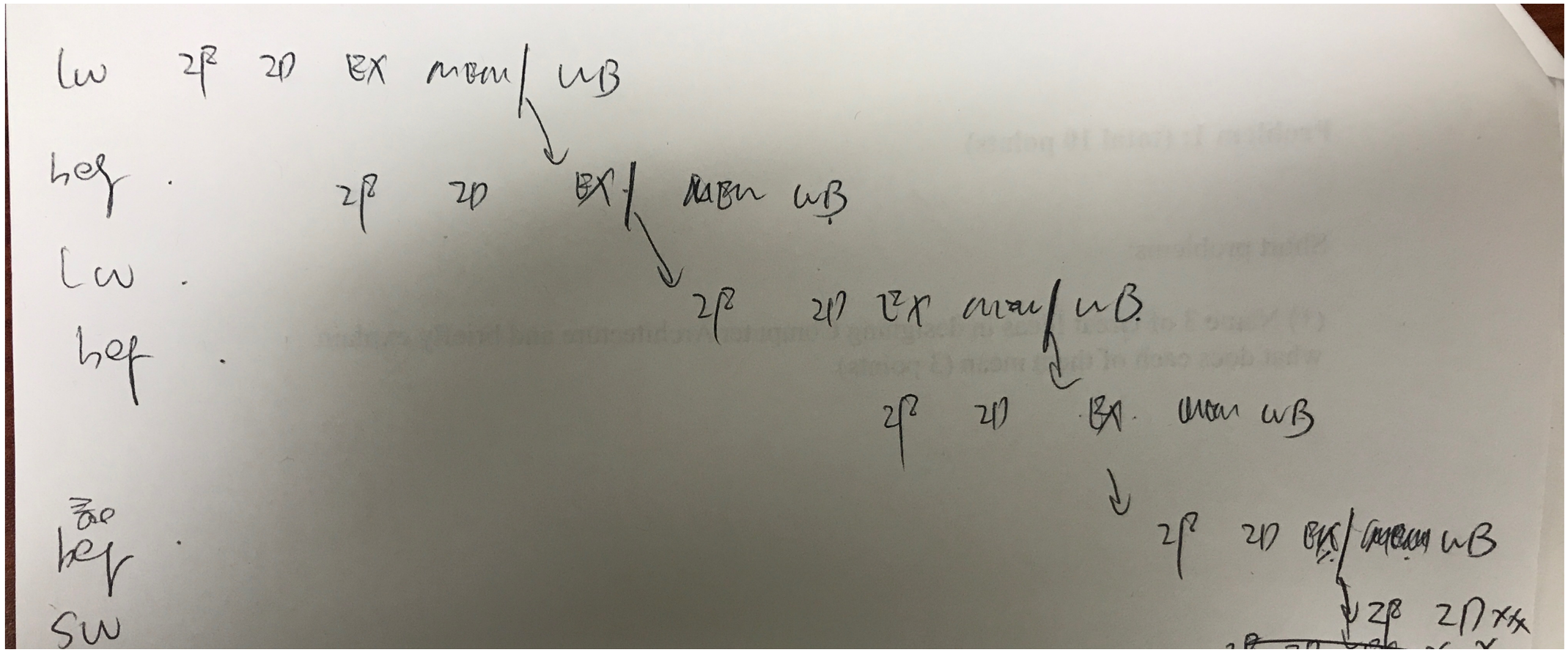
```
        lw r2,0(r1)
label1: beq r2,r0,label2 # not taken once, then taken
        lw r3,0(r2)
        beq r3,r0,label1 # taken
        add r1,r3,r1
label2: sw r1,0(r2)
```

a) full forwarding, branches resolved in EX

b) full forwarding, branches resolved in ID

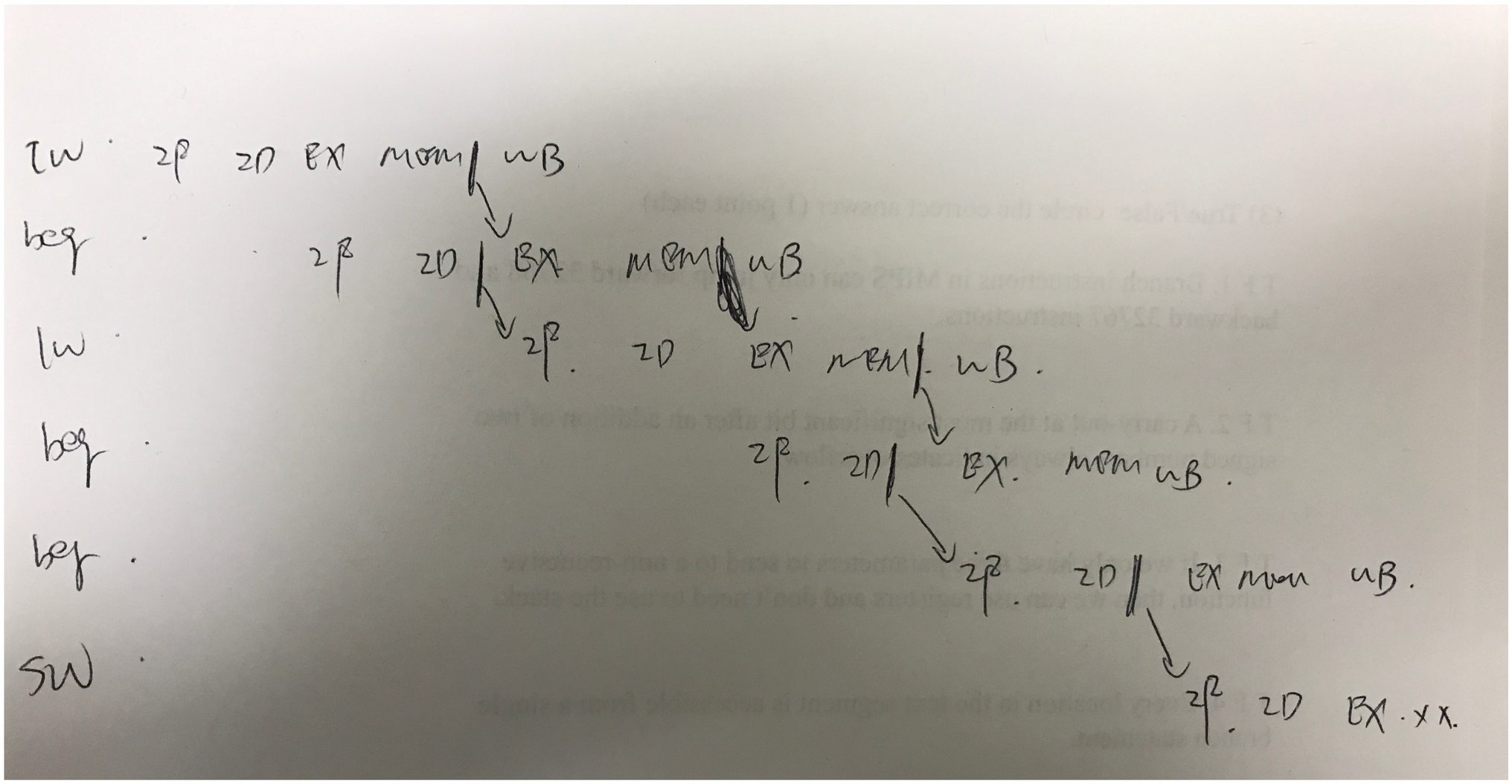
Homework Discussion

a) full forwarding, branches resolved in EX



Homework Discussion

b) full forwarding, branches resolved in ID



Review

- Types of memory
- Memory hierarchy
- Addressing (with memory hierarchy)
- Whole picture: work with memory
- Memory efficiency evaluation
- Associated caches: replacement policy, multilevel-caches

Review

- **Types of memory**
    - Register: Volatile
        - inside cpu. flip-flops.
    - Static RAM (SRAM): Nonvolatile
        - 0.5ns – 2.5ns, $2000 – $5000 per GB
        - **Cache**
    - Dynamic RAM (DRAM): Volatile
        - 50ns – 70ns, $20 – $75 per GB
        - **Main memory**
    - Flash Storage: Nonvolatile **(SSD = Flash + DRAM)**
        - 5000ns – 50000ns ms, $0.50 – $3 per GB
        - write times limit. write seed << read speed
        - **SSD**
    - Magnetic disk: Nonvolatile
        - 5ms – 20ms, $0.20 – $2 per GB
        - **Disk**

Review

- **In running program, main memory is data's "home location".**
  - **Addresses refer to location in main memory.**
  - **"Virtual memory" allows disk to extend DRAM**

When data is accessed, it is automatically moved into cache

- **Processor (or smaller cache) uses cache's copy**

Review

- **Workflow**
    - Register (inside CPU) <- SRAM (CPU cache) <- DRAM (main memory) <- magnetic disk / flash (bulky storage)
- **Memory hierarchy**
    - determine which to fetch?
        - **everything needed.**
        - **things predicted to be needed next:**
            - Temporal locality
            - Spatial locality
            - other strategies

Review

- **Strategy evaluation:**
  - **Hit: access satisfied by upper level**
    - Hit ratio: hits/accesses
- **Miss: block copied from lower level**
  - Miss ratio: misses/accesses = 1 – hit ratio
  - miss penalty: time taken (lower -> upper)

Review

- **Cache (SRAM) data access**
  - How to know if certain data is in cache
  - How to locate it in cache

Review

- **Cache (SRAM) data access**
  - How to locate it in cache -> determine where to store mem data of certain address
  - Answer:
    - direct-map (address of data in cache ~ address of data in mem)

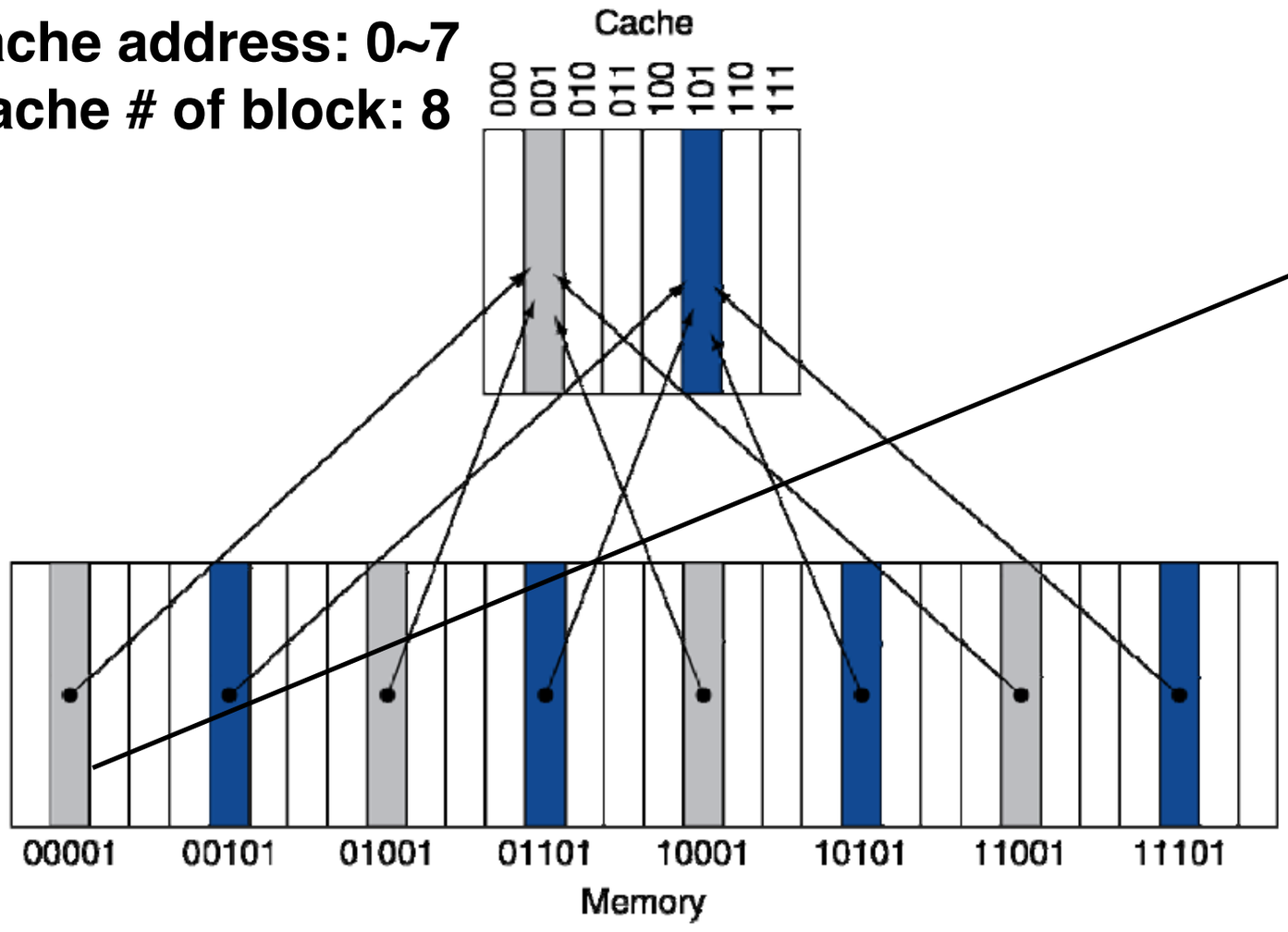
Review

- **Cache (SRAM) data access**
  - How to know if certain data is in cache
  - **Answer:**
    - direct-map (address of data in cache ~ address of data in mem)
      - if there's that data in cache, it must be in one position
      - one check for searching/locating

Review

(Block address) modulo (#Blocks in cache)

cache address: 0~7
cache # of block: 8



mem address: 1
mem add / 8 = 1
should be in 001 in cache

mem address: 00 00 ~ 11 111
last three bits determine cache add
4 men blocks are in one cache block

00 001
01 001
10 001
11 001

cache address: 000 ~ 111

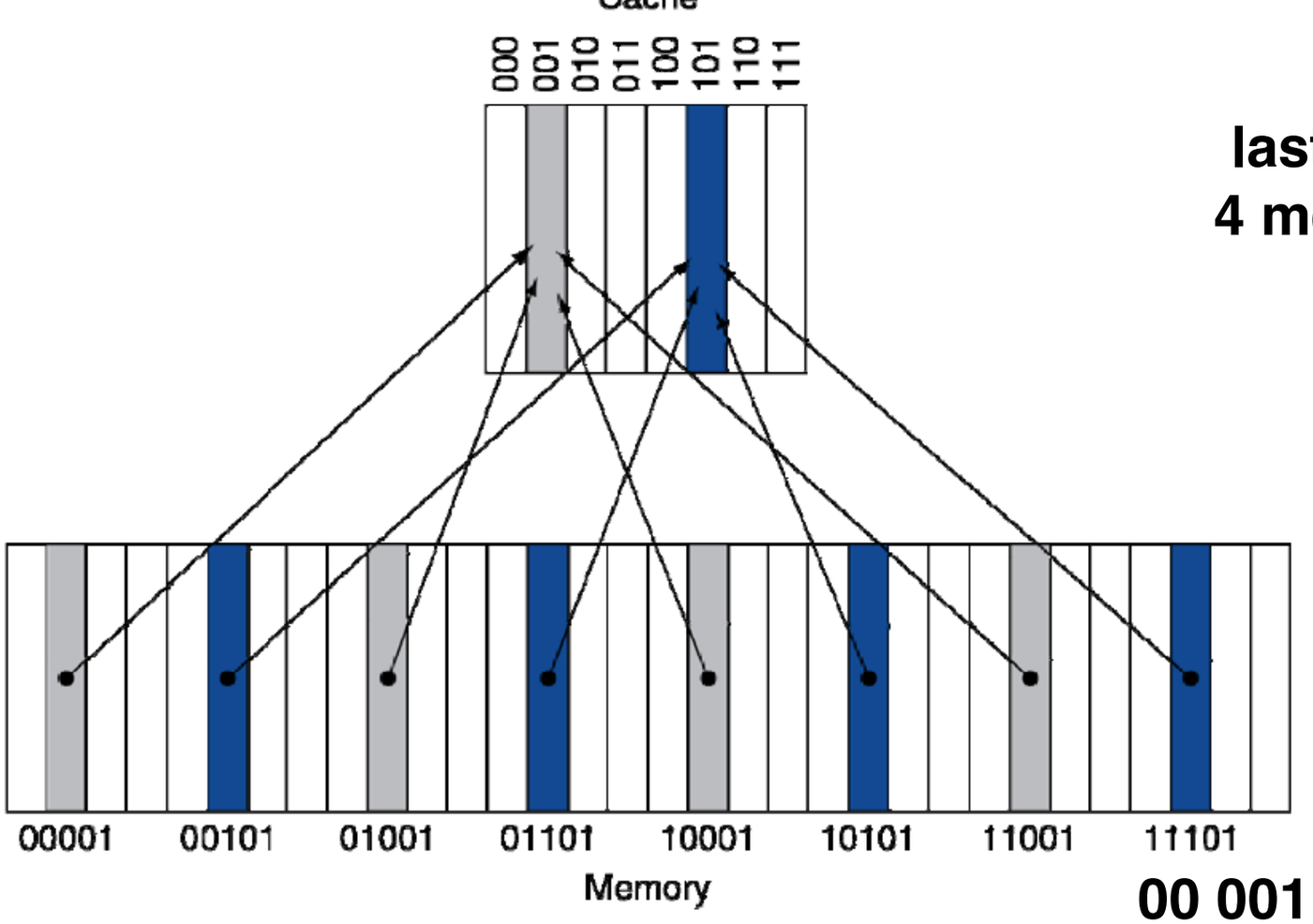
Review

- **Cache (SRAM) data access**
  - How to differentiate multiple mem blocks in one cache block
- **Answer:**
  - way 1: attach original address
  - **way 2: attach tags: reduce redundancy.**

Review

(Block address) modulo (#Blocks in cache)

cache address: 00 ~ 111



mem address: 00 00 ~ 11 111
last three bits determine cache add
4 men blocks are in one cache block

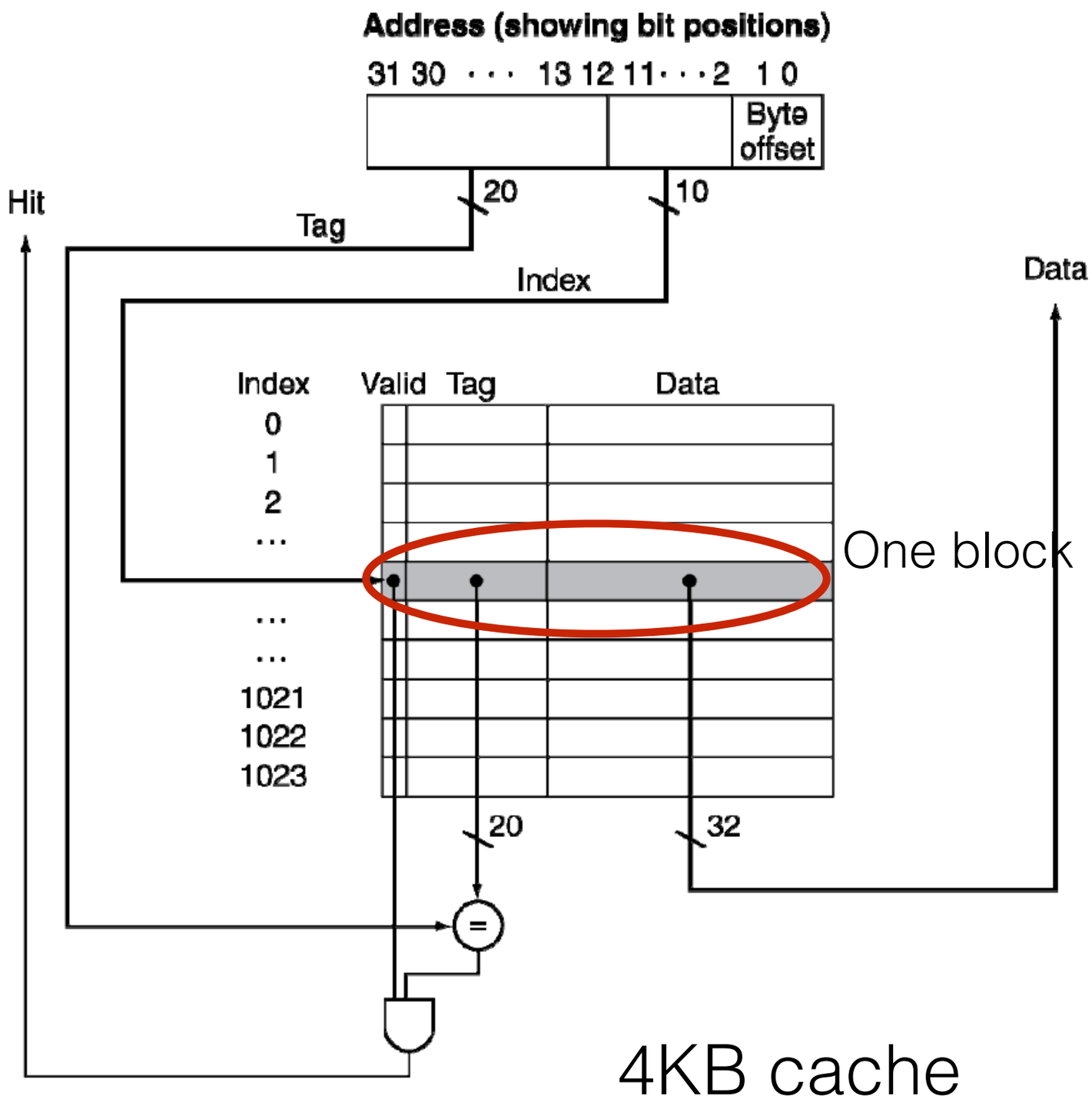
00 001
01 001
10 001
11 001

00 001 : 5 bits to tell from 2^5 possibilities
only 4 possibilities need to be differentiate
first 2 bits are enough.

Review

- **Cache (SRAM) data access**
  - How to know if there is data, or it is empty?
  - **Answer:**
    - valid bit (1 bit).

Review



4KB cache

Review

- **Access data from cache (temporal locality)**
- **Cache (SRAM) with larger block size (spatial locality)**
  - Improve space efficiency
  - by reducing addressing redundancy

Review

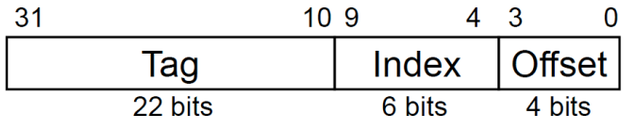
- **Cache (SRAM) with larger larger block size**
  - Improve space efficiency by reducing addressing redundancy'
  - addressing? select in a block with shift

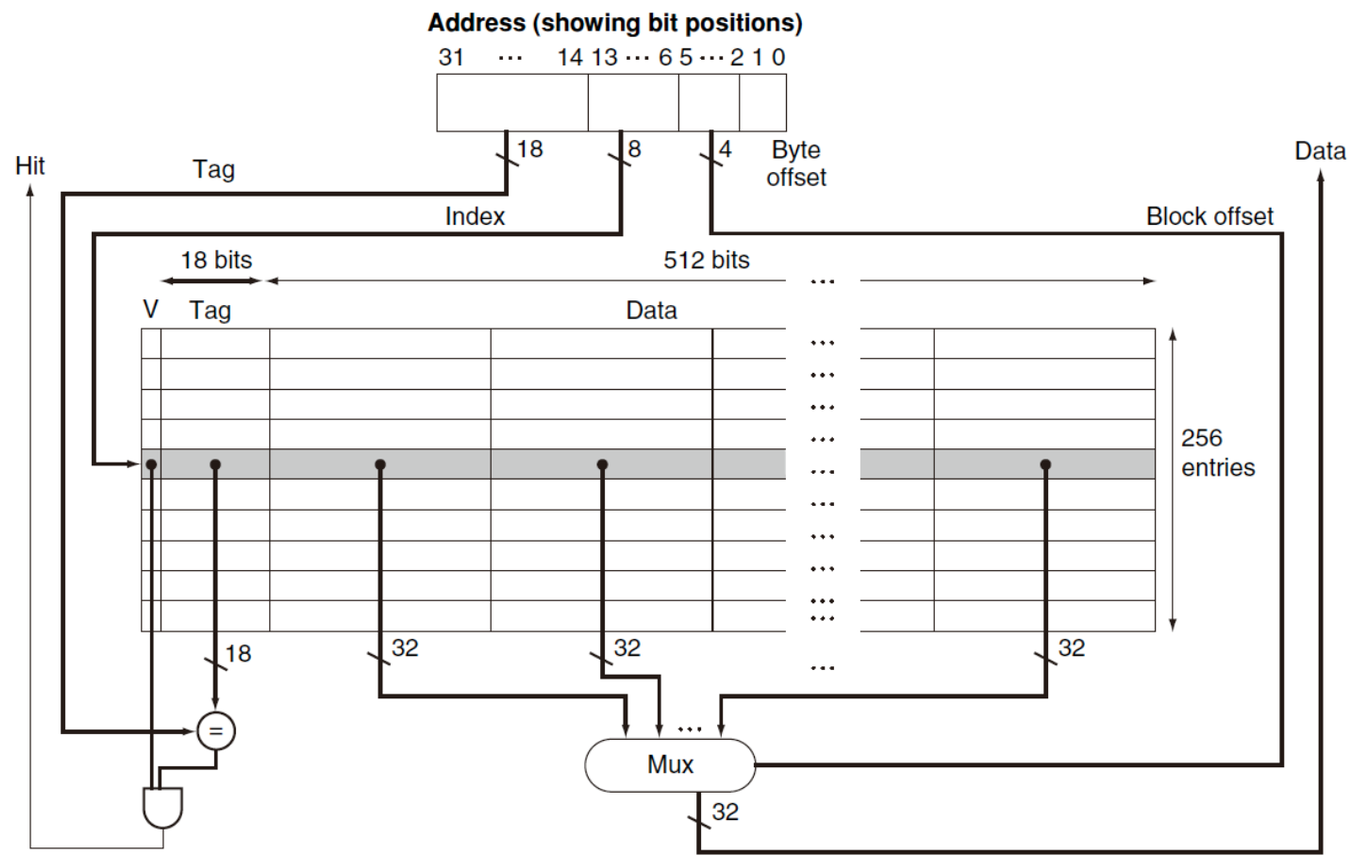
64 blocks, 16 bytes/block

- To what block number does address 1200 map?

Block address = $\lfloor 1200/16 \rfloor = 75$

Block number = 75 modulo 64 = 11



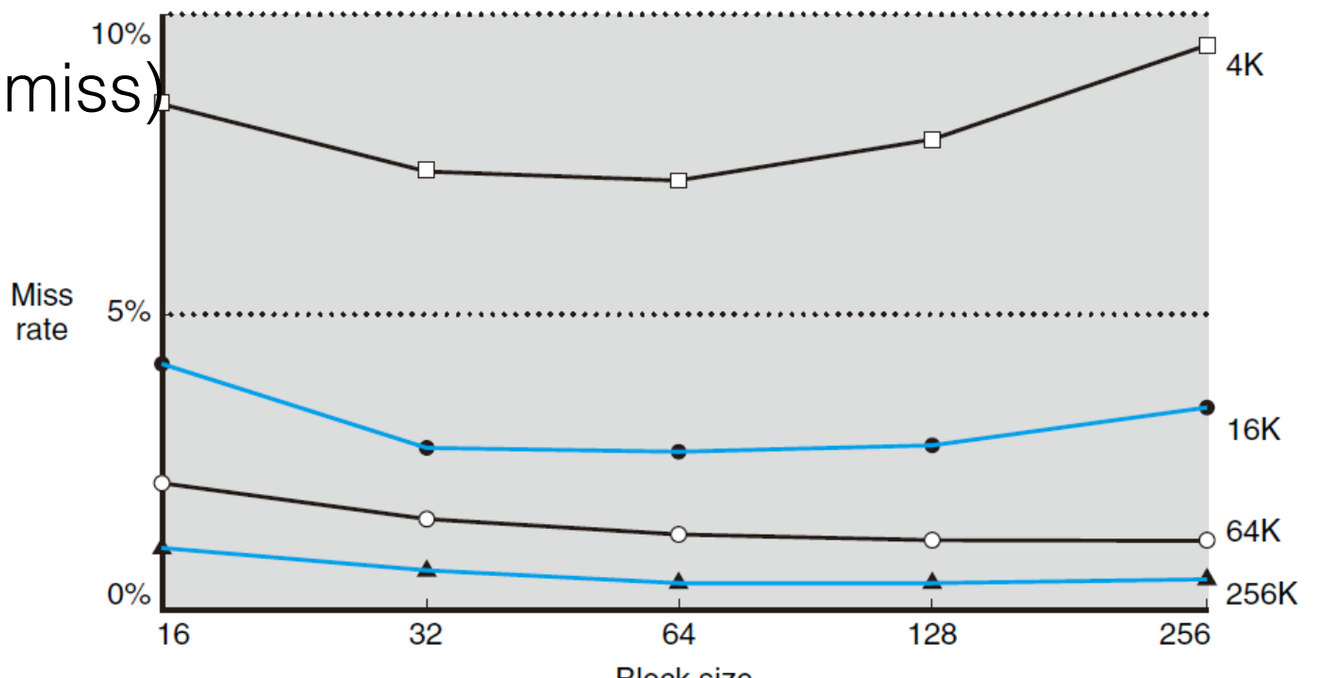


Review

- **Trade-off: Cache (SRAM) with larger larger block size**

Good:
space efficiency (data/total size rate)
spatial locality

bad:
longer transfer time
competition->conflict (higher conflict miss)



Review

- **Work with memory hierarchy**
  - **Read**
    - **hit**
    - **miss:**
      - Stall the CPU pipeline
      - Fetch block from next level of hierarchy
      - Restart instruction fetch / Complete data access
  - **Write**

Review

- **Work with memory hierarchy**
  - **Write**
    - **Hit:**
      - **Write through:** update memory. program wait.
        - Improvement: **write buffer:** Holds data waiting to be written to memory
        - When write buffer is full: hold program.
      - **Write back:** Keep track of whether each block is dirty. When a dirty block is replaced, write it to memory.
        - Improvement: **write buffer:** Holds data waiting to be written to memory
    - **Miss:**

Review

- **Work with memory hierarchy**
    - **Miss:**
        - **write through case:**
            - write allocate
            - write around (no write allocate)
        - **write back case:**
            - dirty block write back to mem (put into write buffer);
            - then fetch block, write cache block for new address (and tag it as dirty)

Review

- **Trade-offs of write-back and write-through**

  - Benefits of Write-Through?
    - Memory and cache are always synchronized and consistent.
  - Downsides?
    - Every write *must* perform a memory access.
    - A write buffer can be used in conjunction with a write-through cache. Write to the write buffer after which the processor is allowed to continue (unless there are further memory accesses) while the buffer writes to main memory.

Review

- **Trade-offs of write-back and write-through**

Benefits of Write-Back?

- Write-through requires 1 write to memory for every store instruction. Write-back effectively collects all of the writes to a particular block and needs to perform one write to memory.
- This is much better compared to write through if you're frequently writing to the same block (ie. when you have store locality).

Downsides?

- Write-through pays the cost of writing to memory for stores, but write-back may mean paying the cost of writing to memory for loads.
- Now when you load, at worst you may need to write a block to memory, then read a block from memory.
- Load performance may be more critical.

Review

- **Mem (cache) efficiency evaluation**
- **Components of CPU time**
- Program execution cycles: includes cache hit time
- Memory stall cycles: mainly from cache misses

Review

- **Mem (cache) efficiency evaluation**

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

access instructions / program

unit: cycles

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

- **Average memory access time (AMAT)**
  - AMAT = Hit time + Miss rate × Miss penalty

- **When CPU performance increased. Miss penalty becomes more significant**

Review

- **Associative Caches**
  - ***Fully associative***: Allow a given block to go in any cache entry
  - None associative (***direct mapped***): assumption as before
  - ***n-way set associative***: Each set contains n entries

Review

For a cache with 8 entries

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	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Review

- **Trade-offs of Associative Caches**
  - Fully associative:
    - More searching time. (hit time)
    - Less hit miss ratio (why?)
  - None associative:
    - less searching time (no searching at all)
    - More hit miss ratio

Review

- **Trade-offs of Associative Caches: example**

Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	**Mem[8]**		Mem[6]	

2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

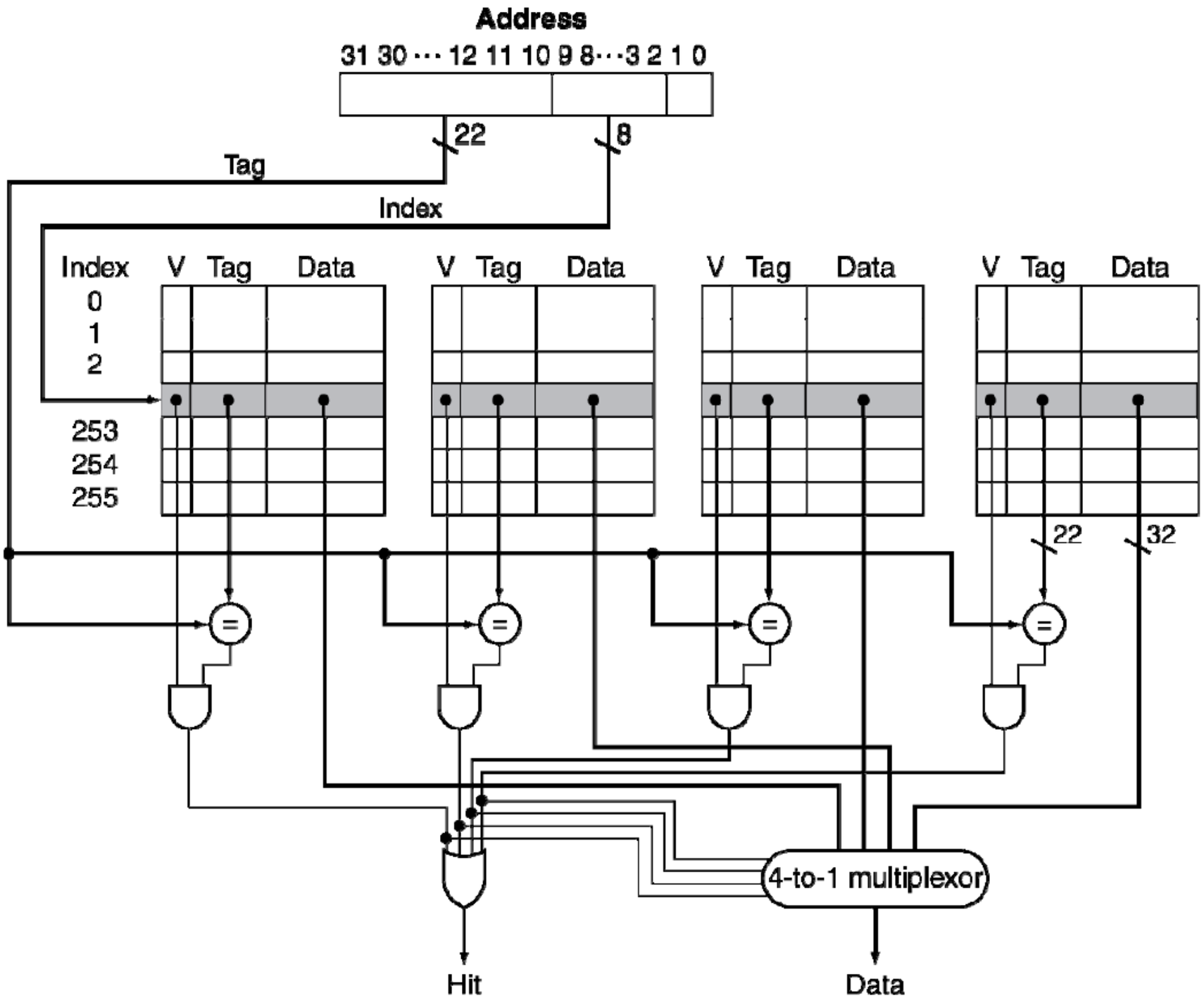
Compare 4-block caches

- Direct mapped, 2-way set associative, fully associative
- Block access sequence: 0, 8, 0, 6, 8

Review

- Cold Miss/Compulsory Miss: A block has never been accessed before. Therefore it cannot be in the cache (ignoring pre-fetching).
- Capacity Miss: The cache is completely full and you access a new block.
- Conflict Miss: The cache ISN'T full, but because of how the cache is organized, the block that was previously brought in had to be evicted (even though there IS space in the cache)

Review


Address
31 30 ··· 12 11 10 9 8···3 2 1 0
22
8
Tag
Index
Index
V
Tag
Data
0
1
2
253
254
255
V
Tag
Data
V
Tag
Data
V
Tag
Data
22
32
=
=
=
=
4-to-1 multiplexor
Hit
Data

Review

- **Replacement policy of Associative Caches**
  - **Direct mapped: no need for policy**
  - **set associative: many types of policy**
    - Least-recently used (LRU)
    - Least-Frequently Used (LFU)
    - First In First Out (FIFO)
    - Last In First Out (LIFO)
    - Random
    - etc.

Review

- **Multilevel of Associative Caches**
  - **SRAM (cache)**
    - Cache L1: focus on quick retrieve
    - Cache L2: focus on low miss rate
    - (Cache L3)
  - **DRAM (main memory)**
  - **Disk/Flash**

Review

- **Multilevel of Associative Caches: example**

  Given

  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns

Review

- **Multilevel of Associative Caches: example**

With just primary cache

- Miss penalty = 100ns/0.25ns = 400 cycles
- Effective CPI = 1 + 0.02 × 400 = 9

Now add L-2 cache

- Access time = 5ns
- Miss Rate = 10%

Primary miss with L-2 hit

- Penalty = 5ns/0.25ns = 20 cycles

Primary miss with L-2 miss

- Extra penalty = 400 cycles

CPI = 1 + 0.02 × (20 + 0.10 × 400) = 2.2

Performance ratio = 9/2.2 = 4.1

Review

- **Main points**

Often direct mapped level 1 cache (closest to CPU), associative further away

Split I and D level 1 caches (for throughput rather than miss rate), unified further away.

Write-through and write-back are both common, but never write-through all the way to memory.

Cache line size at least 32 bytes, getting larger.

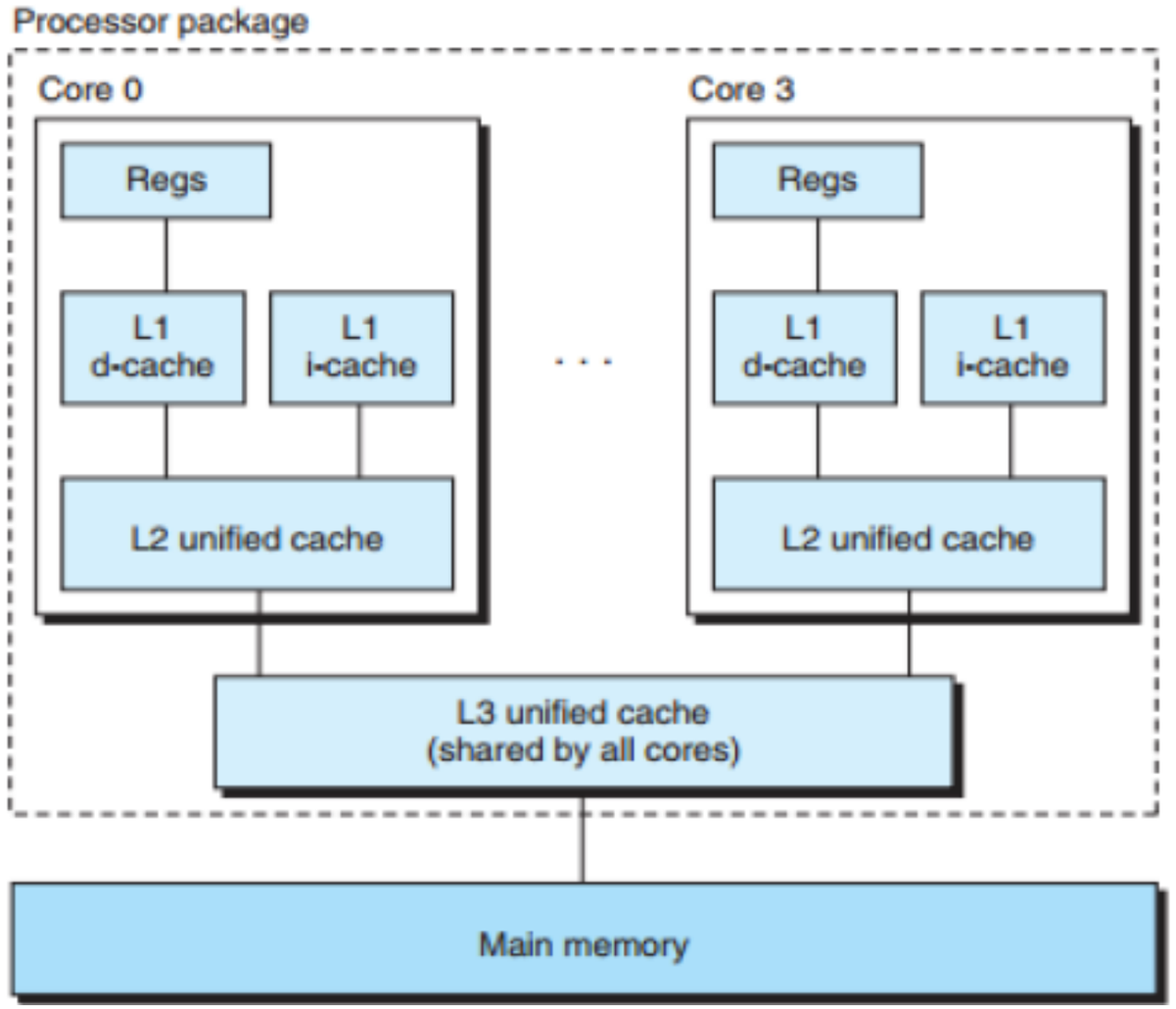
Usually cache is non-blocking

- **processor doesn't stall on a miss, but only on the use of a miss (if even then)**
- **this means the cache must be able to handle multiple outstanding accesses.**

Cache design presents many options (block size, cache size, associativity) that an architect must combine to minimize miss rate and access time to maximize performance.

Review

- **Main points**



Review

- **Main points**
  - **cache hit: access where data is found in the cache**
  - **cache miss: access where data is NOT in the cache**
  - **cache block size or cache line size: the amount of data that gets transferred on a cache miss.**
  - **instruction cache (I-cache): cache that only holds instructions**
  - **data cache (D-cache): cache that only holds data**
  - **unified cache: cache that holds both data & instructions**

A typical processor today has separate "Level 1" I- and D-caches on the same chip as the processor (and possibly a larger, unified "L2" on-chip cache), and larger L2 (or L3) unified cache on a separate chip.

Review

- **Main points**
  - Total CPI (TCPI) = Base CPI (BCPI) + Mem CPI (MCPI)
  - BCPI = Peak CPI (PCPI) + Data Hazard Impact + Control Hazard Impact
  - PCPI = Ideal CPI without any hazards/stalls

Sample example 1

8-blocks, 1 word/block, direct mapped

Initial state

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

10 110

11 010

10 000

00 011

10 000

10 010

How many miss/hit?
What is the final state?

Sample example 1

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

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

Sample example 1

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

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

Sample example 1

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

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

Sample example 1

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

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

Sample example 2

- **Mem (cache) efficiency evaluation**
  - Given
    - I-cache miss rate = 2%
    - D-cache miss rate = 4%
    - Miss penalty = 100 cycles
    - Base CPI (ideal cache) = 2
    - Load & stores are 36% of instructions

Actual (estimation of) CPI?

Sample example 2

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions

I instructions.
I * 0.02 *100 = 2I
I * 0.36 * 0.04 * 100 = 1.44I
(I * 2 + 2I + 1.44I) / I = 5.44

Sample example 3

- Normal 5-Stage Pipeline
- L1 Cache
  - I$ Miss Rate: 2%
  - D$ Miss Rate: 10%
  - Both caches are **write-back**, write-allocate
- Memory (no other caches):
  - Hit time: 400 cycles
- Assume caches are full
- 25% of data cache blocks are dirty
- 40% of instructions are lw and sw.
- sw incurs cache access penalty (no write buffer)

Sample example 3

- MCPI =

  (% of I$ miss) * Insn Memory Penalty

  + (% of mem insn) * (D$ miss rate) * Data Memory Penalty

- Instruction Memory Penalty = 400?
- Data Memory Penalty = 400?
- Since write-back and writing back to memory does incur a penalty, we have to consider that evicting a block may mean an additional memory operation. 25% are dirty
- Data Memory Penalty = 400 + .25*400

Sample example 3

- Do we have to make the same consideration for the I$?
  - No
- Is this question asked in a suspicious manner that definitely means the answer is no?
  - Yes
- We don't ever write to the I$. It's read only. Therefore, the write policy doesn't matter. There can be no dirty blocks.

Sample example 3

- MCPI =

  (% of I$ miss) * Insn Memory Penalty

  + (% of mem insn) * (D$ miss rate) * Data Memory Penalty.

- Data Memory Penalty = 400 + .25*400
- MCPI = .02 * 400 + .4*.1*(400 + .25*400)

  = 8 + .04*(500)

  = 28