CSCI 2400: Computer Systems
Recitation Exercise 5
(Dated: December 1, 2014)

6.30 ◆◆

Suppose we have a system with the following properties:

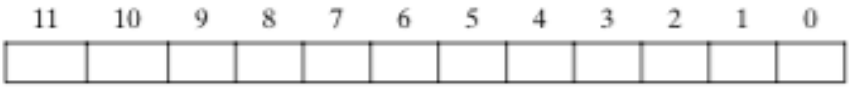
- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not to 4-byte words).
- Addresses are 12 bits wide.
- The cache is two-way set associative ($E = 2$), with a 4-byte block size ($B = 4$) and four sets ($S = 4$).

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecimal notation:

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	—	—	—	—
2	00	1	48	49	4A	4B
	40	0	—	—	—	—
3	FF	1	9A	C0	03	FF
	00	0	—	—	—	—

A. The following diagram shows the format of an address (one bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO	The cache block offset
CI	The cache set index
CT	The cache tag



B. For each of the following memory accesses indicate if it will be a cache hit or miss when **carried out in sequence** as listed. Also give the value of a read if it can be inferred from the information in the cache.

Operation	Address	Hit?	Read value (or unknown)
Read	0x834	_____	_____
Write	0x836	_____	_____
Read	0xFFD	_____	_____

8.23 ◆◆

One of your colleagues is thinking of using signals to allow a parent process to count events that occur in a child process. The idea is to notify the parent each time an event occurs by sending it a signal, and letting the parent's signal handler increment a global `counter` variable, which the parent can then inspect after the child has terminated. However, when he runs the test program in Figure 8.41 on his system, he discovers that when the parent calls `printf`, `counter` always has a value of 2, even though the child has sent five signals to the parent. Perplexed, he comes to you for help. Can you explain the bug?

code/ecf/counterprob.c

```
1   #include "csapp.h"
2
3   int counter = 0;
4
5   void handler(int sig)
6   {
7       counter++;
8       sleep(1); /* Do some work in the handler */
9       return;
10  }
11
12  int main()
13  {
14      int i;
15
16      Signal(SIGUSR2, handler);
17
18      if (Fork() == 0) {  /* Child */
19          for (i = 0; i < 5; i++) {
20              Kill(getppid(), SIGUSR2);
21              printf("sent SIGUSR2 to parent\n");
22          }
23          exit(0);
24      }
25
26      Wait(NULL);
27      printf("counter=%d\n", counter);
28      exit(0);
29  }
```

code/ecf/counterprob.c

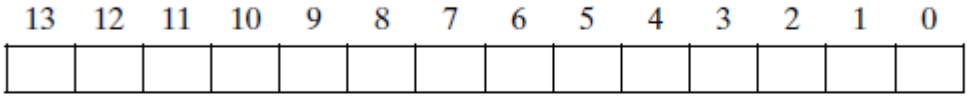
Figure 8.41 **Counter program referenced in Problem 8.23.**

9.11 ◆

In the following series of problems, you are to show how the example memory system in Section 9.6.4 translates a virtual address into a physical address and accesses the cache. For the given virtual address, indicate the TLB entry accessed, the physical address, and the cache byte value returned. Indicate whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs. If there is a cache miss, enter "–" for "Cache Byte returned." If there is a page fault, enter "–" for "PPN" and leave parts C and D blank.

Virtual address: 0x027c

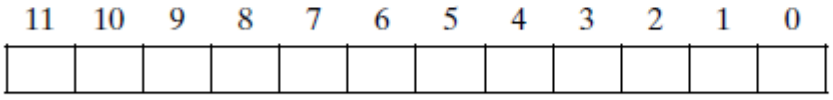
A. Virtual address format



B. Address translation

Parameter	Value
VPN	_____
TLB index	_____
TLB tag	_____
TLB hit? (Y/N)	_____
Page fault? (Y/N)	_____
PPN	_____

C. Physical address format



D. Physical memory reference

Parameter	Value
Byte offset	_____
Cache index	_____
Cache tag	_____
Cache hit? (Y/N)	_____
Cache byte returned	_____