Bonus Assignment 4 — CPU Architecture

Instructions

- The assignment should be solved *individually*.
- The solutions should be properly *motivated*, a few short sentences is usually enough.
- Answer the questions in *your* own words. Copying answers from other sources, such as the text book, is not acceptable. You may quote other sources, in that case, make sure to include the source and say how you interpret the quoted text.
- Solutions should be given in *English* or *Swedish*, English is preferred.
- Solutions should be posted in *Andreas Sandberg's* mailbox (4th floor in house 1, mailbox number 147) before the deadline.
- Fill out (preferably on your computer) and include this cover page when you hand in your solution.
- Unless you have an excellent handwriting, please use a text editor, $\TeX$ or a word processor to write your answers.
- You need to score at least 11 points to get a bonus point on the exam.

Deadline

To get a bonus on the exam, correct solutions must be handed in before the deadline specified on the course homepage. Solutions handed in after the deadline will be marked on a best effort basis and will not result in any bonus on the exam.

Student

Name

Civic registration number (personnummer)

Email address

Date

1 ISAs

1. Over the years, several different classes of instruction sets have evolved (and died). Three common strategies to handle operands is to use a stack-based ISA, an accumulator based ISA, or a register based ISA. All of them have been implemented in hardware at some point.

Stack based instruction sets were once used by HP in their HP 3000 series. Today they are mainly used in virtual machines, e.g. the Java Virtual Machine (JVM).

Accumulator machines have been popular since the dawn of time, the iconic PDP-8 is one example. They are still common in microcontrollers, such as the PIC-series. The x86-architecture evolved from this class, but is today (mostly) a register based architecture.

Almost all modern machines are register based and generally do not use fixed function registers. A good example in this class is the MIPS.

a. One of the problems with *stack machines* is that it is generally hard to make efficient hardware implementations. However, they have one large benefit that has made them successful in the JVM. What is the main benefit of *stack machines* over *register machines*? (1)

b. Most operations require 3 different operands, 2 input operands and 1 destination operand. In *accumulator based* architectures, one of the input operands is always the accumulator. What is normally the source of the 2nd operand? (1)

2 Hazards

2. There are three different classes of hazards, *structural hazards*, *data hazards* and *control hazards*.

a. What is a *structural hazard*? (1)

b. Describe how a *control hazard* can be transformed into a *data hazard*. Give *two* examples. (2)

3. Data hazards can be further divided into three different types:

RAW Read After Write

WAR Write After Read

WAW Write After Write

Op_i and Op_{i+1} are two consecutive instructions in program order.

a. A RAW hazard occurs when Op_i modifies A and Op_{i+1} reads A before Op_i has committed its new value. Describe how this situation can occur in a simple 5-stage pipeline and a simple hardware solution. (1)

b. Why can't WAR and WAW hazards occur in a simple in-order 5-stage pipeline? (1)

3 Instruction scheduling

4. There are two main strategies to exploit instruction level parallelism (ILP) and feed multiple parallel execution units. What is the main difference between *VLIW* and *super scalar* processor? Think about how functional units are scheduled. (1)

5. *Tomasulo's algorithm* introduces several new hardware structures to support *out-of-order execution*.

a. When instructions are issued, they arrive are put in a *reservation station*. What are *reservation stations* used for? (1)

b. What is the *reorder buffer* used for and what does it guarantee when instructions commit? (1)

6. A highly desirable feature in a processors is *precise exceptions*, which guarantees that all side effects of instructions happening before the exception are visible and no side effects from later instructions are visible.

How can precise exceptions be implemented in a CPU that implements out-of-order execution using Tomasulo's algorithm? (1)

4 Branch Prediction

7. A very simple branch predictor is the 1-bit branch history table. Describe how it works. (1)

8. What is the difference between the *1-bit* and *2-bit* branch prediction scheme? What does it try to optimize? (1)

9. The *branch target buffer* (BTB) allows something that is known as *branch folding*. Describe how the BTB works and how it can improve performance by folding branches. (1)