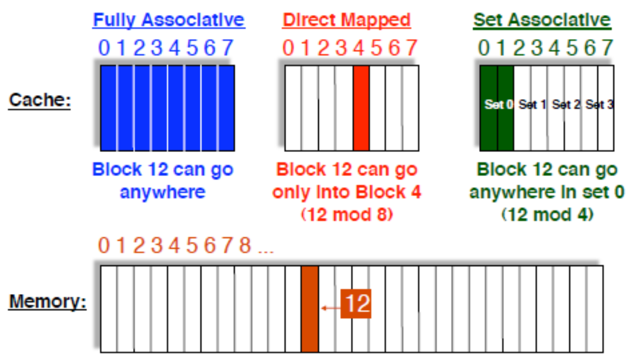
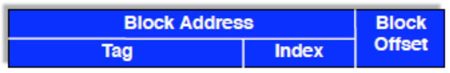
Where can a block be placed in a cache?



How is a block found in the cache?



- Block offset field selects data from block
  - (i.e. address of desired data within block)
- Index field selects a specific set
- Tag field is compared against it for a hit

- Could we compare on more of address than the tag?
  - Not necessary; checking index is redundant
    - Used to select set to be checked
    - Ex.: Address stored in set 0 must have 0 in index field
  - Offset not necessary in comparison – entire block is present or not and all block offsets must match

Exercise 5.4

For a direct-mapped cache design with 32-bit address, the following bits of the address are used to access the cache.

	Tag	Index	Offset
a.	31–10	9–4	3–0
b.	31–12	11–15	4–0

5.4.1 [5] <5.2> What is the cache line size (in words)?

5.4.2 [5] <5.2> How many entries does the cache have?

5.4.3 [5] <5.2> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Starting from power on, the following byte-addressed cache references are recorded.

Address	0	4	16	132	232	160	1024	30	140	3100	180	2180

5.4.4 [10] <5.2> How many blocks are replaced?

5.4.5 [10] <5.2> What is the hit ratio?

5.4.6 [20] <5.2> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>.

Solution 5.4

5.4.1

a.	4
b.	8

5.4.2

a.	64
b.	128

5.4.3

a.	1 + (22/8/16) = 1.172
b.	1 + (20/8/32) = 1.078

5.4.4 3

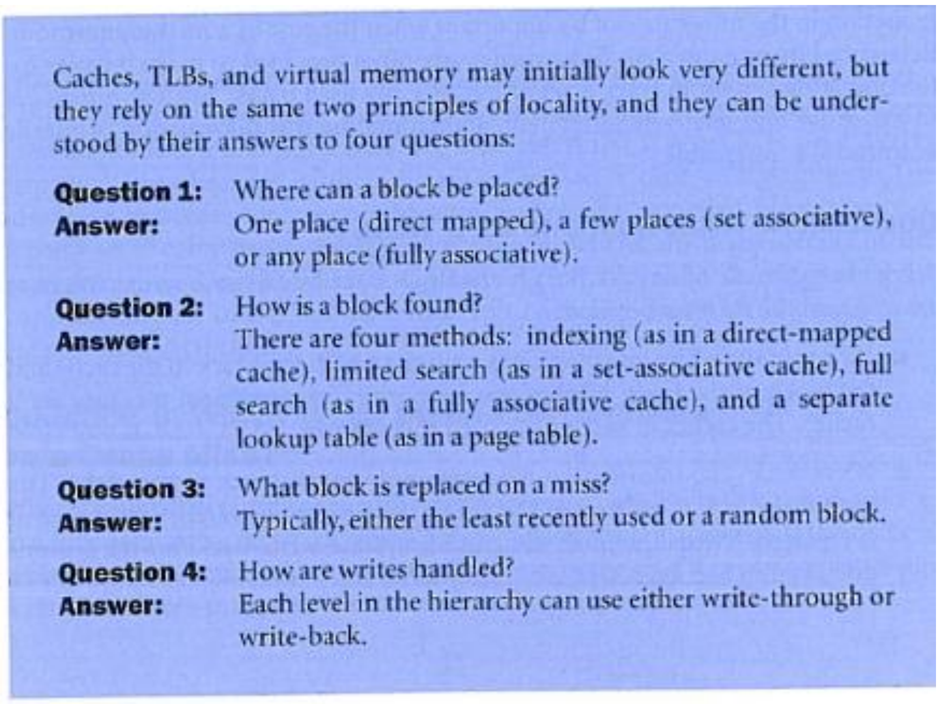
Address	0	4	16	132	232	160	1024	30	140	3100	180	2180
Line ID	0	0	1	8	14	10	0	1	9	1	11	8
Hit/miss	M	H	M	M	M	M	M	H	H	M	M	M
Replace	N	N	N	N	N	N	Y	N	N	Y	N	Y

5.4.5 0.25

5.4.6 <Index, tag, data>:

<000001_2, 0001_2, mem[1024]>
<000001_2, 0011_2, mem[16]>
<001011_2, 0000_2, mem[176]>
<001000_2, 0010_2, mem[2176]>
<001110_2, 0000_2, mem[224]>
<001010_2, 0000_2, mem[160]>

The BIG Picture



Caches, TLBs, and virtual memory may initially look very different, but they rely on the same two principles of locality, and they can be understood by their answers to four questions:

Question 1: Where can a block be placed?
Answer: One place (direct mapped), a few places (set associative), or any place (fully associative).

Question 2: How is a block found?
Answer: There are four methods: indexing (as in a direct-mapped cache), limited search (as in a set-associative cache), full search (as in a fully associative cache), and a separate lookup table (as in a page table).

Question 3: What block is replaced on a miss?
Answer: Typically, either the least recently used or a random block.

Question 4: How are writes handled?
Answer: Each level in the hierarchy can use either write-through or write-back.

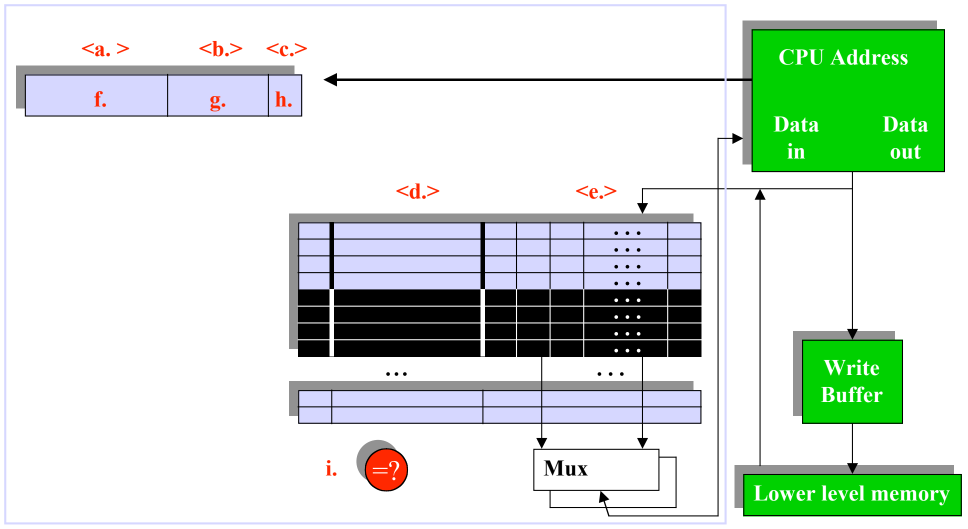
Question 1:

Part A:

You have been asked to design a cache with the following properties:

- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The cache is direct mapped
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache
- Addresses are to the word

Pictured below is the general structure of a cache. There are 8 fields (labeled a, b, c, d, e, f, g, and h). In the space below you'll need to indicate the proper name or number of bits for a particular portion of this cache configuration. Whether a name or number should be entered will be specified:



Suggestion – you don't have to fill in the answers "in order". In fact, you need to determine the number of bits associated with the index and offset for example before determining the number of bits in the tag. I'll present the answer this way:

f. (name) _______

- You are being asked to show what part of a physical address form the index, offset, and tag. < f > refers to the most significant bits of the address – so this is the tag.

g. (name) _______

- It follows that the next part of the address is the index.

h. (name) _______

- The least significant bits form the offset.

c. (number) ______

- There are 2^{11} bits / block and there are 2^5 bits / word. Thus there are 2^6 words / block so we need 6 bits of offset.

b. (number) ______

- There are 2^{11} blocks and the cache is direct mapped (or "1-way set associative"). Therefore, we need 11 bits of index.

a. (number) ______

- The remaining bits form the tag. Thus, 32 – 6 – 11 → 15 bits of tag.

d. (number) ______

- Field < d > refers to the fact that a tag must be stored in each block. Thus, 15 bits are kept in each block.

e. (number) ______

- Field < e > asks you to specify the total number of bits / block. This is 2048.

i. What 3 things must be compared at ? to determine if the cache entry is useable or not?

- We need to compare the valid bit associated with the block, the tag stored in the block, and the tag associated with the physical address. The tags should be the same and the valid bit should be 1.

j. What is the total size of the cache?

- There are 2048 blocks in the cache and there are 2048 bits / block.
    - There are 8 bits / byte
    - Thus, there are 256 bytes / block
- 2048 blocks x 256 bytes / block → 2^{19} bytes (or 0.5 MB)

Part B:
Now, let's consider what happens if we make our cache 2-way set-associative instead of direct mapped. However, as before, the following still applies:

- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache
- Addresses are to the word

Number of bits in offset?
- There are still 6 bits in the offset; data is still word addressed

Number of bits in index?
- We now need one less bit of index because we address to the set
    - 2^{11} blocks / 2^1 blocks/set = 2^{10} sets
    - (10 bits of index needed)

Number of bits in tag?

- 32 – 6 – 10 = 16 bits.

<u>Part</u> <u>C</u>:
Now, let's consider what happens if we make our cache 4-way set-associative instead of direct mapped. However, as before, the following still applies:

- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache
- Addresses are to the word

Number of bits in offset?

- There are still 6 bits in the offset; data is still word addressed

Number of bits in index?

- We now need one less bit of index because we address to the set
    - 2^{11} blocks / 2^2 blocks/set = 2^9 sets
    - (9 bits of index needed)

Number of bits in tag?

- 32 – 6 – 9 = 17 bits.

<u>Part</u> <u>D</u>:
Now, let's consider what happens if data is byte addressable. We'll keep the cache 4-way set associative for this question. However, as before, the following still applies:

- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache

Number of bits in offset?

- There *were* 6 bits in the offset
- Now, each of the 4 bytes of a given word can be individually addressed
    - Therefore, we need 2 more bits of address *per word*
- Thus, 2^6 words * 2^2 bytes / word $\rightarrow$ 2^8
    - 8 bits of offset are needed.

Number of bits in index?

- We need the same number of index bits as in Part D
    - 2^{11} blocks / 2^2 blocks/set = 2^9 sets
    - (9 bits of index needed)

Number of bits in tag?

- 32 – 8 – 9 = 15 bits.

<u>Part</u> <u>E</u>:

Now, let's consider what happens if the size of our physical address changes from 32 bits to 64 bits. We'll keep the cache 4-way set associative and data will still be addressable to the byte for this question. However, as before, the following still applies:

- A cache block will contain 2048 bits of data
- There are 2048 blocks in the cache

Number of bits in offset?

- 8 (as above)

Number of bits in index?

- 9 (as above)

Number of bits in offset?

- 64 – 8 – 9 → 47 bits

<u>Question</u> <u>2</u>:

Our system has a main memory with 16 megabytes of addressable locations and a 32 kilobyte direct mapped cache with 8 bytes per block. The minimum addressable unit is a byte.

<u>Part</u> <u>A</u>:

How many blocks are there in the cache?

- (1 block / 2^3 bytes) x (2^{15} bytes / cache) = 2^{12} blocks / cache
- Therefore, need 12 bits of index

<u>Part</u> <u>B</u>:

Show how the main memory address is partitioned.

- 16 MB of addressable locations implies 24 bits of address are needed (2^{24} = 16 MB)
- Therefore, need:
    - 3 bits of offset
    - 12 bits of index
    - 9 bits of tag

<u>Question</u> <u>3</u>:

Find the average memory access time for a processor given the following:

- The clock rate is 1 ns
- The miss penalty is 25 clock cycles
- 1% of instructions are not found in the cache.
- 5% of data references are not found in the cache
- 15% of memory accesses are for data.
- The memory system has a cache access time (including hit detection) of 1 clock cycle.
- Assume that the read and write miss penalties are the same and ignore other write stalls.

AMAT	=	Hit Time	+	Miss Rate	x	Miss Penalty
	=	1 ns	+	(0.01 x 0.85 + 0.05 x 0.15)	x	25 ns
	=	1 ns	+	0.16	x	25 ns
	=	1.4 ns				

Question 4:

Consider a cache with the following specs:

- It is 4-way set associative
- It holds 64 Kbytes of *data*
- Data words are 32 bits each
- Data words *are not* byte addressed, they are *word* addressed
- A physical address is 40 bits.
- There are 16 words per cache block
- A First in, First out replacement policy is used for each set
- All cache entries are initially empty (i.e. their valid bits are not set)

At startup, the following physical addresses (in hexidecimal) are supplied to this cache in the order shown below:

	MSB								LSB	
1	F	F	F	B	D	0	9	8	7	3
2	A	B	C	D	E	F	1	1	8	3
3	A	B	C	D	E	F	2	1	8	3
4	A	B	C	D	E	F	1	1	8	4
5	A	B	C	D	E	F	1	1	8	4
6	F	F	F	B	D	0	9	8	7	4
7	F	F	F	B	D	0	A	9	7	4
8	F	F	F	B	D	0	A	8	7	8
9	A	B	C	D	E	F	2	1	8	3

- There are 16 (2^4) addressable entries / block
  - Thus, we need 4 bits of offset
- How many bits of index are needed?
  - = 2^{16} bytes x (1 word / 2^2 bytes) x (1 block / 2^4 words) x (1 set / 2^2 blocks)
  - = 2^8 sets
  - Therefore, need 8 bits of index.
- The remaining (40 – 4 – 8 = 28) 28 bits form the tag

Reference	Set reference maps to:	Status	Comment
1	87	Compulsory Miss	1st access
2	18	Compulsory Miss	1st access
3	18	Compulsory Miss	1st access; tag different than reference 2
4	18	Hit	Data brought in during reference 2
5	18	Hit	Data brought in during reference 2
6	87	Hit	Data brought in during reference 1
7	97	Compulsory Miss	Nothing else has mapped to set 97
8	87	Compulsory Miss	Tag same as reference 7, but maps to different set
9	18	Hit	Data brought in during reference 3

The cache looks something like this:

Set	Data
18	Data brought in with Reference 2
	Data brought in with Reference 3
87	Data brought in with Reference 1
	Data brought in with Reference 8
97	Data brought in with Reference 7

<u>Question</u> <u>A</u>:
Q: How many sets of the cache has this pattern of accesses touched?
3

<u>Question</u> <u>B</u>:
Q: How many compulsory misses are there for this pattern of accesses?
5

<u>Question</u> <u>C</u>:
Q: How many hits would best be described as occurring because of spatial locality?
3: References 4, 5, and 6 (although answers could vary)

<u>Question</u> <u>D</u>:
Q: How many hits would best be described as occurring because of temporal locality?
1: Reference 9 (although answers could vary)

<u>Question</u> <u>E</u>:
Q: How many conflict misses are there for this pattern of accesses?
0

<u>Question</u> <u>F</u>:
Q: What is the overall miss rate for this pattern of accesses?
5 / 9