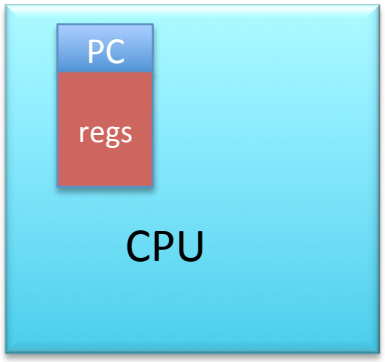
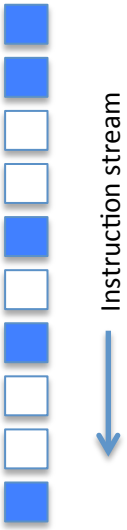
Example Set 5

1. How is multi-threading handled on a conventional processor?

A conventional processor is one with a single register set that is owned or used by one thread at a time. It contains a single PC (program counter) that is used to fetch instructions from a single stream at a time.



Conventional processor

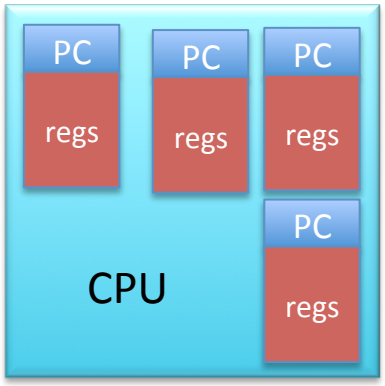


With superscalar processors the lack of ILP (instruction level parallelism) can cause some execution units to be underused.

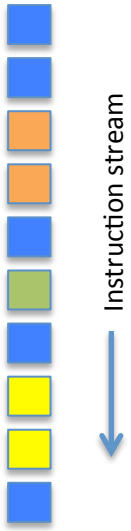
Only one thread can be executing at a time. When the active thread becomes blocked due to, say, a page fault or due to having to wait for an I/O operation to complete, the thread's "context" (i.e., the contents of its registers, its PC, status register, etc.) must be saved to memory and the context of another thread that is ready and waiting to execute must be copied from memory into the CPU's registers.
These context switches usually involve executing operating system code and can take thousands of CPU cycles. If there are no ready threads available, the CPU will idle.

2. How does a multi-threaded processor differ from a conventional processor?

A multi-threaded processor is able to follow multiple streams of execution without the need for software context switches. It employs hardware multi-threading as opposed to software multi-threading.



Multi-threaded processor



Multi-threaded processors can compensate for the lack of ILP by allowing execution resources not used by one thread to be used by another thread. That is, they employ TLP (thread level parallelism).

The multi-threaded processor contains multiple PC's, register sets, etc. which constitute a hardware context. The number of hardware contexts defines the number of threads that can be supported without software intervention.

Since the hardware contexts reside within or near the processor, a context switch can be performed in just a few cycles by simply switching to a different set of registers (including a different PC, status register, etc.).

However the threads share the same memory, branch predictor, cache and TLBs.

3. Often moving companies try to lower cost and increase profits by combining multiple loads within a single moving van rather than sending a partially filled van cross country. How is this analogous to a multi-threaded processor?

A single multi-threaded processor increases throughput by supporting in hardware a number of threads that run concurrently so as to take greater advantage of the available execution resources and to avoid wasting CPU cycles when the CPU becomes idle or when extra cycles are used for software supported context switching. This is analogous to avoiding the waste of unused space in the moving van.

4. Can it be said that hardware multi-threading "virtualizes" the processor?

Yes, hardware multi-threading makes a single-core processor appear to software as a multi-processor. Each thread running on a separate hardware context appears to be running on a separate virtual core that has the hardware capabilities of the physical CPU minus the resources used by the other threads.

5. As processor performance and speed continue to increase, why has hardware multi-threading become more important?

The difference between the time required for internal CPU operations and the time required to access memory on earlier slower processors was not as dramatic as it is for more modern processors. The use of cache memory helped to decrease this difference. However for the much faster modern processors, even a simple cache miss (not to mention waiting for I/O) can incur many hundreds or thousands of cycles as a penalty (due to the need to access the next level in the memory hierarchy to resolve the cache miss). The ability to quickly switch from one thread to another hides these penalties or latencies by allowing other threads to use the idle resources.

6. What are the main models of hardware multi-threading?

Three of the main models are course-grain multi-threading (CMT), fine-grained multi-threading (FMT), and simultaneous multi-threading (SMT).

Course-grained multi-threaded processors execute one thread at a time, but employ hardware contexts to switch from one thread to another in just a few cycles to hide long latencies (i.e., delays) such as memory accesses.

Fine-grained multi-threaded processors can context switch every cycle with no delay. They interleave instructions from different threads on a cycle by cycle basis. However, during any given cycle, they only issue instructions from a single thread.

Simultaneous multi-threaded processors can issue instructions from multiple threads in the same cycle. This allows the issue width (i.e., the number of available execution units) to be more fully used. The issue width is also known as the superscalar degree. This takes advantage of the issue width even when one thread does not have enough ILP to fully use all of the available execution units. Units that are not required by instructions from one thread may be needed by instructions from another thread.

7. List some of the possible sources of latencies or delays that can be experienced by modern processors.
The sources of latency include data cache misses, memory latencies, instruction cache misses, instruction dependencies, branch miss-predictions, TLB misses, and communication delays between processors.