DELFT UNIVERSITY OF TECHNOLOGY
Faculty of EEMCS
Computer Engineering & Embedded Systems
MSc Programmes

CS – **ET4074 Modern Computer Architecture**

IN4340 Embedded Computer Architecture

November 3, 2010, 2PM – 5PM

- The duration of the test is 3 hours.
- The test is open book, Hennessy and Patterson, Computer Architecture: A Quantitative Approach.
- The test is worth 50% of your final grade.
- Write your name, student number, and the course you are enrolled in (ET4074 or IN4340) on each answer page.
- This exam consists of 5 assignments.
- When answering the questions detail the way you obtained your answer, i.e., present the line of reasoning and the mathematical derivations behind your solution. Answers without explanations will be given **NO** credit. Be explicit and concise and use the textbook's terminology. Vague and/or difficult to understand text will not be taken into account. Remember, numbers and formulas are explicit and concise.
- Read through the entire question, then think about what you are asked to provide before answering; make sure that you're answering only what the question asked.
- State all assumptions that you make and explain why you have made them.

Assignment 1 – Miscellaneous questions (40 points)

1. (5 points) Consider a CPU with the conventional IF, ID, EXE, M, WB stages. For a floating-point instruction the execution stage, EXE, takes 5 cycles, and all the others 1 cycle. Assuming 2 floating-point instructions, the first producing an operand for the second, what is the minimum distance between such 2 instructions in a program, such that no stall occurs, in the cases: (i) with forwarding logic and (ii) without forwarding logic.

2. (5 points) Would dynamic instruction scheduling make sense without speculative execution of instructions? Explain your answer.

3. (5 points) Would branch prediction make sense without speculative execution of instructions? Explain your answer.

4. (5 points) Suppose we want to achieve a speedup of 10 on a 12-core machine. What fraction of the total execution time can be sequential?

5. (4 points) Explain the differences, with respect to the global system performance, of snooping cache coherence in the following cases: (i) a multiprocessor with nodes connected via a bus; and (ii) a multiprocessor with nodes connected via an on chip network (NoC).

6. (4 points) Why is VLIW a better design option for embedded systems than for general-purpose machines?

7. (4 points) Generally speaking, TLBs benefit more from high associativity than caches. Why is this the case?

8. (4 points) Give 2 reasons why L1 caches are split in instruction and data caches.

9. (4 points) Which of the following situation has a larger need for high bandwidth: (i) at instruction fetch or (ii) at the load/store memory port? Explain your answer.

Assignment 2 – processor performance and power consumption (20 points)

Consider a CPU that has a single simple pipeline that is clocked at 1GHz. This CPU does not have a functional unit for a given operation "OP" and the compiler has to generate 10 one-cycle instructions to implement "A OP B". Adding a functional unit to implement an OP instruction increases the clock period of the CPU, so that the maximum clock frequency that can be achieved is 800MHz. Assume and ideal CPI for all other instructions.

1. (10 points) What is the fraction of OP instructions that a program should have so it makes sense, from a *performance* point of view, to add a functional unit dedicated to perform OP?

2. (10 points) Assuming that the power consumed by this processor each cycle is proportional to its frequency (the frequency decrease from 1GHz to 800MHz is not accompanied with a voltage scaling), what is the minimum fraction of OP instructions that a program should have to make sense, from a power point of view, to add a functional unit dedicated to OP?

Assignment 3 – branches (15 points)

1. A branch has the following history (T stands for taken, NT for not taken):

T, T, T, NT, NT, NT, T, NT, NT, NT, NT, T, T, T, T

a) (2 points) How often would a branch history table with 1-bit entries correctly predict this branch? Assume the initial prediction is taken. Show how you obtained your answer.
b) (3 points) How often would a branch history table with 2-bit entries correctly predict this branch? Assume the initial prediction is "weakly taken" (10). Show how you obtained your answer.

2. We define the effectiveness of instruction fetch as the fraction of instructions committed from the total number of instructions fetched. Ignore any stalls the instruction fetching might experience in case of hazards or cache misses.

a) (5 points) Consider a program in which 1 out of 5 instructions is a branch. How does the instruction fetch effectiveness vary as a function of the number of the CPU pipeline stage where the branch is decided?

b) (5 points) For the same program as in (a), assume a branch target buffer with 70% hit rate and branch prediction accuracy of 97%. How does the instruction fetch effectiveness vary as a function of the number of the CPU pipeline stage where the branch is decided in this case?

Assignment 4 – Caches and virtual memory (15 points)

A CPU has a 2 ways associative L1 data cache of size equal to 1MByte, and with a block size equal to 64Bytes and the CPU issues 32 bit addresses.

1. (2 points) How many sets does this cache have?

2. (2 points) How large is the memory that stores the cache tags?

3. (6 points) Describe the possible options to increase the 1Mbyte cache to a 2MByte cache. Which bits ranges (tag, index, block) would change in this case? How are the compulsory, conflict, and capacity misses affected by the cache size increase?

4. (3 points) What are the options to organize the 1MByte (associativity, number of sets, block size) such that it can be physically tagged, virtually indexed in a virtual memory system with page sizes of 128KB.

5. (2 points) In a cache as in (4), assuming the physical address has 30 bits, and there are 3 access rights bits, what is the size of a TLB entry?

Assignment 5 – Multiprocessors (10 points)

A critical region is a section of code which must be executed by only one process/processor at the time. Aquire/release(lock) are syncronisation operations that can be used to guard a critical region, as in the following example:

```
...
// begin of critical region
aquire(lock);
 ...
 // code of the critical region
 ...
release(lock);
// end of critical region
```

Aquire(lock) works as follows: atomically tests if the value at a given memory location (the "lock") is 0, and if so it sets it to 1 and allows entering the critical region. Entering the critical section should not be permited as long as the value at the test memory location is 1. Implement, in assembly code, two functions to allow entry (aquire()) and exit (release()) of a critical region, utilising load link (LL) and store-conditional (SC) instructions. Do not forget to comment the code.

-- End of test --