CASE WESTERN RESERVE UNIVERSITY
ECES 318
HW 3

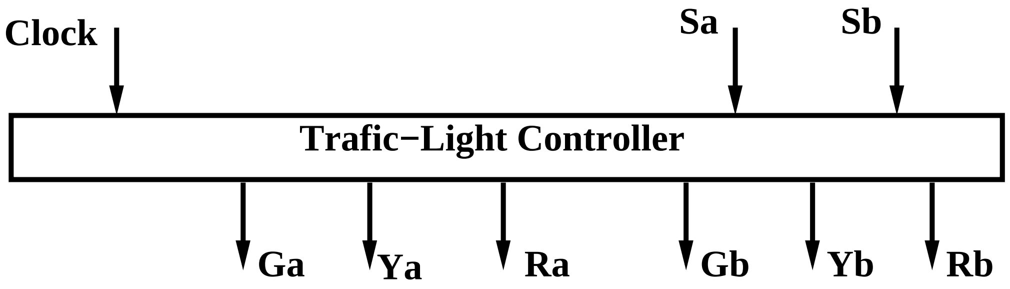
Fall 2014

Due Nov 18, 2014

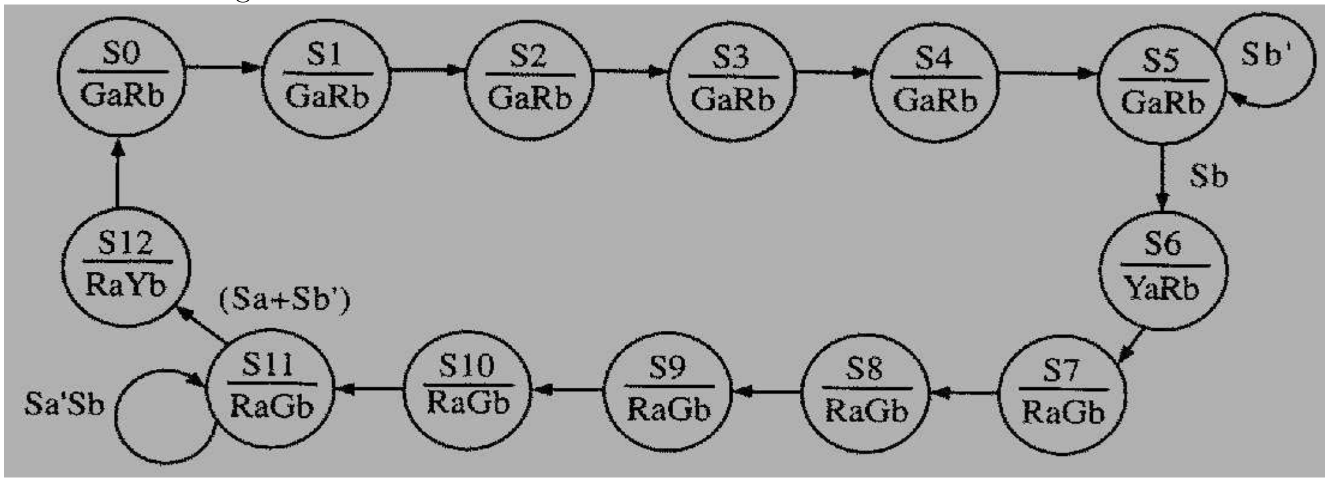
NAME:

Problem	Scale	Score
1	30	
2	60	
3	10	
4	30	
total	130	

Problem 1: The objective of this lab is to design a sequential traffic-light controller for the intersection of "A" street and "B" street. Each street has traffic sensors, which detect the presence of vehicles approaching or stopped at the intersection. Sa = 1 means a vehicle is approaching on "A" street, and Sb = 1 means a vehicle is approaching on "B" street. "A" street is a main street and has a green light until a car approaches on "B". Then the light changes, and "B" has a green light. At the end of 50 seconds, the lights change back unless there is a car on "B" street and none on "A", in which case the "B" cycle is extended 10 more seconds. When "A" is green, it remains green at least 60 seconds, and then the lights change only when a car approaches on "B". The figure below shows the external connections to the controller. Three of the outputs (Ga, Ya, and Ra) drive the green, yellow, and red lights on "A" street. The other three (Gb, Yb, and Rb) drive the corresponding lights on "B" street.



The figure below shows a Moore state graph for the controller. For timing purposes, the sequential network is driven by a clock with a 10-second period. Thus, a state change can occur at most once every 10 seconds. The following notation is used: GaRb in a state means that Ga = Rb = 1 and all the other output variables are 0. Sa'Sb on an arc implies that Sa = 0 and Sb = 1 will cause a transition along that arc. An arc without a label implies that a state transition will occur when the clock occurs, independent of the input variables. Thus, the green "A" light will stay on for 6 clock cycles (60 seconds) and then change to yellow if a car is waiting on "B" street.



Write a VHDL model for the trafic light controller. Simulate the file in ModelSim to verify correctness.

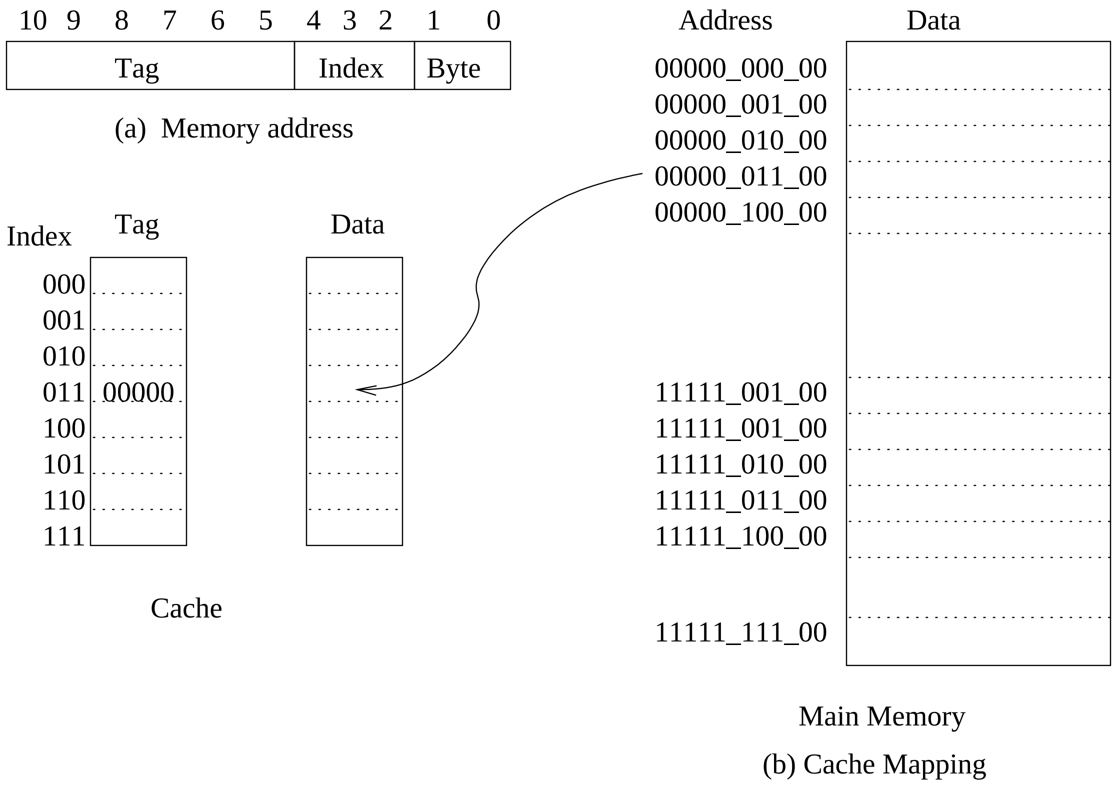


Figure 1: Direct Mapped Cache

1 Typical Cache Memory

To illustrate the concept of cache memory, we assume a very small cache of eight 32-bit words and a small main memory with 1 KB (256 words), as shown in Figure 1. Both of these are too small to be realistic, but their size makes illustration of the concepts easier. The cache address contains 3 bits, the memory address 10. Out of the 256 words in main memory, only 8 at a time may lie in the cache.

In order for the CPU to address a word in the cache, there must be information in the cache to identify the address of the word in main memory. If we consider the example of a for loop, clearly, we find it desirable to contain the entire loop within the cache, so that all of the instructions can be fetched from the cache while the program is executing most of the passes through the loop. The instructions in the loop lie in consecutive word addresses. Thus, it is desirable for the cache to have words from consecutive addresses in main memory present simultaneously. A simple way to facilitate this feature is to make bits 2 through 4 of the main memory address be the cache address. We refer to these bits as the index, as shown in Figure 1. Note that the data from address 0000001100 in main memory must be stored in cache address 011. The upper 5 bits of the main memory address, called the tag, are stored in the cache along with the data. Continuing the example, we find that for main memory address 0000001100, the tag is 00000. The tag combined with the index (or cache address) and byte field identify an address in main memory.

Suppose that the CPU is to fetch an instruction from location 000001100 in main memory. This instruction may actually come from either the cache or main memory. The cache separates the tag 00000 from the cache address 011, internally fetches the tag and the stored word from location 011 in the cache memory, and compares the tag fetched with the tag

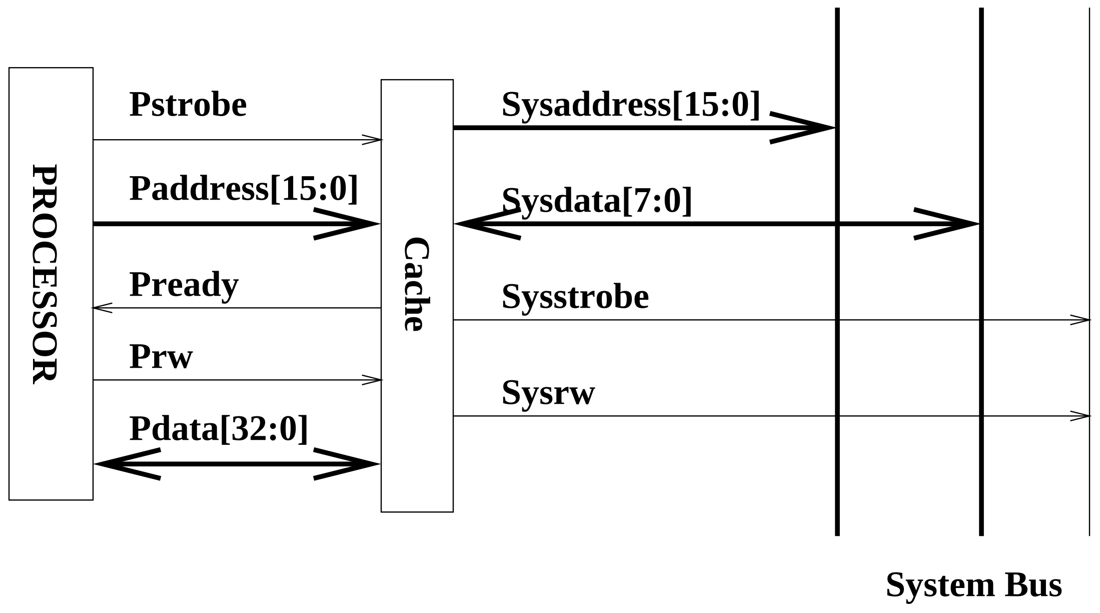


Figure 2: System Interface

portion of the address from the CPU. If the tag fetched is 00000, then the tags match, and the stored word fetched from cache memory is the desired instruction. Thus, the cache control places this word on the bus to the CPU, completing the fetch operation. This case in which the memory word is fetched from cache is called a cache hit. If the tag fetched from cache memory is not 00000, then there is a tag mismatch, and the cache control notifies main memory that it must provide the memory word, which is not available in the cache. This situation is called a cache miss. For a cache to be effective, the slower fetches from main memory must be avoided as much as possible, making considerably more cache hits than cache misses necessary.
When a cache miss occurs on a fetch, the word from main memory is not placed just on the bus for the CPU. The cache also captures the word and its tag and stores them for future access. In our example, the tag 00000 and the word from memory will be written in cache location 011 in anticipation of future accesses to the same memory address.

2 Interfaces

A cache system typically lies between the processor and the main system bus. The following block diagram Figure 2 describes the signals and buses that the cache needs to communicate with the processor and the system. Note that in our system the system bus is synonymous with main memory.

2.1 Processor Interface

The processor interface consists of the processor address bus, Paddresst[15:0], the processor data bus, Pdata[32:0], and control signals Pstrobe and Pready. The Pstrobe is asserted when the processor is starting a bus transaction and a valid address is on the Paddress bus. Pready is used to signal to the processor that the bus transaction is completed. The timing

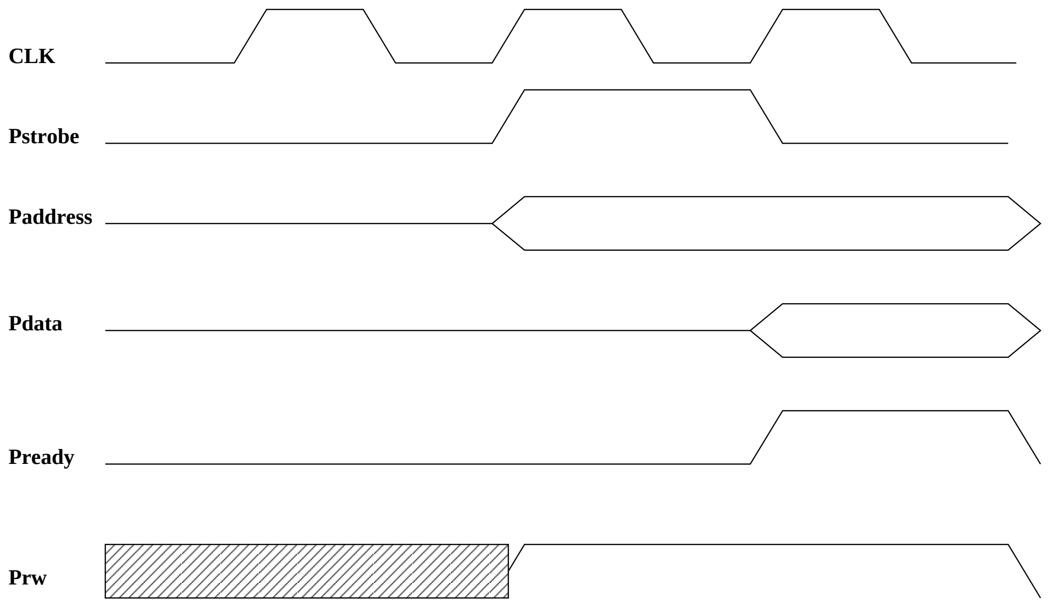


Figure 3: Processor Interface

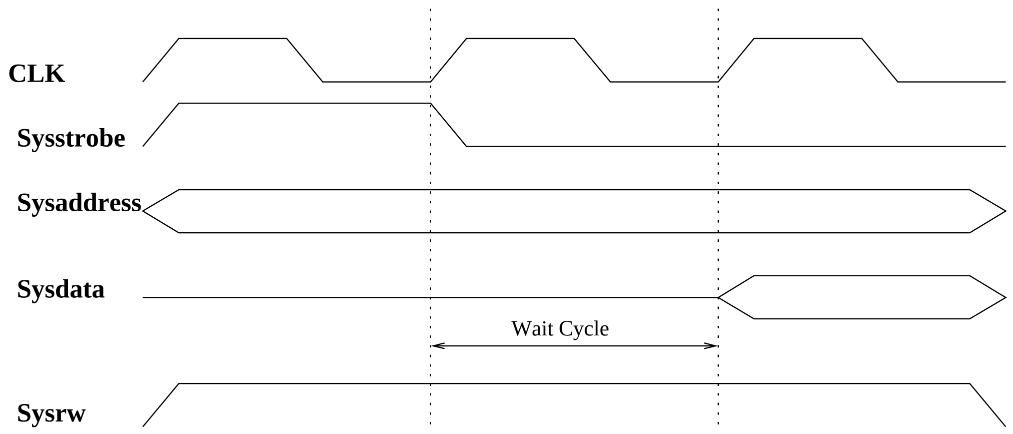


Figure 4: System Interface

diagram in Figure 3 demonstrates a simple read cycle. The Prw signal is high for a read and low for a write.

2.2 System Bus Interface

For our cache model we assume a simple bus model. For a read operation, the Sysaddress is first presented to the bus along with the Sysstrobe signal and the Sysrw. The Sysrw signal is high for read operations and low for write operations. After a set number of wait states (in our case it is four clock cycles), the data is returned. A write operation is similar, but the data is driven onto the Pdata bus immediately and then waits for the set number of wait states (in our case it is four clock cycles) before issuing another write operation. Figure 4 shows a system read with one wait state.

2.3 Cache Architecture

The direct-mapped cache is the simplest of all cache architectures. A direct mapped cache consists of a single tag RAM, cache RAM, and a simple controller. You have to model each of these parts separately and then bring them together in the final model.
Assume that the processor has a 16-bit address composed of the following fields: The byte field (bits [1:0]), the index field (bits [9:2]), and the tag filed (bits [10:15]).
To simplify the cache, all writes from the processor update both the cache and main memory. This is called write-through mode operation. In write-through mode, the cache memory is always kept coherent with main memory.
Assume the following that a bus should be tristated at the end of a bus transaction, memory write and cache write can be performed in parallel, a signal assignment incurs one unit delay, and the clock period is 100 unit.

Problem 2: Write a VHDL model for the above system.

Problem 3: To test your system, simulate a processor request of the following sequence of Hex addresses: '76','66','76','59'.

Problem 4:

Figure 1. shows an example of the long division of unsigned binary integers. First, the bits of the dividend are examined from left to right, until the set of bits examined represents a number greater than or equal to the divisor: this is referred to as the divisor being able to divide the number. Until this event occurs, Os are placed in the quotient from left to right. When the event occurs, a 1 is placed in the quotient and the divisor is subtracted from the partial dividend. The result is referred to as a partial remainder. From this point on, the division follows a cyclic pattern. At each cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor. As before, the divisor is subtracted from this number to produce a new partial remainder. The process continues until all the bits of the dividend are exhausted.

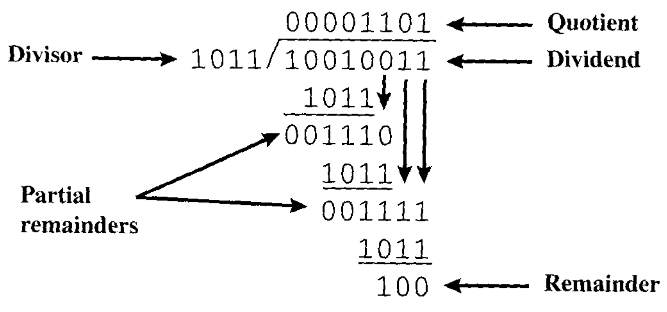


Figure 1. Example of division of unsigned binary integers.

Figure 2. shows a machine algorithm that corresponds to the long division process. The divisor is placed in the M register, the dividend in the Q register. At each step, the A and Q registers together are shifted to the left 1 bit. M is subtracted from A to determine whether A divides the partial remainder.3 If it does, then Q0 gets a 1 bit. Otherwise, Q0 gets a 0 bit and M must be added back to A to restore the previous value. The count is then decremented, and the process continues for n steps. At the end, the quotient is in the Q register and the remainder is in the A register.

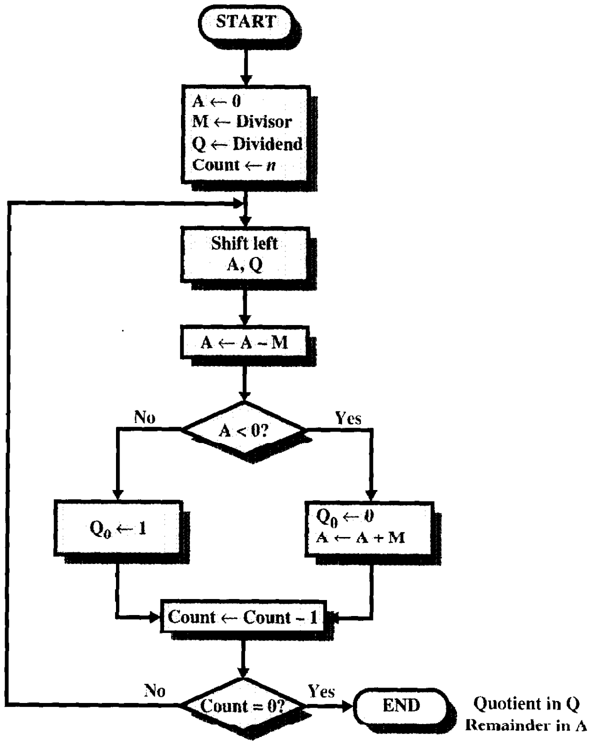


Figure 2. Flowchart of division of unsigned binary integers.

We give here one approach to extend the division algorithm to handle negative twos complement numbers. Two examples of this approach are shown in Figure 3. The algorithm can be summarized as follows:

1. Load the divisor into the M register and the dividend into the A, Q registers The dividend must be expressed as a 2n-bit twos complement number Thus for example, the 4-bit 0111 becomes 00000111, and 1001 becomes 11111001.

2. Shift A, Q left 1 bit position.

3. If M and A have the same signs, perform $A = A - M$; otherwise, $A = A + M$.

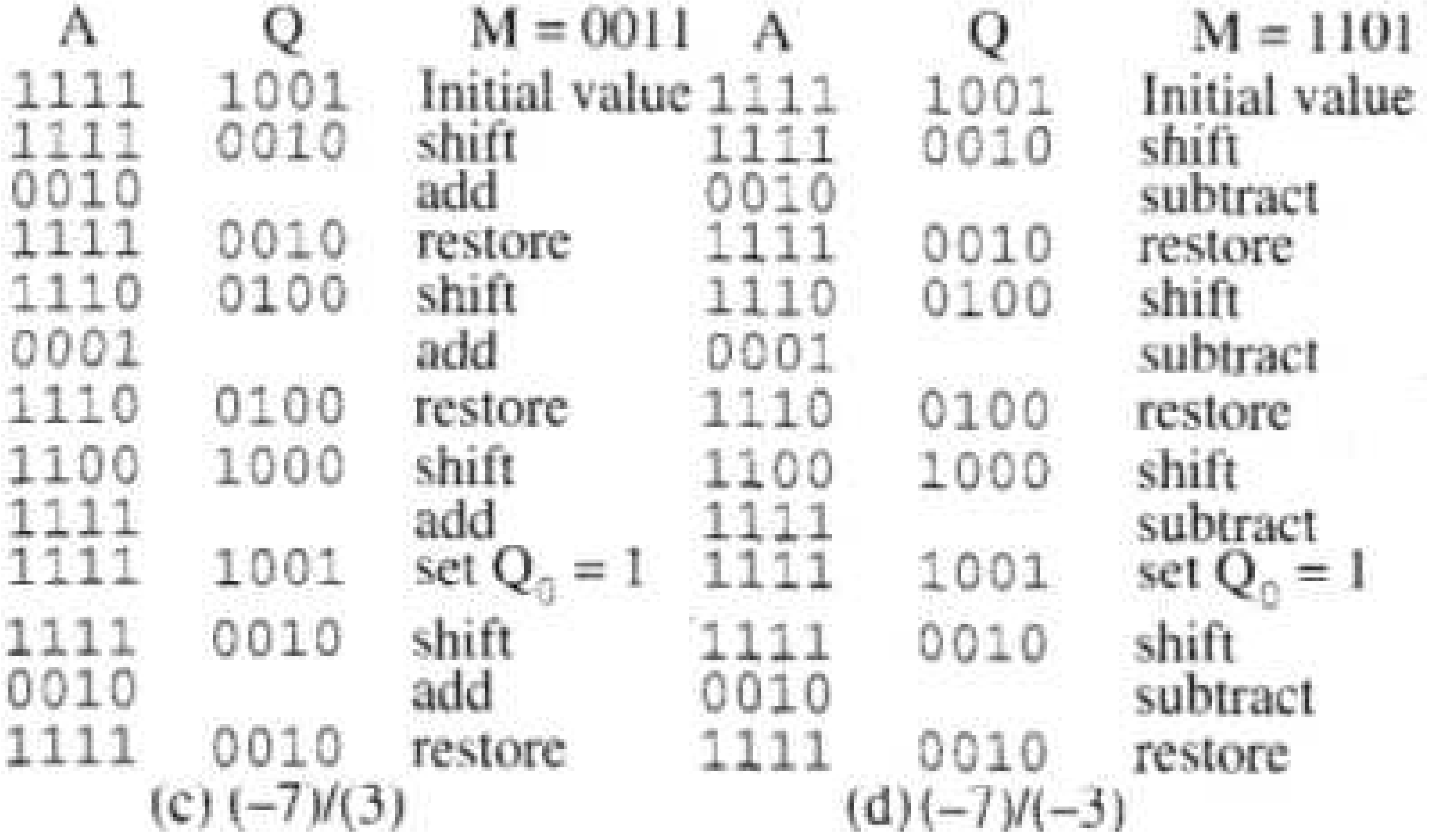
4. The preceding operation is successful if the sign of A is the same before and after the operation.

   a. If the operation is successful or $A = 0$, then set $Q_0 = 1$.

   b. If the operation is unsuccessful and $A \neq 0$, then set $Q_0 = 0$ and restore the previous value of A.

5. Repeat steps 2 through 4 as many times as there are bit positions in Q.

6. The remainder is in A. If the signs of the divisor and dividend were the same, then the quotient is in Q; otherwise, the correct quotient is the twos complement of Q.



A	Q	M = 0011	A	Q	M = 1101
1111	1001	Initial value	1111	1001	Initial value
1111	0010	shift	1111	0010	shift
0010		add	0010		subtract
1111	0010	restore	1111	0010	restore
1110	0100	shift	1110	0100	shift
0001		add	0001		subtract
1110	0100	restore	1110	0100	restore
1100	1000	shift	1100	1000	shift
1111		add	1111		subtract
1111	1001	set $Q_0 = 1$	1111	1001	set $Q_0 = 1$
1111	0010	shift	1111	0010	shift
0010		add	0010		subtract
1111	0010	restore	1111	0010	restore

(c) (−7)/(3) (d) (−7)/(−3)

Figure 3. Example of twos complement division.

a) Write a VHDL file to model the above behavior for 4 bits Divisor and 4 bits divident.
b) Use Modelsim to compile your VHDL file.
c) Use Modelsim to simulate your division circuit with the following cases: 7/-2, 6/-2.