SOLUTION NOTES

Comparative Architectures 2003 Paper 7 Question 1 (IAP)

(*a*) Compare and contrast each of the following techniques for achieving instruction-level parallelism:

- statically-scheduled super scalar
- out-of-order speculative execution
- Very Long Instruction Word (VLIW)
- EPIC (as used by IA-64)

[12 marks]

Statically-scheduled super scalar processors dispatch instructions to multiple pipelines in a simple in-order issue fashion. A block of instructions are fetched, and then scoreboarding and structural hazard checks are made before dispatching instructions that can execute in parallel, and holding back later instructions that can't. When all instructions in the block have been dispatched, the processor moves onto the next block.

Such processors typically achieve relatively low ILP because they are forced to completely stall at the first hurdle (e.g. instruction using a src operand that hasn't arrived from the cache yet). Out-of-order speculative execution attempts to avoid these stalls by searching ahead for non-dependent instructions to execute. This speculative execution can proceed beyond branch instructions thanks to branch prediction. Because execution beyond the original stalled instruction is speculative, the processor must be capable of backing out execution to any point in case anything is found to have gone wrong (mispredict, exception etc). Processor state is committed in an in-order fashion by the retire unit, freeing up resources to allow further speculation.

VLIW is an extreme form of statically-scheduled super scalar, whereby more of the burden for interlocking and pipeline slotting is placed on the compiler (or human assembler) rather than being implemented in hardware. A wide instruction word is exposed which contains multiple sub components corresponding to each of the processor's pipelines (each of which may be specialized in some way). By removing these h/w checks the architects hope the chip can be clocked faster.

"EPIC" is an attempt to provide some of the benefits of VLIW, without loss of binary compatibility between different generations when numbers of pipelines or their restrictions

change. Each 128 bit bundle contains 3 instructions, but there are rules about what class of instructions can appear in which position in the bundle (load, int, fp, branch etc). A 'stop bit' is used to group instructions into sets that are guaranteed independent and hence can be issued in parallel. (the groups may cross bundle boundaries).

(*b*) Discuss multi-threading, and hence the different implementation approaches that have been tried to enable a single CPU core to execute from multiple instruction streams. How can multi-threading be used to improve system performance in some usage scenarios? What are the pitfalls? [8 marks]

Even for an out-of-order speculative execution engine, some code sequences will yield poor ILP – the processor stalls because it can find no further work to perform. This could happen with a load that misses all the way to main memory. One way of attempting to keep the CPU's functional units utilized is to execute multiple threads in the same CPU, having multiple PC's and multiple register files.

Course-grained MT processors execute one thread until they hit a major stall (cache miss, long latency instruction), and then switch to another thread. In contrast, fine-grained SMT processors issue instructions from some large pool of threads in a round robin fashion. Since there is only one instruction from a given thread in the CPU at a time, no interlocking is necessary at all. However, progress of individual threads will be poor.

Simultaneous multi-threaded processors attempt to execute concurrently from some (typically small) pool of threads. By having multiple threads to select from, the CPU is more likely to be able to find non-dependent, non-resource conflicting instructions that can be issued in parallel, hence maximising functional unit utilisation.

The down-side of executing multiple threads at the same time on the same CPU is that there combined cache footprint may cause excessive capacity misses, leading to poor performance. Hence, it is better to schedule co-operative threads from the same application rather than different system processes (to support the latter case, resources such as TLB entries will have to be tagged to indicate which threads they are valid for). If a single thread yields high ILP, SMT will provide no benefit (and can cause harm unless cache effects are considered). However, it's quite possible that two or more low-ILP threads may be executed with very significant overall speedup.