Module 12 Example Set 1

1. How does a process differ from a thread?
A program that has been loaded into memory and is able to execute is referred to as a process. Each process has an address space determined by the width of the PC register (e.g., a 32-bit PC corresponds to a 4GB address space). The address spaces of different processes are separate and distinct. One process cannot access items within the address space of another process. A process contains one or more threads of execution. Each thread within a process has a separate context which defines the state of the thread. The threads within a process share a common address space and can communicate via the shared address space while separate processes can only communicate via files or with the assistance of the operating system. Threads are sometimes referred to as "lightweight processes".

2. What defines the context or state of a thread?
The state of a thread is primarily composed of the value in the program counter (PC), the contents of the general purpose CPU registers and the contents of special purpose registers such as memory management registers or program status registers. A thread that has been previously stopped or suspended can later be resumed with no adverse effects if its state (i.e. context) is saved and restored properly.

3. How is multi-threading handled on a conventional uniprocessor system?
A conventional processor only executes instructions from one thread at a time. If the active thread has to stall or wait for an event such as a page fault or I/O request to be processed, the operating system will save the context (all of the registers associated with the thread) in memory (for example, on a stack) and will fill the CPU registers with the information (i.e. the state) of another thread that the OS selects to be executed next. The software that performs a "context switch" can take thousands of CPU cycles.

4. What is meant by a "multi-threaded architecture"?
A multi-threaded architecture is one in which a single processor has the ability to execute multiple streams of instructions without the need for software supported context switches. The CPU register set is replicated to provide the abililty to quickly (in just a few cycles) switch from one hardware context to another without software intervention. A separate set of registers is available for each thread.

5. How do superscalar and VLIW systems differ from multi-threaded processors?
Superscalar and VLIW processors are single-threaded because they execute programs using a single PC (program counter) and a single register file. Multi-threaded processors, on the other hand, have multiple program counters and register files each of which can be used to fetch and execute instructions from different threads at a time. Superscalar and VLIW systems issue multiple instructions in a single cycle, but these instructions all come from the same thread. Multi-threaded processors can execute instructions from multiple threads in the same cycle. The available execution units can be shared between threads in a multi-threaded system.

6. What are the different styles or types of multi-threading?
There are three major types:

I. Coarse-grained – in which a processor runs a single thread until some long latency event such as a cache miss triggers a switch to a different thread. The instructions from the stalled thread are drained from the pipeline and instructions from the newly activated thread are fed into the pipeline.

II. Fine-grained – the processor is switched from one active (i.e. not stalled) thread to another on every clock cycle (the instructions from different threads are interleaved), but only instructions from one thread are issued during any particular cycle.

III. Simultaneous multi-threading – in which instructions from multiple threads are issued during the same cycle to any available execution units that they require within a superscalar processor.

7. How can instructions from different threads be handled within a single pipeline without causing data dependencies or interferring with each other?
Multiple separate register sets (one for each thread) have to be used so that instructions from one thread do not overwrite the registers used by other threads.

8. With a multi-processor system, independent threads executing in parallel must run on separate processors. Assume a parallel program containing three threads (thread1, thread2 and thread3) is to be executed on a multi-processor system that contains 2 processors. Thread 1 takes T time units, thread 2 takes 2T time units and thread 3 takes 3T time units.

a) How many time units are required for the entire program if threads 1 and 3 are executed on the two processors, followed by executing thread 2 on one of the processors?
Executing threads 1 and 3 in parallel would take 3T time units (the time required for the longer of the two threads). Then thread 2 would take 2T time units; so the total would be 3T + 2T = 5T time units.

b) How many time units would be required for the entire program if ony one of the processors was available?
The three threads would have to execute sequentially one after the other, so the total time would be 1T + 2T + 3T = 6T time units.

c) What would be the best order in which to execute the three independent threads?
This would correspond to the shortest total time which is achieved by executing threads 2 and 3 in parallel. Thread 2 would complete after 2T time units. The processor that ran thread 2 could then be used to execute thread 1 in parallel with the completion of thread 3. Hence after 3T time units all three threads would be completed. After the first 2T time units, thread 2 would finish and thread 3 would complete its first 2T time units. In the next T time units threads 3 and 1 would complete.

9. What is meant by "barrier synchronization"?
Separate processes or threads when colaborating on a particular problem may have to synchronize their activity at strategic points. One way to do this is to have each process or thread execute up to a certain point within the code and wait for all of the others to reach the synchronization point after which they all proceed. This technique is referred to as barrier synchronization.

10. How might sharing a common address space cause dependencies between threads?
A result generated and stored into memory by one thread may be required by one or more other threads as an input. Often semaphores or mutexes must be used to prevent threads from accessing a shared memory operand at the same time. Semaphores or mutexes can be thought of as flags that can be set or cleared to indicate whether a shared item is in use or is free.

11. According to Amdahl's law as it applies to multi-processor systems, what effect does increasing the number of processors have on the execution time for a fixed size program consisting of a sequential part and a parallel part that can be split among multiple processors?
The sequential part would execute on a single processor and only the parallel part of the program would benefit from the use of the additional processors. Hence as the number of processors is increased, the parallel part would take a smaller and smaller fraction of the total execution time.