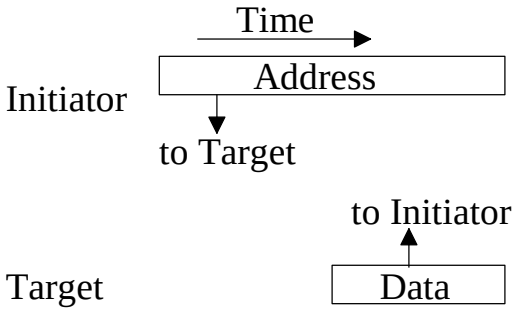
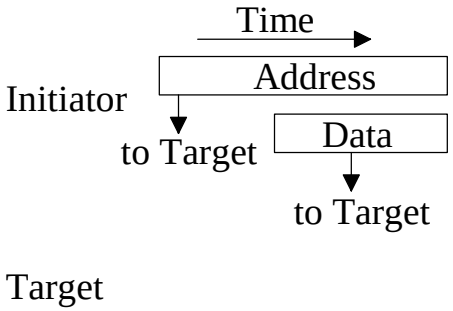
Computer Systems Test 2

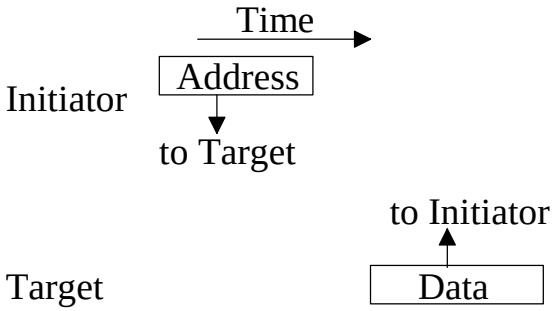
Question 1. (5 points) A READ operation on a bus with dedicated (non-multiplexed) address and data lines has a timing diagram of



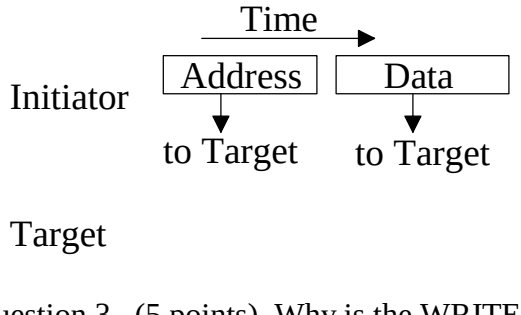
Draw the timing diagram of a WRITE operation on a bus with dedicated address and data lines.



Question 2. (5 points) A READ operation on a bus with time-multiplexed address and data lines has a timing diagram of

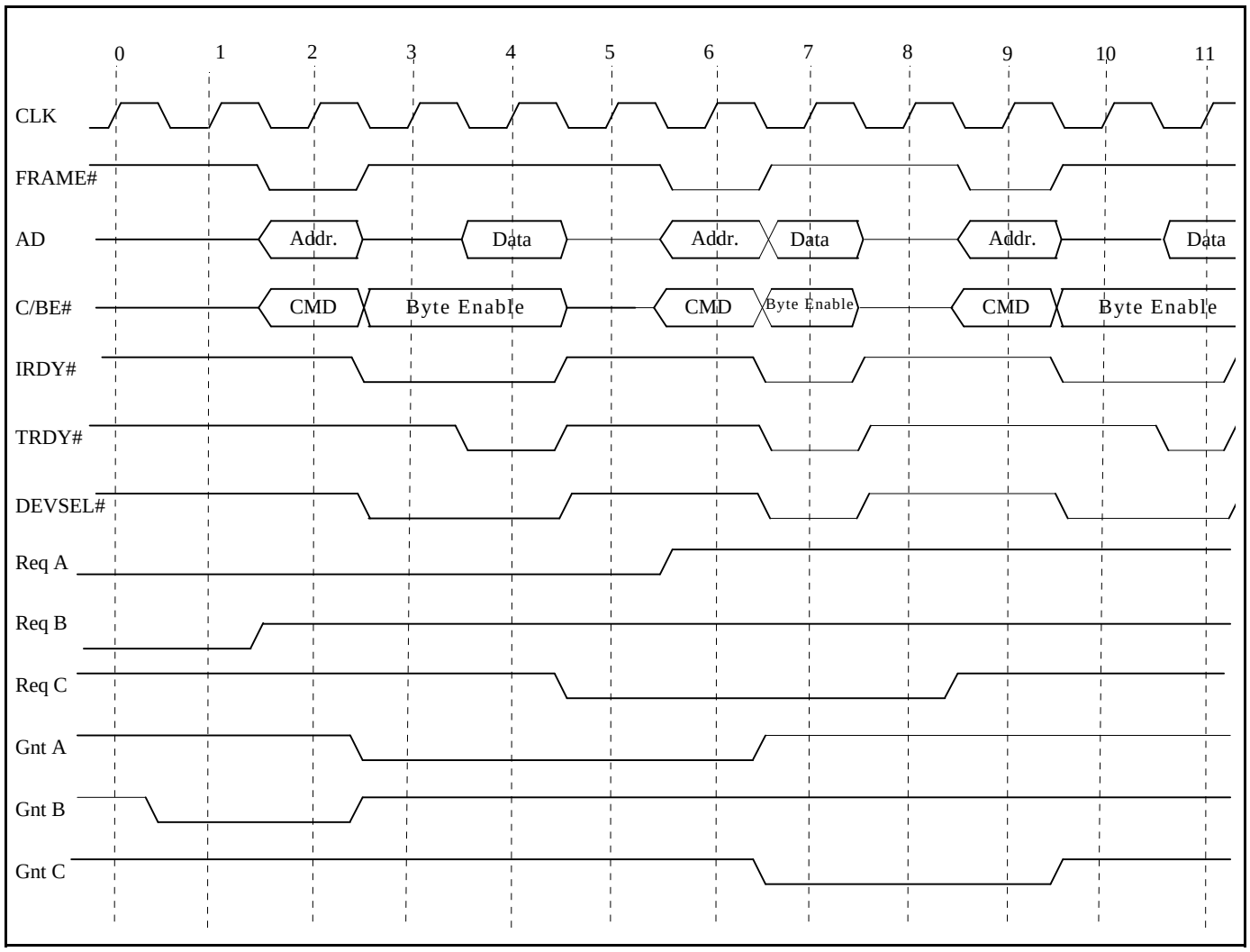


Draw the timing diagram of a WRITE operation on a bus with time-multiplexed address and data lines.



Question 3. (5 points) Why is the WRITE operation's timing lengthened more than the READ operation when using time-multiplexed address and data lines instead of dedicated address and data lines on a bus?
With dedicated lines the WRITE initiator can send the address and data at the same time, but not with time-multiplexed. On a READ the address must precede the data even with dedicated lines.

Question 4. (20 points) Consider the following PCI timing diagram.



a) At what clock cycles is data read off the AD wires? 4, 7, 11

b) Who controls (i.e., puts signals on) the "Req B" wire? Device B

c) Who controls (i.e., puts signals on) the "Gnt B" wire? the PCI arbitrator

d) For the second bus transaction, which device asserted the FRAME wire at clock cycle 5.5? Device A

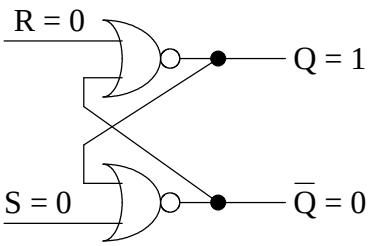
e) For the second bus transaction, how does the device know when it is time to assert the FRAME wire?

Device A has been granted the bus next since GNT A is asserted. When FRAME, DELSEL, IRDY, and TRDY are all deasserted, it knows it can grab the bus.

f) Use the above diagram to explain how the PCI protocol pipelines the arbitration for using the bus next with the current bus transfer.

As the first PCI bus transaction is occurring (cycles 1.5 to 4.5) arbitration for the second bus transaction by device A is occuring. Device A is notified that it can use the bus next at cycle 2.5 when GNT A is asserted. When device A sees that the bus is free at cycle 5, it grabs the bus immediately at cycle 5.5

Question 5. (10 points) The SR-latch pictured is remembering a "1". If R goes to a 1 value, what happens to the output on Q and $\overline{Q}$? (Justify your answer)



When R goes to 1, the top NOR gate starts outputting 0 so Q changes to 0. The Q of 0 is then fed as input to the bottom NOR gate so that it has two 0 inputs and thus $\overline{Q}$starts outputting a 1. Once this 1 if fed back to the upper NOR gate, the R input can go back to 0 and stable 0-state will be maintained.

Question 6. (30 points) We looked at the comparison between implementing large memories using (i) a register-file implementation, and (ii) a square-array-of-bits implementation with a row and column decoder.
a) Explain why the square-array-of-bits implementation scales better for large size memories.

A register-file implementation with an n-input address needed an n-to-2^n decoder, while the square-array-of-bits implementation needed only two (n/2)-to-$2^{n/2}$ decoders. Since the number of gates needed to implement a decoder grows exponentially with the number of number of address bits, the square-array-of-bits implementation requires much fewers gates. Additionally, using the tri-state buffers in the square-array-of-bits implementation allows us reuse the two decoders for the READ port and thus eliminated all of the MUXs used in the register-file implementation.

b) List one advantage of the register-file implementation over the square-array-of-bits implementation. (Explain why it is a necessary feature for a register file)

The register-file implementation allows a WRITE and multiple READ ports to operate simultaneously. This is necessary in a CPU since were need to send two operands off to the ALU and receive a result simultaneously.

c) Explain how the square-array-of-bits implementation aids advanced memories (such as the EDRAM and SDRAM) when supplying a "burst" of localized memory accesses in a reduced amount of time.

A row in the square-array represents a block of consecutive memory locations. Once a row is accessed from the slow DRAM (capacitor-based) memory, the row can be stored in faster SRAM and multiple accesses can be done quickly and without full specifying the complete address.

d) In the memory hierarchy (registers, L1 cache, L2 cache, main memory, ...), why is it important for the main memory to be able to supply a "burst" of localized memory accesses in a reduced amount of time?

The main memory is a bottleneck in the system so it is important to make it as fast as possible. When the processor has a cache miss, the cache needs a block of memory. Having a memory that can supply a fast burst reduces the cache miss time.

e) For a square-array-of-bits implementation with 2^{21} words, why would it be better to have a 10-bit row decoder and 11-bit column decoder instead of a 11-bit row decoder and 10-bit column decoder?

Each row would be longer and there would be fewer rows with the 10-bit row decoder. The advantages would be:

i) a burst from a row could be longer, and

ii) refreshing in done a row at a time, so having fewer rows means less time spent refreshing

Question 7. (20 points) Suppose we have 30-bit memory addresses, a byte-addressable memory, and a 256 KB (2^{18} bytes) cache with 16 (2^4) bytes per line.

a) How many total lines are in the cache? $2^{18}/2^4 = 2^{14}$

b) If the cache is direct-mapped, how many cache lines could a specific memory block be mapped to? one

c) If the cache is direct-mapped, what would be the format (tag bits, cache line bits, block offset bits) of the address? (Clearly indicate the number of bits in each)

12-bits	14-bits	4-bits
tag	line #	offset

d) If the cache is 8-way set associative, how many cache lines could a specific memory block be mapped to? 8

e) If the cache is 8-way set associative, how many sets would there be? $2^{14}/2^3 = 2^{11}$ sets

f) If the cache is 8-way set associative, what would be the format of the address?

15-bits	11-bits	4-bits
tag	set #	offset

g) Why would a fully-associative cache of the above size be impractical?

In a fully-associative cache, the block could get mapped to any cache line. The tag of the block being accessed by the CPU must be compared to all the line tags. This can be done in parallel by providing a hardware comparator for each tag line, but it is expensive. Plus, the results of these comparisons still needs to be searched for a match.

Question 8. (5 points) Why is a set-associative cache more flexible than a direct-mapped cache with respect to what is in the cache at the same time?

In a direct-mapped cache, each memory block only gets mapped to a single cache line. In a set-associative cache, memory blocks get mapped to any one of a small set of cache lines. If a code block accesses a data block that gets mapped to the same cache line, then in a direct-mapped cache these blocks are constantly swapped into and out of cache. In a set-associative cache these blocks would map to the same cache set, but since a set can hold several blocks both can be in the cache at the same time.