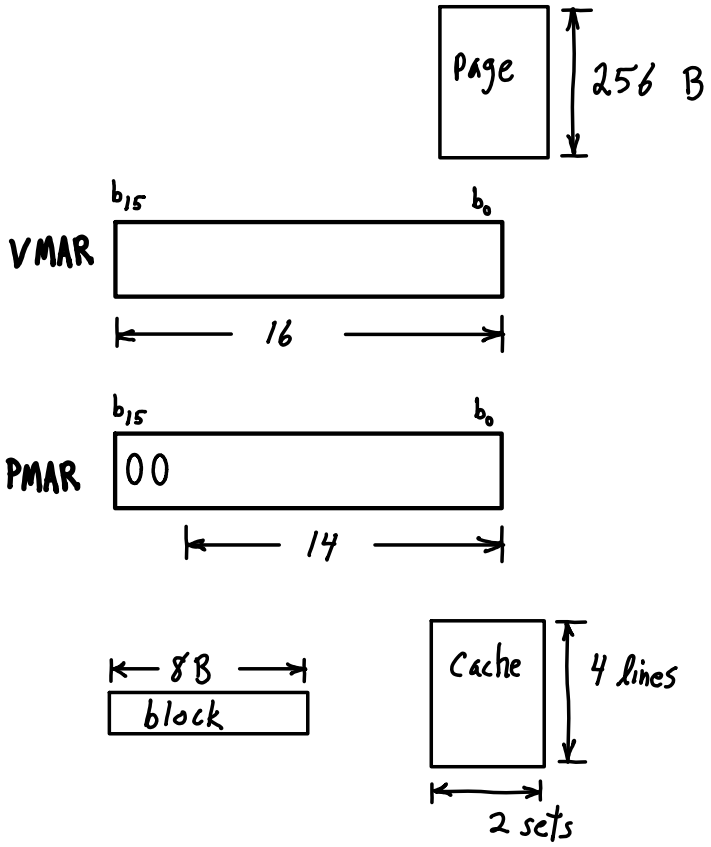
Lec-3-HW-1-cacheVM

Consider the following 16-bit machine (datapath and register width is 2B).

-- Virtual memory:
- -- byte addressable.
- -- 16-bit virtual addresses.
- -- **VMAR** is 16-bits.
- -- 256 B pages

-- Physical memory:
- -- byte addressable.
- -- 14-bit physical addresses.
- -- **PMAR** is 16-bits, upper 2 bits = 00, read-only.

-- Cache:
- -- 8B cache blocks;
- -- 2-way Set-Associative (SA): two Direct-Mapped (DM) caches running in parallel;
- -- 4 entries per DM;
- -- physically tagged, physically indexed
- -- **CMAR** has same features as **PMAR**



Q. Give the sizes of the following in bytes, words, cache blocks, and pages:

-- physical memory

-- virtual memory

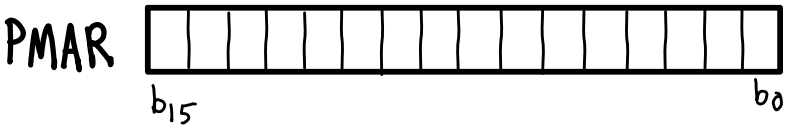
Q. How many words per page?

Q. How many cache blocks per page?

A memory address can be looked at in this format:

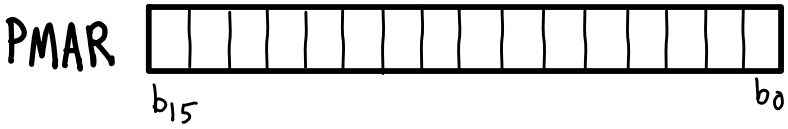
P# --- which page of virtual memory
(or **F#**, which physical memory frame)
bl# --- which block within the page
W# --- which word within the block
B# --- which byte within the word

Q. Given the physical memory address register (**PMAR**) shown below, indicate which bits correspond to each of the bit fields defined above (**F#**, **bl#**, **W#**, **B#**). Explain the size of each field.

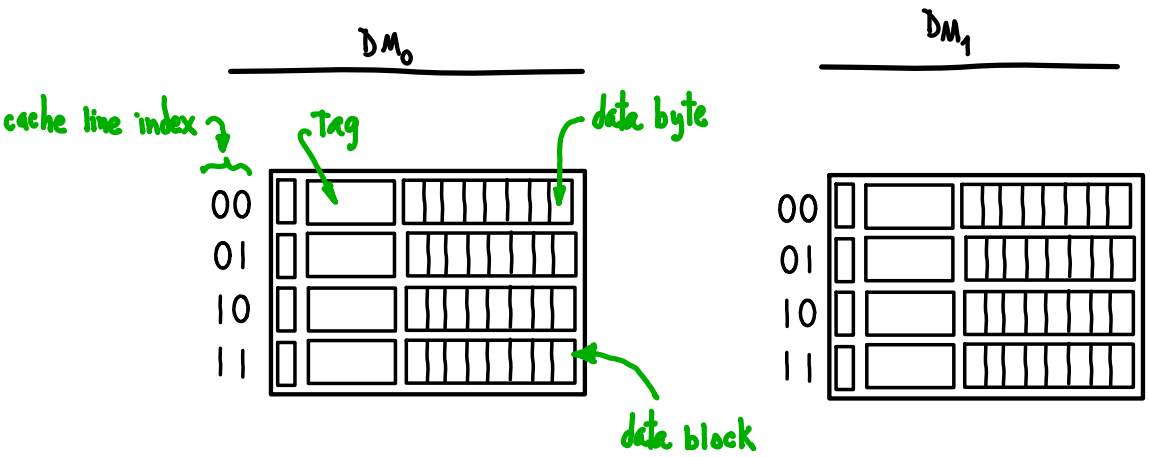


The same address bits can be thought looked at in this format: [**TAG**, **index**, **W#**, **B#**]:
TAG --- Address bits to identify the cache entry
index --- bits to specify which block or set of blocks in the cache
W# --- which word within the block
B# --- which byte within the word

Q. Again given the **PMAR**, indicate which bits correspond to each category above. Explain.

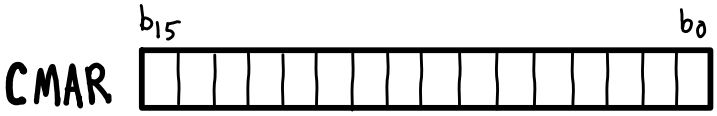


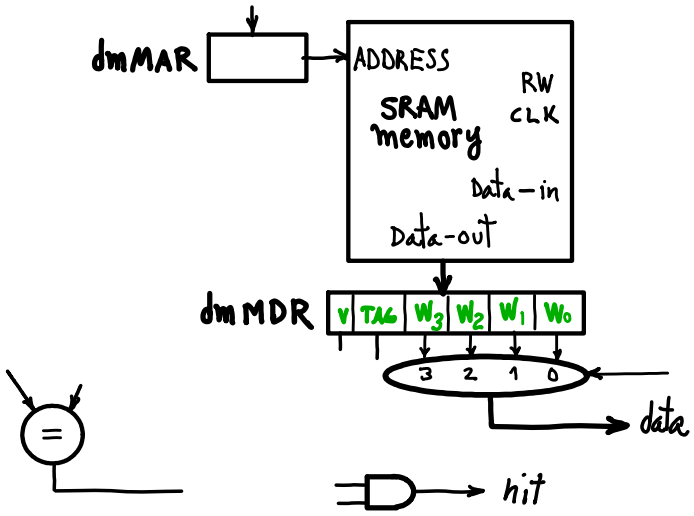
Q. The figure below shows the cache's two DMs. Indicate which blocks form each set and the index for each set.



Note that each DM is a small SRAM memory with its own address port, data ports, and write-enable as shown below. We can think of each DM memory as having an address register, **dmMAR**, and a data register, **dmMDR**. The diagram shows one of the two DMs.

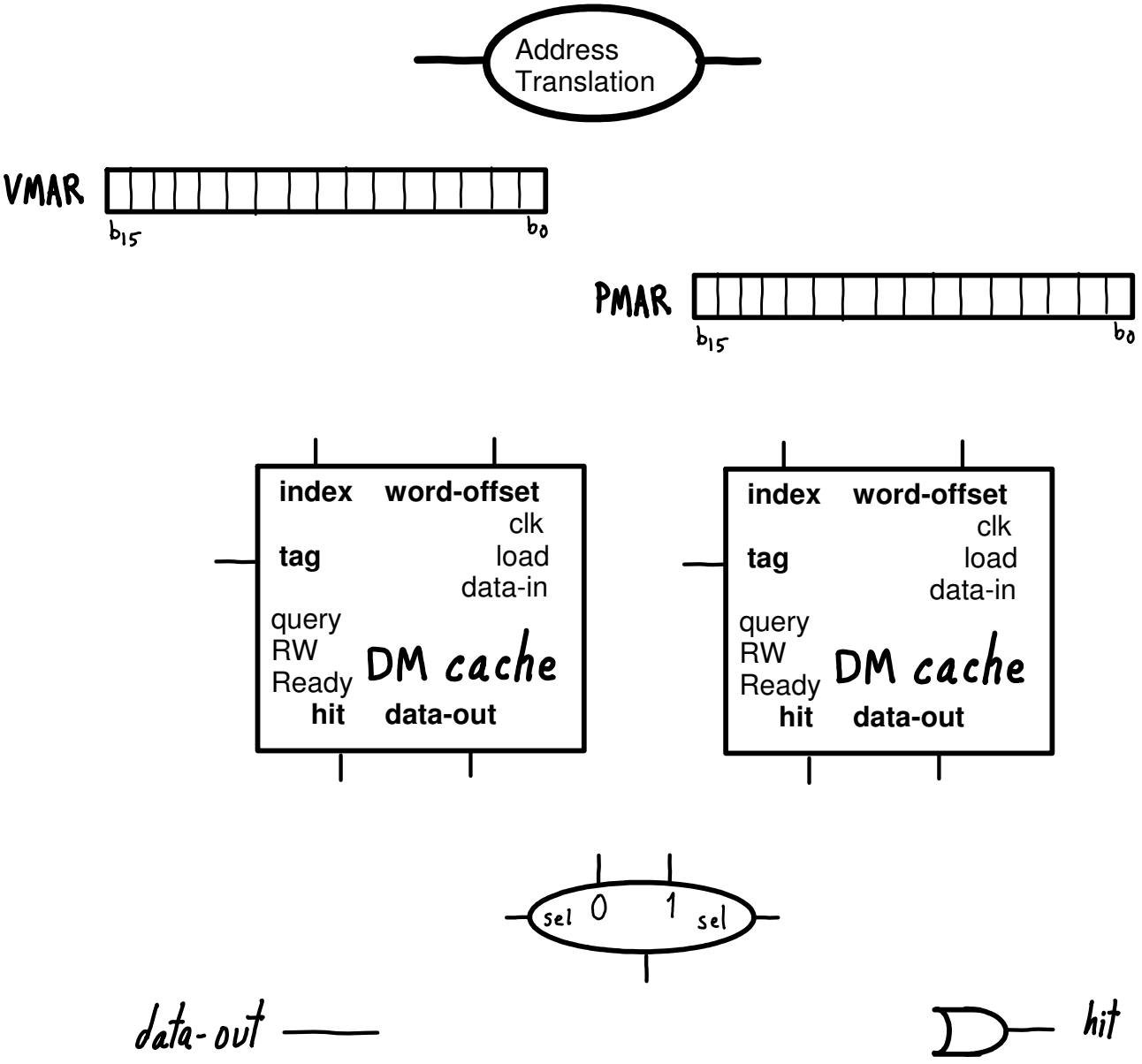
Q. Show the wire connections from the **CMAR** to the **dmMAR**, to the **MUX select**, and to the **comparator**. Show logic connects for the **hit** signal. Indicate the number of bits in each connection, in the **dmMDR,** and total number of **DM SRAM** bits.





Q. For the same system as above, the SRAM memories and DM logic is packaged as two individual DM caches. Show connections for virtual address translation, set indexing, tags, word-offset, hit signal logic, and data-out. Indicate which VMAR and PMAR bits are connected, and the number of bits for each connection. (Don't bother with other signals, e.g., **load**.)

The connections and logic are for a read operation. The MUX has two select lines. These are not the usual mux selects: if the left "sel" is 1, then input 0 is routed to the input, and similarly for the right "sel" and input 1. Don't worry about both selects being 1 at the same time.



Q. This cache is designed for a 1 cycle hit time. Delays are: translation = 20 ps, mux = 4 ps, DM caches = 25 ps, logic gate = 1 ps. What is the fastest CR possible?

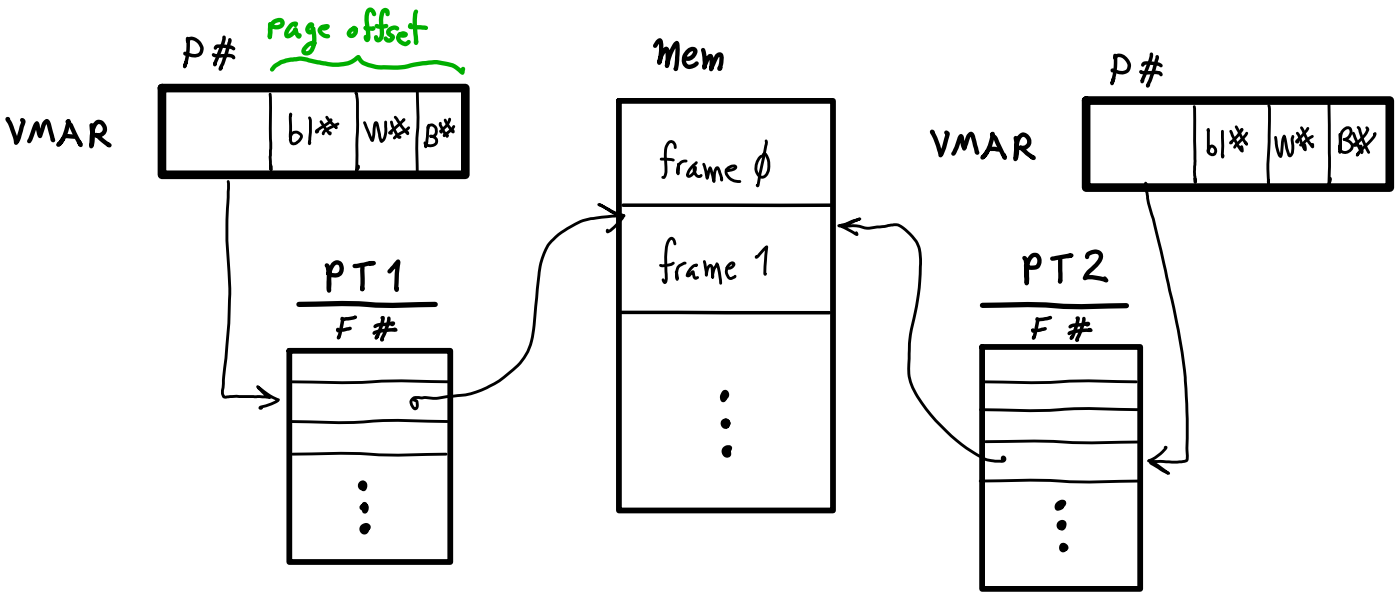
Below is shown two processes making virtual memory references (on the above system). Process 1 (on the left) is using page table **PT1**. **PT2** is Process 2's table. The page number of a virtual address is shown as **P#**, the physical frame number as **F#**. Memory addresses and page table indices start from 0 at the top.

Page offset bits can be separated into three offsets: a cache block number within the page (**bl#**), a word number within a block (**W#**), and a byte number within a word (**B#**).

Q. For the 16-bit virtual address, how many bits designate a page number, P#? How many entries in a page table?

Q. How many bits are frame numbers, F#? How big (in bytes) is a page table? Aside from the address translation information, these page table entries include 4 protection-bits, a valid-bit, a dirty-bit, and an accessed-bit. The size of a page table entry is rounded up to an integer number of bytes.

Q. Fill in P#s and F#s to match the indicated mappings.



The above scenario shows two processes sharing a page, but using different virtual mappings. Suppose our cache uses virtual addresses for tags and indexing instead of physical addresses. Also, cache entries contain a process id field, so the cache does not have to be flushed on a context switch. And, both process are executing concurrently. Both processes could write to the same page.

Q. Could the two process see different data for the same page? Explain how this could happen. (This is called the "synonym problem".)