Computer Architecture 4

Assessed Exercise 2
Processor Architecture

Handout: Thursday, February 13, 2014
Handin: 4:30pm Friday, March 7, 2014

Introduction

The purpose of this exercise is to learn in detail how a processor works: how a digital circuit can execute instructions and run programs. The exercise requires you to understand a working circuit called M1 that implements the Sigma16 architecture, and then to work out how to modify it. There are two parts; one part requires modifying the M1 circuit to implement a new instruction and testing the correct execution of the instruction, while the other part is a paper design that focuses on the chief design decisions but does not require implementation.

Part 1. Implementation of the loadxi instruction

Add a new instruction to the Sigma16 architecture, as defined below. Modify the datapath and control, as needed, in order to implement the new instruction in the M1 circuit. Modify the simulation driver so the operation of the instruction can be observed, and demonstrate the execution of the instruction using a machine language test program.

The new instruction is *load with automatic index increment* (the `loadxi` instruction). Its format is RX: there are two words; the first word has a 4-bit opcode f, a 4-bit destination register (the d field), a 4-bit index register (the sa field), and a 4-bit RX opcode of 7 (the sb field). As with all RX instructions, the second word is a 16-bit constant called the displacement. In assembly language the instruction is written, for example, as loadxi R1,$12ab[R2].

The effect of executing the instruction is to perform a load, and also to increment the index register automatically. The effective address is calculated using the old value of the index register (i.e. the value before it was incremented.) Thus the instruction loadxi R1,$12ab[R2] performs R1 := mem[12ab+R2], R2 := R2+1.

Test your new instruction by writing a machine language program that uses it, and running the program on the circuit. There should be at least one or two loadxi instructions in the program; an excellent test would be to write a loop that traverses an array using loadxi to fetch the current element and to increment the index.

Part 2. Designing a multiply instruction

In this part you should sketch how the circuit would be modified to support a new instruction, but you do not need to get it fully implemented and working.

The Sigma16_M1 processor has a multiply instruction, but this instruction does nothing at all. Design a modification to the M1 circuit that would implement the instruction. You should explain your approach to the design, and outline how the datapath and control would be modified, but you do not need to carry out a complete implementation or to test its execution.

The multiply instruction uses the RRR instruction format, with operation code 2. For example

```
  mul R4,R12,R3
```

is represented as `24c3`, and it performs the action `R4 := R12 * R3`.

The result is represented as a 16 bit word, so there will be an overflow if the operands are too large. However, you do *not* need to detect overflow or do anything special if overflow occurs. Your solution should work for negative integers as well as nonnegative integers.

Your design should not cause a large increase in the critical path depth of the processor circuit. This means that a combinational multiplier is infeasible; a functional unit should be used instead. If you wish, your design can use the multiplier functional unit covered in lectures.

How to proceed

- Begin by studying the M1 circuit and the Sigma16 architecture.
- Run the `ArrayMax.hs` example program, and study its execution. Compare the actions carried out by the circuit with the Sigma16 emulator while running this program.
- Think about what changes to the datapath and/or the control algorithm are needed in order to implement the new instructions.
- Write an explanation of your approach.
- Make the required changes to the system.
- Test your new circuit by running a test program on it.

Assessment

The exercise will be marked out of 100 marks, and it counts for 10% of the CA4 assessment. The two exercises together are worth 20% of the assessment, while the examination is the other 80%. Each part of this exercise (loadxi and multiply) is weighted equally.

What to hand in

The handing will be via the Moodle page. Your handin should consist of a single file with a name of the form John_Smith.tgz (but change it to your name, not John Smith's). (If you prefer you can use a zip format instead of tgz, but other formats, such as rar, are not acceptable.) This file defines a directory named John_Smith containing the following files:

- StatusReport.txt — a text file containing your status report. This should contain:
  - Identification: your name, the exercise number (Computer Architecture 4, Assessed Exercise 2), and the date.
  - Documentation for your solution to the loadxi instruction.
  - An extract from the simulation output demonstrating your instruction. Annotate this, showing where key events occur. Don't give the full simulation output, include just the parts where your instruction is executing.
  - Your design for the multiply instruction
- The files needed to run your modified system.