Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

High Performance Microprocessors

Date: Monday 18th January 2010
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 permitted providing they do not store text.
1. In the following fragment of code written in C, the variables \(i\) and \(n\) are of type \texttt{int}, while variables \(\text{sum}\), \(f\) and \(k\) are of type \texttt{double}. You should assume that \(n\) has a large positive value. The function \texttt{rand()} returns a random value of type \texttt{double} in the range \(-1.0 < \text{rand()} < +1.0\) with uniform probability.

\[
\begin{align*}
\text{sum} & = 0; \\
\text{for} (i = 0; i < n; i++) \\
& \quad \{k = \text{rand}(); \\
& \quad \quad \text{if} (k > 0) \\
& \quad \quad \quad \{f = i / n; \\
& \quad \quad \quad \quad k = f \times k; \\
& \quad \quad \quad \quad \text{sum} += k; \\
& \quad \}\}
\end{align*}
\]

The above fragment of C code is compiled into a sequence of MIPS-like instructions in which the variables in the C code have been allocated to registers as shown below.

<table>
<thead>
<tr>
<th>variable name</th>
<th>type</th>
<th>register(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>(int)</td>
<td>(r_{16})</td>
</tr>
<tr>
<td></td>
<td>(double)</td>
<td>(f_{4}, f_{5}) ; actually held as (1/n)</td>
</tr>
<tr>
<td>(i)</td>
<td>(int)</td>
<td>(r_{17})</td>
</tr>
<tr>
<td>(k)</td>
<td>(double)</td>
<td>(f_{6}, f_{7})</td>
</tr>
<tr>
<td>(f)</td>
<td>(double)</td>
<td>(f_{10}, f_{11})</td>
</tr>
<tr>
<td>(\text{sum})</td>
<td>(double)</td>
<td>(f_{12}, f_{13})</td>
</tr>
</tbody>
</table>

In addition, registers \(f_{0}, f_{1}\) and \(f_{2}, f_{3}\) are used to hold constant \texttt{double} values of 0.0 and 1.0, respectively. The resulting sequence of MIPS-like instructions is as follows.

```
LW      r16,   n       ; r16 holds n (int)
MOVE    r17,  r0       ; r17 holds i (int); r0 as source yields 0
MTC1     f0,  r0       ; f0 holds 0 (int)
CVTDW    f0,  f0       ; f0,f1 now holds 0.0 (double)
ADD     r18,  r0,  #1  ; r18 holds 1 (int)
MTC1     f2, r18       ; f2 holds 1 (int)
CVTDW    f2,  f2       ; f2,f3 now holds 1.0 (double)
MTC1     f4, r16       ; f4 holds n (int)
CVTDW    f4,  f4       ; f4,f5 now holds n (double)
DIVD     f4,  f2,  f4  ; f4,f5 now holds 1/n (double)
MTC1    f12,  r0       ; f12 holds 0 (int)
CVTDW    f12, f12      ; f12,f13 now holds sum = 0.0 (double)
```

**loop:**

```
JAL     rand           ; call \text{rand}() ; no arguments needed
MTC1     f8,  r2       ; result (double) returned in r2,r3 ...
MTC1     f9,  r3       ; ... transfer to f8,f9
CLED     f8,  f0       ; compare k with 0.0
BC1T    cont           ; skip next five instructions if k <= 0.0
MTC1    f6,  r17       ; f6 holds i (int)
CVTDW    f6,  f6       ; f6,f7 now holds i (double)
MULD     f10, f6,  f4  ; computes \(f = i/n\)
MULD     f8,  f10,  f8  ; computes \(k = f\times k\)
ADD     f12,  f12,  f8  ; adds k to sum
```

**cont:**

```
ADD     r17, r17,  #1  ; i++
BNE     r17, r16, loop ; exit loop if i=n, otherwise repeat
```

(Question 1 continues on the following page)
This sequence of instructions is to be executed on a MIPS-like processor based on a classic 5-stage pipeline with the following characteristics. Instructions can enter the pipeline at a maximum rate of one per cycle. Branch instructions are executed in the first two stages of the pipeline; their effect takes place immediately (i.e. is not deferred). The next instruction in the sequence is always fetched and, if the branch is not taken, the pipeline continues processing subsequent instructions as normal. If the branch is taken, there is a one cycle stall (due to the necessity of discarding the fetched next instruction) before the target instruction is fetched. If the branch condition is from the FP unit, there may be additional stalls while the condition is being computed. Unconditional jumps, such as JAL, similarly incur a one cycle stall while the target address is determined.

There are four parallel execution units: an integer unit, a floating point (FP) adder, a FP multiplier and a FP divider. The integer unit performs loads and stores and moves (such as MTC1 – move to co-processor 1, the FP unit), including those to and from the 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; type conversion instructions (such as CVTDW – convert to double from int) and FP comparison instructions (such as CLED – compare less than or equal to double) are performed in this unit. The FP multiplier is similarly pipelined, but it takes 6 cycles to complete execution. The FP divider is not pipelined and takes 24 cycles to complete. The FP units are fed from reservation stations implementing Tomasulo’s algorithm, so that instructions can be issued even if the values they require have not yet been computed. You may assume that instructions never need to wait for a free reservation station.

Execution of the instruction sequence proceeds in two phases. First the instructions before the label loop: are executed a single time each, then the instructions from loop: onwards are executed repeatedly, \( n \) times in total.

a) Assuming that no prior instructions cause stalls during execution of the given sequence, and that each call to \( \text{rand()} \) takes a constant number, \( C \), of cycles and does not induce any stalls in subsequent instructions, determine the total number of cycles required to execute the given code fragment. Explain your reasoning. You may assume that \( C > 10 \). (8 marks)

b) Why has the compiler decided to hold \( 1/n \) to use inside the loop, rather than \( n \)? What saving has this decision achieved? Explain your reasoning. (4 marks)

c) Can you find any opportunities to reduce the number of cycles required by rescheduling instructions (i.e. by moving instructions to different places in the sequence in order to reduce the number of stalls that occur)? If so, explain what these opportunities are and how many cycles can be saved by implementing them. (3 marks)
(Question 1 continues from the previous page)

d) As a computer architect you are asked to analyse the potential benefits of adding branch prediction to the MIPS-like processor. Discuss the possible approaches that might be used, and comment on the benefits that might be obtained for this specific code under each approach. (5 marks)

2. a) Draw a diagram for a floating point unit in a high performance microprocessor based on Tomasulo’s algorithm using a register renaming unit and a reorder buffer, and explain its operation for the 2 major categories of instruction (arithmetic, load/store). Include a description of how Tomasulo’s algorithm avoids memory based Read After Write (RAW) hazards. (8 marks)

b) Explain what is meant by the term speculative execution. Under what circumstances might speculative execution be beneficial to the performance of a microprocessor? Why is it essential that a reorder buffer be used for speculative execution? (4 marks)

c) An extreme form of speculative execution can be based on value prediction, that is, predicting the value that an issuing instruction will produce and putting this directly into the assigned physical register and continuing computing as if this is the correct result. Under what circumstances might this improve performance? (3 marks)

d) An integer instruction in a program successively produces the values 0, 1, 2, 4, 8, 0, 1, 2, 4, 8, and so on in a repeating cycle. A context value predictor is used to predict the next value in the sequence by building up a (possibly partial) history of the values in the form of a table that stores the least significant three bits of the value produced by that instruction on its two previous executions, and using this to look up the full predicted value in a companion table. Draw a diagram of the storage structure necessary to implement this scheme, and show the contents of the key parts of the structure after the cycle has completed several times. (5 marks)
3. a) Explain why high performance microprocessors attempt to issue more than one instruction in the same clock cycle.  
   (2 marks)

   b) Discuss the limitations that may be placed on the combination of instructions that can be issued in one clock cycle, and why these limitations are necessary.  
   (3 marks)

   c) The following loop of MIPS-like instructions is to be executed on a dynamically scheduled, 2-way superscalar microprocessor which has one pipelined integer/load/store unit alongside its floating point unit, and which executes branches in deferred fashion. A maximum of one integer/load/store instruction plus one floating point instruction may be issued each clock cycle.

   ```
   start:   LD      f0, 0(r1)
           MULD    f4, f0, f2
           SUB     r1, r1, #8
           BNEZ    r1, start
           SD      f4, 8(r1)
   ```

   Determine the minimum number of clock cycles required to issue (i.e. commence) each iteration of this loop and show the (2-way) instruction schedule that achieves this.  
   (4 marks)

   d) Show how a compiler can unroll this loop by a factor of two, and reschedule the new sequence of instructions to determine the best (2-way) instruction schedule per iteration of the resulting loop. What is the resulting number of clock cycles required to issue (commence) each iteration of the original loop? What property of the original program does this loop unrolling rely on?  
   (4 marks)

   e) Repeat part d), but using an unrolling factor of eight. What do you conclude about the potential benefits of loop unrolling?  
   (5 marks)

   f) Explain in outline how the compiler might deal with unrolling of loops for which the total number of iterations is not determinable at compile time.  
   (2 marks)
4. a) Describe how a 2-way *set-associative* cache is organised and how it operates in response to read and write requests that reach it from the processor. 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. (5 marks)

A processor has a 16k byte 2-way set-associative level 1 data cache using 32-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 0x00001FE0 and shifting the result right 5 places. The following program fragment (written in C) is compiled and executed.

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

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. Explain your reasoning. (2 marks)

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? Explain your reasoning. (A Most Recently Used policy would always select the most recently accessed line to be replaced.) (4 marks)

f) What would be the effect of adding a single entry *victim cache* to the above system, using either of the above replacement policies. Explain your reasoning. (4 marks)
5. a) Describe the problem of cache coherence. Explain how this problem may arise in a scalar microprocessor that has separate data and instruction caches at Level 1. (3 marks)

b) Explain how a ‘snoopy bus’ can be used to overcome the above problem. (2 marks)

c) Describe how cache coherence becomes a more serious problem in shared memory multiprocessors. In particular, explain why a ‘snoopy bus’ cannot be used to overcome the problem when the number of processors gets large. What alternative approach might be used and what new problems would it entail? (5 marks)

d) Discuss the difference between write invalidate and write update protocols for maintaining coherence in a shared memory multiprocessor. (4 marks)

e) 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)