Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

High Performance Microprocessors

Date: Tuesday 2nd June 2009
Time: 09:45 – 11:45

Please answer any THREE Questions from the FIVE questions provided

This is a CLOSED book examination

The use of electronic calculators is NOT permitted.
1. Assume a single five stage pipeline, similar to that shown in Figure 1, but with four parallel execution units, as follows: an integer unit, a floating point (FP) adder, a FP multiplier and a FP divider. The integer unit performs loads and stores, including those to/from FP registers, as well as integer operations; it always completes execution in 1 cycle. The FP adder takes 4 cycles to complete, but it is pipelined in such a way that it can start execution of a new operation every cycle. The FP multiplier is similarly pipelined, but it takes 6 cycles to complete execution. The divider is not pipelined and takes 24 cycles to complete.

![Figure 1](image)

a) Explain the notion of a precise exception in a pipelined processor architecture. (2 marks)

b) Explain why it is difficult to implement precise exceptions in the multi-unit pipeline described above if instructions entering faster units are allowed to complete before those that previously entered slower units (i.e. later instructions are allowed to overtake earlier ones). (2 marks)
c) The following loop (written in C) is compiled into a short sequence of MIPS-like machine instructions:

\[
\text{for } (i := 4095; i >= 0; i--) \ a[i] := a[i] \times s;
\]

where \(a\) is an array of double precision FP values and \(s\) is a double precision FP value held in a double precision FP register. List the sequence of MIPS-like instructions without attempting any optimisation. (3 marks)

d) Indicate where stalls will occur in the execution of a single iteration of this loop using the above pipeline. Hence or otherwise determine the number of machine cycles taken per iteration of the loop. Assume the pipeline has full forwarding and state any other assumptions you need to make. (3 marks)

e) Show how the instructions in the sequence might be re-scheduled (in a different order) by a compiler so as to reduce the number of stalls that occur. Calculate the resulting number of machine cycles taken per iteration of the loop. (4 marks)

f) Show how the loop can be unrolled by a factor of two and then re-scheduled so as to achieve a further reduction in the number of stalls. State any further assumptions you need to make. Calculate the new number of machine cycles needed per iteration of the loop. (4 marks)

g) Estimate the effect of unrolling the loop by a factor of four. (2 marks)
2.  
   a)  Explain what is meant by ‘dynamic instruction scheduling’ and discuss why this has advantages over static scheduling as performed by a compiler.  (3 marks)

   b)  Sketch the structure of a floating point execution unit that uses Tomasulo’s algorithm to implement dynamic scheduling. Assume that the unit has a pipelined floating point adder and a pipelined floating point multiplier which are to execute in parallel. Assume also that the unit has Load and Store buffers to handle memory accesses.  (4 marks)

   c)  Describe how instructions are dynamically scheduled in the above structure. Your description should concentrate on the way in which the correct data for each instruction is assembled from registers, the execution units, or the memory. You should also describe how the correct data ends up in the registers or gets written to memory.  (6 marks)

   d)  Describe how Tomasulo’s algorithm avoids memory based Read After Write (RAW) hazards.  (3 marks)

   e)  How does the Tomasulo approach achieve dynamic register renaming and why is this superior to static register renaming implemented in the compiler? What other technique is available to implement dynamic register renaming?  (4 marks)
3. a) Explain why branch prediction is generally regarded as being essential in a modern high performance microprocessor. Explain why static branch prediction is generally unsatisfactory. (3 marks)

b) Describe the implementation of a simple 1-bit dynamic branch prediction buffer and explain the limitations of such a scheme. (3 marks)

c) Show how an extension of the above scheme to 2-bit prediction can help to improve the accuracy of prediction. (2 marks)

d) Explain how the actual behaviour of *preceding* branch instructions might have an influence on the expected behaviour of a branch, and how the simple 2-bit predictor above can be modified so as to take advantage of such *correlated* branch behaviour. (3 marks)

e) Explain how a tournament branch predictor can potentially predict branch behaviour even more accurately than the above schemes. (1 mark)

f) Describe the structure and operation of a branch target buffer, and explain why this has advantages over a branch prediction buffer. (3 marks)

g) Identify three kinds of indirect jump that are likely to occur in programs. Discuss whether any of the above branch prediction schemes can successfully predict the outcome of each kind of indirect jump and outline any other schemes that might help. (5 marks)
4. a) Describe the techniques of Direct Mapped, Set Associative and Fully Associative caching, comparing the potential performance and cost of each. Pay particular attention to the policies adopted for deciding (1) which cache line to replace when a new line is needed but there is no free space for it, and (2) whether or not to allocate a new line in cache when the access is a write. (7 marks)

A processor has a 4k byte 2-way set-associative data cache using 16-byte cache lines with a write allocate policy. The memory is byte-addressed and the cache line address is obtained by masking the memory address with 0x03F0 and shifting the result right 4 places. The following program fragment (written in C):

```
for (i=0; i<65536; i++) c[i] := a[i] + b[i];
```

is compiled and executed. Assume that the loop index i is held in a register, and that the three arrays (a, b and c) each comprise 64k words of 32-bit integers which are stored consecutively in memory, starting at address 0x10000000 (for element a[0]).

b) How many data accesses are made to the cache/memory hierarchy during execution of the above program fragment. (1 mark)

c) Bearing in mind that some cache misses are compulsory, what is the minimum possible number of cache misses that could be expected from any data cache used to support execution of the above program fragment; what is the corresponding cache hit rate. (1 mark)

d) How many of the data accesses will be cache misses if a Least Recently Used (LRU) replacement policy is used; what is the resulting cache hit rate. Explain your reasoning. (4 marks)

e) How many of the data accesses would be cache misses if a Most Recently Used replacement policy were adopted; what would be the resulting cache hit rate. (A Most Recently Used policy would always select the most recently accessed line to be replaced.) (3 marks)

d) How many misses would you expect if a Random replacement policy were used instead; what would be the resulting cache hit rate. Explain your reasoning. (4 marks)
5. a) Explain why it is necessary to implement some form of cache coherence protocol in order to monitor and maintain the values held in the various cache memories within a multi-core microprocessor chip. (3 marks)

b) Describe the operation of the MESI (Modified; Exclusive; Shared; Invalid) cache coherence protocol. Your answer should include a discussion of the MESI protocol states and an informal description of the processor and bus actions that cause transitions between them. There is no need to reproduce a complete state transition diagram. (6 marks)

The phenomenon of false sharing occurs when two cores execute codes that write repeatedly and frequently to two distinct memory locations (where each distinct memory location is used by only one of the two cores) that happen to map onto the same cache line.

c) Explain what will happen over time in a cache controlled by the MESI protocol you described above as the two cores repeatedly try to read and update their distinct memory locations. You may assume that no other memory accesses involve the same cache line. (5 marks)

d) Given that the decision where to map each memory location in cache is made by the processor architect while the decision where to map variables onto memory locations is made by the compiler, suggest ways in which false sharing might be avoided in practice. You should include suggestions for the programmer to use in cases where the system cannot be relied on to do the job. (3 marks)

e) Assume that you have been successful in avoiding false sharing in the above example. What will now happen as the two cores repeatedly try to read and update their distinct memory locations? You may continue to assume that no other memory accesses involve the same cache lines. What performance improvement, if any, would you expect to see as a result of this new behaviour? (3 marks)

END OF EXAMINATION