Review Session 5
Fri. Feb 16, 2:15 pm – 3:05 pm
Thornton 102

1. Agenda

- Announcements
- Homework hints
- Memory hierarchy
- Cache access example

2. Announcements

- Homework 3, due Tuesday, February 20.
- The Z drive problem with lab pc accounts fixed.
- You may pick up your quizzes from Teresa Lynn's office at Gates 310.
- Verify that your quiz grade on eeclass is entered correctly.
- Check for any clerical mistakes. Regrades are entertained. Please follow the guideline as given on the class website under quizzes.

3. Homework hints

Problem 1:

- a) Consider all dependencies, not only data hazards.
- b) No forwarding means the result of an instruction is available only after the write back stage.
- Hints for drawing pipeline timing diagram:

```
instr1 F D X M W
lw       F D X M W
add        F D - X M W
lw           F - D X M W
add              F D - X M W
instr6             F - D X M W
```

Since the same hardware cannot be used for 2 different instructions at any time, the same letter cannot appear more than one in each column.

How to insert bubbles correctly? Bubbles are actually this:

```
instr1       F D X M W
add            F D D X M W
lw               F F D X M W
instr4               F D X M W
```

But you should not draw them as

```
instr1       F D X M W
add            F - D X M W              WRONG!
lw                 F D X M W
instr4               F D X M W
```

Once a bubble appears, everything beneath it should be a bubble.

Problem 2:

- a) 2 instruction fetch stages F1 and F2, 2 memory access stages M1 and M2. F1 and F2 can stagger in time, so can M1 and M2 but no data can be retrieved until after M2.

- Discuss impact on number of instruction scheduling and forwarding logic.

- b) Consider branch prediction and how more stages affect the mispredict penalty.

Problem 3:

- Assume PC = 100 just before the positive clock edge.
- Assume the sub instruction was already in the IF stage before the clock edge.

- Use this table:

Stage	*Field*	*Width*	*Value (hex)*	*Value (dec)*
IF/ID	*PC + 4*	*32*		
	Instruction	*32*		
ID/EX	*PC + 4*	*32*		
	Read Data 1	*32*		
	Read Data 2	*32*		
	Immediate	*32*		
	Load Destination	*5*		
	Register Destination	*5*		
EX/MEM	*Branch Target*	*32*		
	ALU Result	*32*		
	Zero	*1*		
	Store Data	*32*		
	Destination Register	*5*		
MEM/WB	*Memory Data*	*32*		
	ALU Result	*32*		
	Destination Register	*5*		

Problem 4:

- **`lw`** followed by **`sw`**

```
lw                F D X M W
sw                  F D X - M W
```

- Syntax for expressing forwarding and stalling logic:

if (MEM/WB.MemRead) and (EX/MEM.MemWrite) and … etc.
    fwd = 1
else
    fwd = 0

- Draw diagram showing additional functional blocks e.g. mux.

Problem 5:

- b) The lw instruction involves activity in all 5 pipeline stages. With zero offset addressing, think about which of these 5 stages will no longer be active within a single instruction, could one then change the number of pipeline stages?

- c) Note that the new CPI for lw is 1.5 without stalling. Given the fraction of lw followed by dependent ALU instructions, how do you calculate the CPI of lw in the normal 5 stage pipeline MIPS incorporating the effect of stalling?

- c) Adjust the number if instructions due to 0-base addressing mode and renormalize to obtain the new instruction mix.

Problem 6:

- In case of misses, indicate what kind it is.
- Note the difference between "16 one-word blocks" and "16 words now arranged with a two-word block size".
- In cases of 2-word blocks, both of the cache blocks are filled when anyone of them is read from main memory.
- NB: a cold miss does **not** mean that the cache location where the data is going to be stored is being occupied for the very first time. A cold miss **does** mean the data being sought by the current memory reference has never been

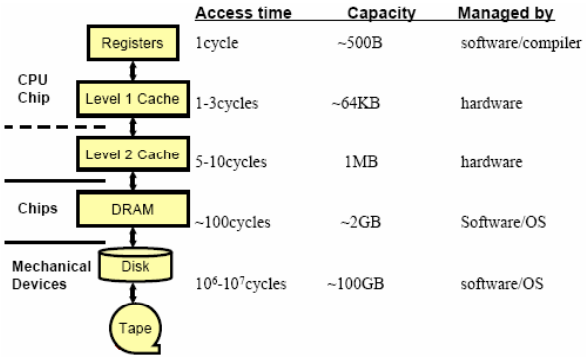
- referenced previously, therefore it cannot possibly be in the cache and thus we have to go to the next level in the memory hierarchy. A cold miss has nothing to do with whether or not the cache location where the retrieved data will be stored was already occupied by other data.
- Simple example: memory reference = {1, 5, 11, 4, 27}

	Tag	Data
0		
1		Mem[1]
2		
3		
4		
5		
6		
7		

4. Memory hierarchy

Need memory that is both fast and large, but fast is expensive and large is slow - tradeoff between cost and speed. What do we do? Solution: use a hierarchy of memories.

- Small, fast, expensive caches near the processor
- Large, slow, and cheap memory further away feeding the faster memories
- We can do this via exploiting spatial and temporal localities in programs



Cache summary:

- Direct mapped cache, full associative cache, set associative cache
- Tag, valid, index and offset bits – depends on cache and block sizes.
- Write/update policy.

Cache write: what happens when there is a miss?

Write through:

- Pick replacement.
- Fetch block.
- Write to cache.
- Write to main/next level memory – write buffer.

Write back:

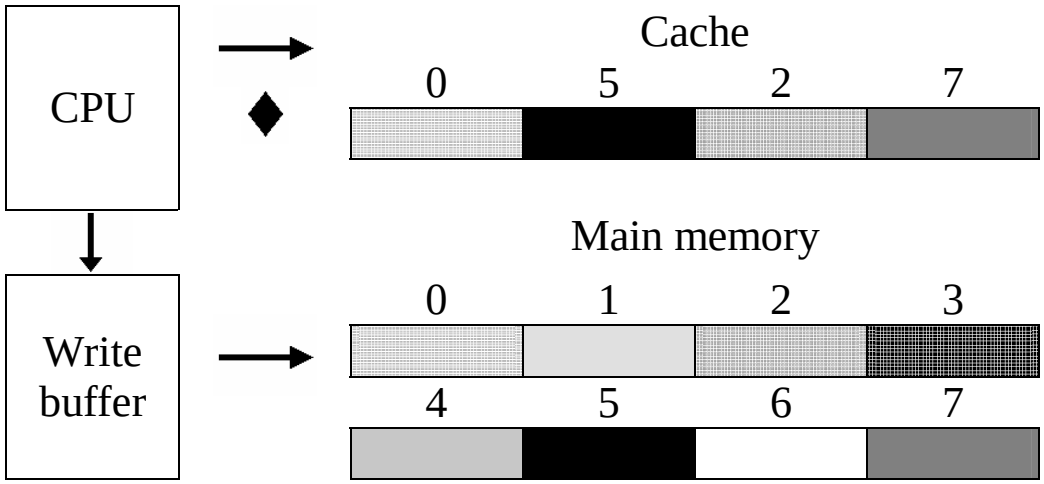
- Pick replacement.
- Write back.
- Fetch block.
- Write to cache.

- Address mapping, hits, compulsory and conflict misses.

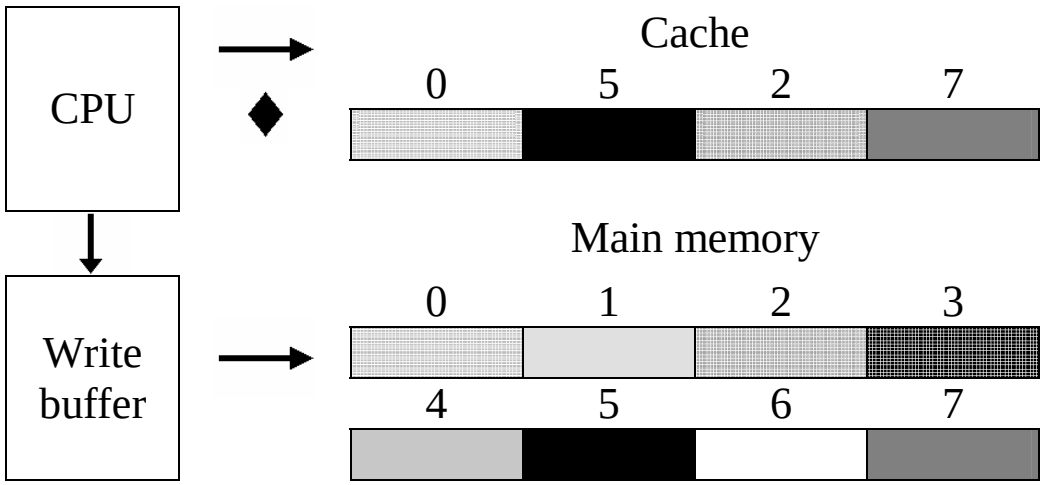
5. Cache access examples

Write through example

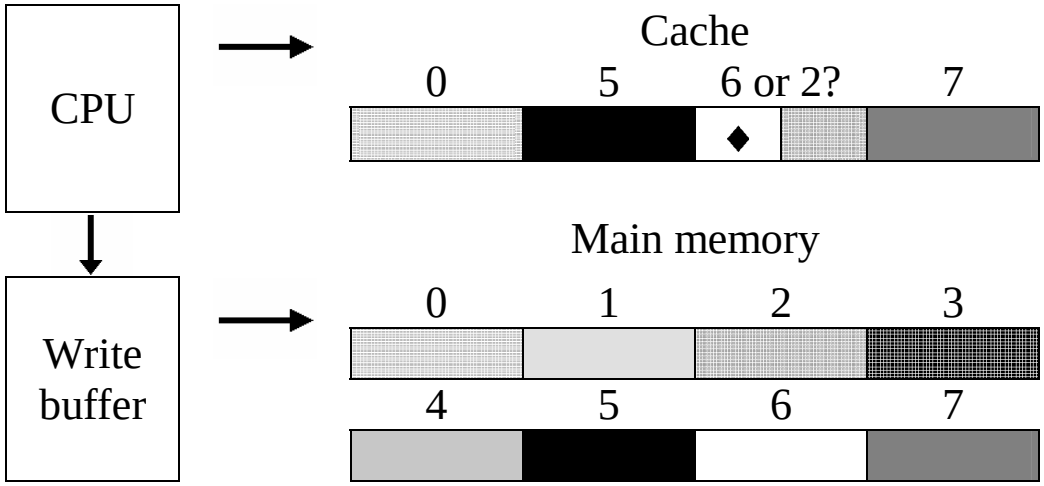
Assume: block size of 2 words, direct mapped cache.
Step 1: CPU wants to write ♦ to 1st word of block at address 6.



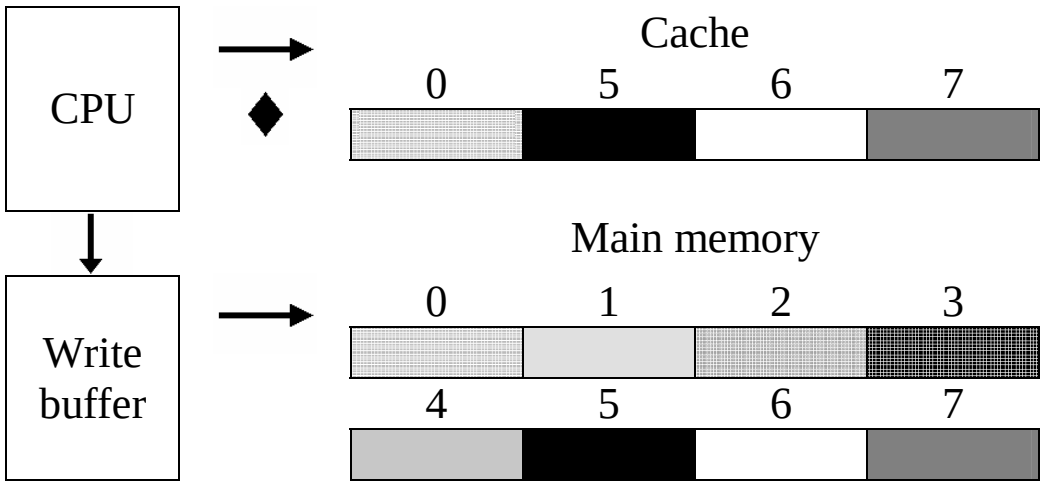
Step 2: Pick replacement in cache, direct mapped → replace 2 in cache.



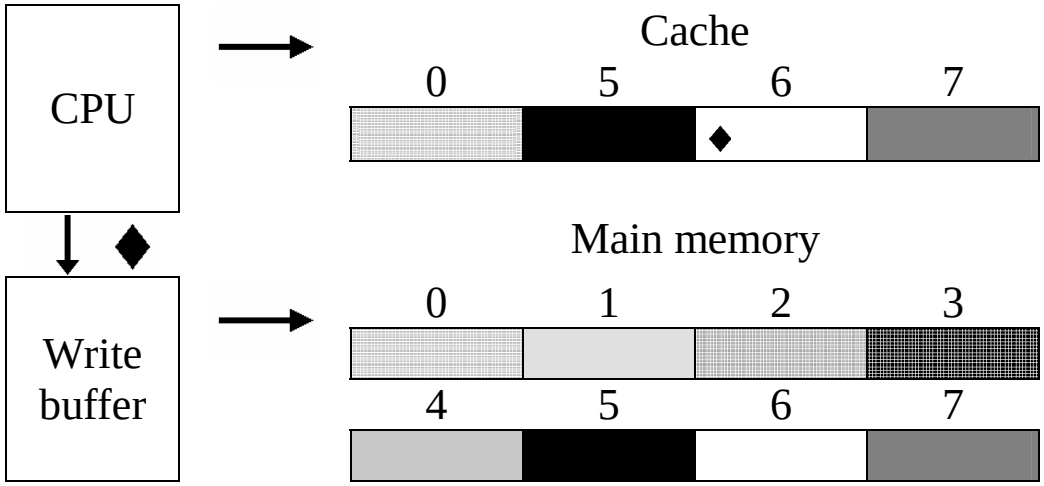
Wrong step 3: Simply overwrite the existing 1st word of block 2.



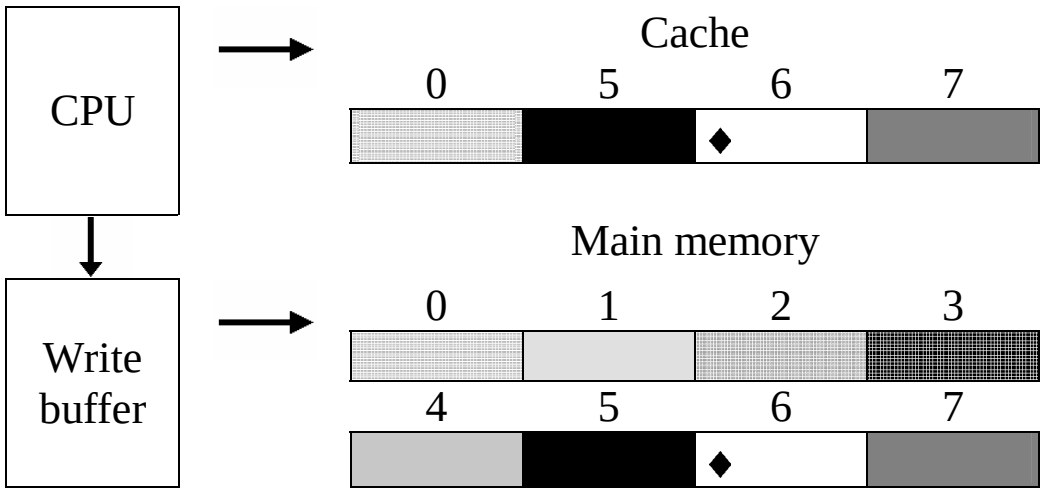
Step 3: Fetch block 6 from main memory.



Step 4: Write to cache.



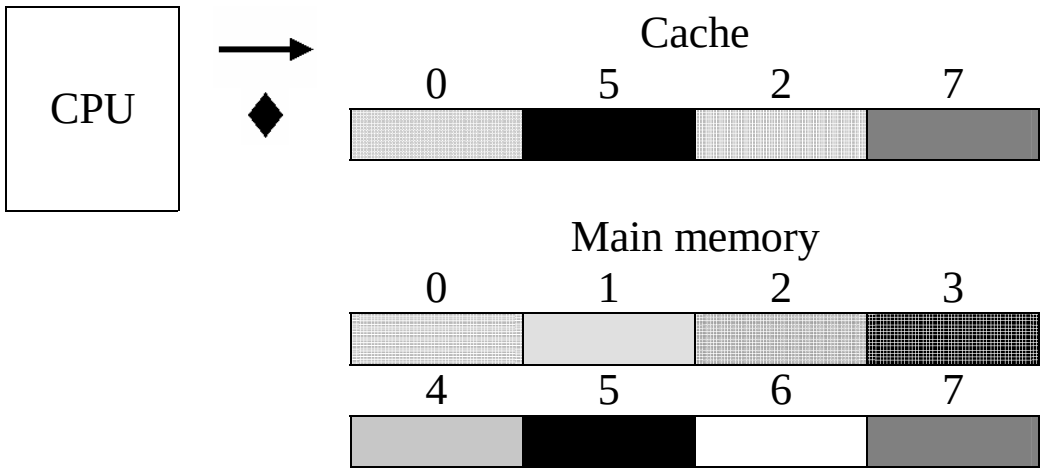
Step 5: Write to main memory.



Write back example

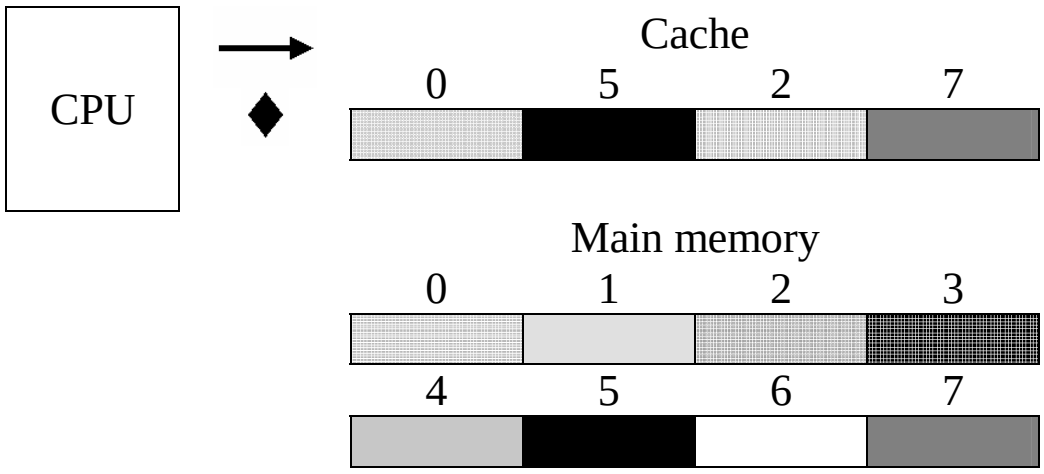
Assume: block size of 2 words, direct mapped cache.

Step 1: CPU wants to write ♦ to 1st word of block at address 6.

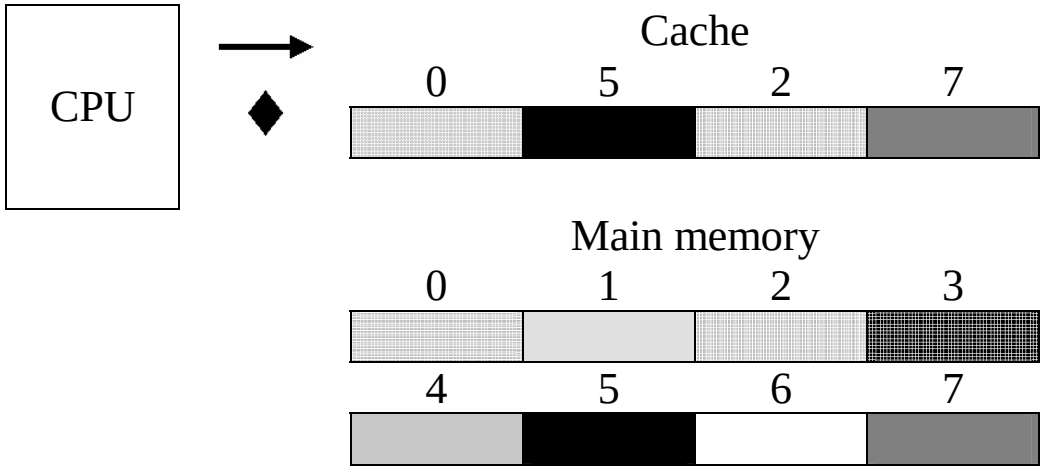


EE108B
Winter 2006 – 2007 Session TA: Yi Gu

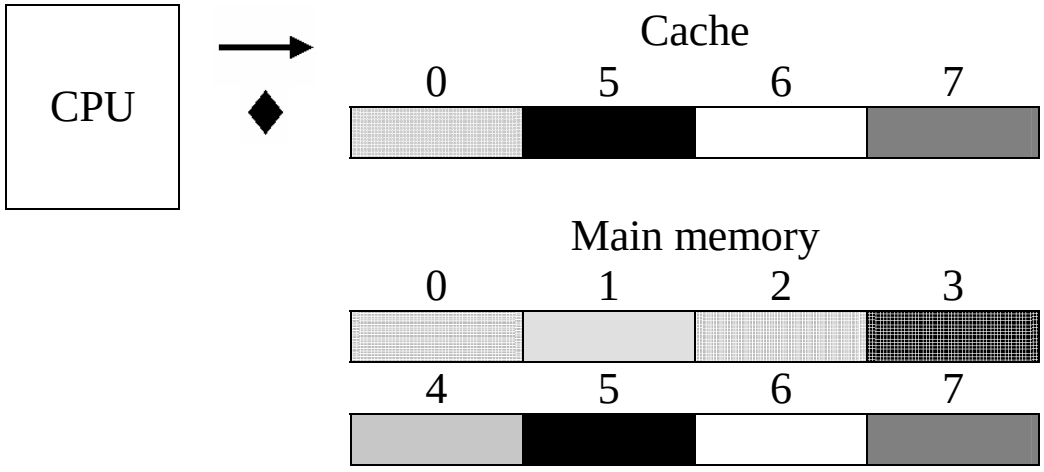
Step 2: Pick replacement in cache, direct mapped → replace 2 in cache.



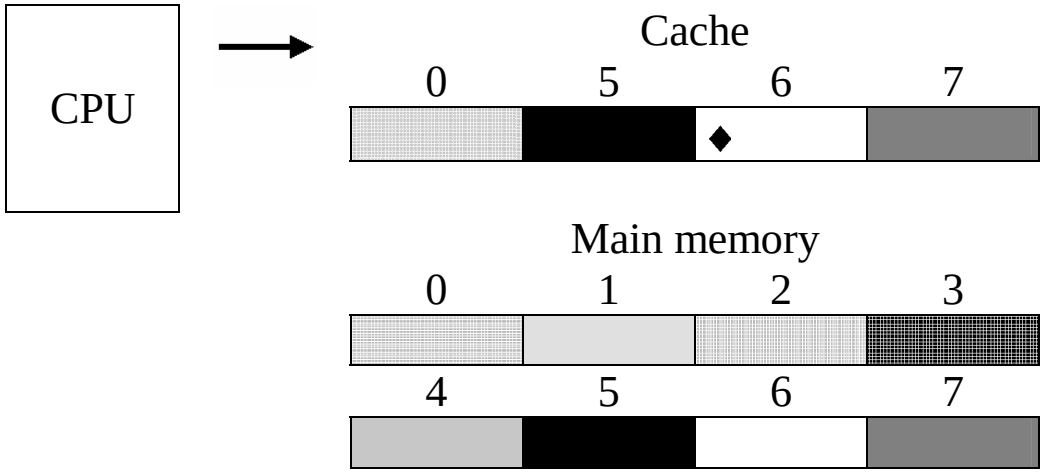
Step 3: Write back.



Step 4: Fetch block from memory.



Step 5: Write to cache.



	Write through				Write back	
	Write allocate		No write allocate		Write allocate	
Steps	fetch on miss	no fetch on miss	write around	write invalidate	fetch on miss	no fetch on miss
1	pick replacement	pick replacement			pick re-placement	pick re-placement
2				invalidate tag	[write back]	[write back]
3	fetch block				fetch block	
4	write cache	write partial cache			write cache	write partial cache
5	write memory	write memory	write memory	write memory		