1. What advantage is there to using a DMA controller (as part of an I/O device controller) in moving blocks of information between memory and I/O devices?

2. Using a branch delay slot would now be considered obsolete. Branch prediction is the preferred method of dealing with the problem. How can branch prediction be implemented in ID? in IF?

3. It often happens that a program will block on `read()` since the required input information may not yet be available. Assuming we have the output information ready in our user buffer, under what circumstances could a program still block on `write()`?

4. In a typical page table entry there is a physical page number, a “present (or in-core)” bit, a “modified” bit, a “lock-in-core” bit, a “volatile” bit and a “COW” bit. What is each of these used for?

5. A process running on a virtual memory system is currently using \( n \) actual hardware pages. It makes a `fork()` system call. Will the resulting number of hardware pages in use by both the parent and offspring now be \( 2n \)? Why or why not?

6. On executing an instruction to read a user variable (int `status`;) from the data segment into a CPU register a page fault occurs. However, the data page that the variable `status` resides in is valid in core. What exactly happened to cause the page fault?

7. What are some of the complications involved in going from one CPU to two CPU’s in a computer system?

8. Data structures using pointers are often more elegant than those which do not use pointers. What are the efficiency concerns with regard to L1 and L2-caches, and TLB caches when making extensive use of pointers in programs? Be specific.

9. A typical I/O device controller has a small number of command/status ports and several internal memory buffers. What are each of these used for? How does the controller inform the CPU that it has completed a command?

10. Fully associative caches are more flexible than set-associative caches. Why is it then that fully associative caches are used so seldom and, when they are used, are rather small (by comparison with L2 and L3-caches)?

11. Consider the following program fragment:

```c
int flag = 1;
void *task(void *arg) {
    int k;
    static int m;
    ...
    return((void *)&mode); }
```

11a. In which segment (code, data, stack) will each of the variables \( k, m, \text{flag} \) be stored and what scope (local, global) will the name of the variable have?

11b. if `task()` is run as an auxiliary thread, will it need its own stack (separate from `main()`’s stack)? Why or why not?

12. For each of the numbered items tell whether the use of virtual memory will accomplish what is claimed or not, and why:

12a. Allow more programs to be simultaneously run using the same physical memory size (assuming a healthy job mix).

12b. Allow programs which access very large data sets through linked lists of pointers to run faster.

12c. Allow operating systems to manage memory is a more straightforward fashion.

12d. Make it easier for programmers writing programs which do lots of dynamic memory allocation.