Computer Architecture

Assessed Exercise 2

Processor Circuit

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.

Part 1. Implementing a new instruction: loadxi

Add a new instruction `loadxi` to the M1 circuit for the Sigma16 architecture. Modify the datapath and control, as needed, in order to implement the new instruction in the M1 circuit, and modify the simulation driver so the operation of the instruction can be observed. 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 secondary opcode of f (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.

(Historical note: many real computers have this instruction. The idea is that computers spend a lot of time iterating over arrays, and in each iteration you need to load $x[i]$ into a register and also increment i. The loadxi instruction lets you do this work in one instruction rather than two.)

Test your new instruction using a machine language program that calculates the sum of the elements of an array X, which contains n elements. To do this, start with a program that simply calculates the sum using an ordinary iteration, with an index i that goes from 0 to $n-1$. (This program is one of the unassessed exercises, and you may use the model solution.) Then change the program by replacing the load instruction for $x[i]$ by a loadxi instruction, and getting rid of the explicit add instruction to increment i. Run the modified program on the circuit, and verify that it gets the correct result.

If you simply remove the add instruction that increments i from the model solution, the subsequent memory addresses will shift down, which is inconvenient for comparing the execution of the two versions of the program. Instead of deleting the instruction, just replace it with something like add R0,R0,R0 which won't do anything and will leave all the instructions and data values with the same address as before.

Part 2. Controlling a functional unit: mul

The Sigma16 architecture has a multiply instruction, but in the M1 circuit this instruction does nothing at all: it acts as a *nop* (no operation). Modify the M1 circuit so the multiply instruction works properly. Use the multiply functional unit that has been discussed earlier.

We will actually implement a simplified version of the instruction, which operates on binary numbers and assumes the result fits in a 16-bit word. The full Sigma16 architecture actually is a little more general, but you don't need to consider that for this exercise.

The instruction `mul R1,R2,R3` places the product of R2 and R3 into R1. The numbers are assumed to be binary, and overflow is ignored. 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`.

Write a test program that uses the mul instruction and determine whether it works. Be sure to perform several multiplications; some should involve small numbers and some should involve relatively large numbers. The instruction should finish when the functional unit has produced a result; in other words, the instruction will take a variable number of clock cycles, depending on the sizes of the numbers being multiplied.

How to proceed

- Begin by studying the M1 circuit and the Sigma16 architecture.
- Run the `ArrayMax` example program, and study its execution. Compare the actions carried out by the circuit with the Sigma16 emulator while running this program.
- Study the problem sets and solutions. You have been given a lot of hints!
- 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.
- Provide two test programs, `LoadxiRun.hs` and `MulRun.hs`. These should be main program files that contain a machine language program, and a call to the circuit simulator to run it, similar to ArrayMaxRun.hs.

Work in small groups

This assessed exercise should be carried out in a small group consisting of two to four students. Any amount of discussion and shared work within the group is fine, and the product you hand in counts fully for each member of the group.

You may keep the same group you used for the first exercise. If you change your group membership, please email the lecturer with the names and matriculation numbers of everyone in your group, and use *CA group members* as the email subject line.

Handin

The handin will be via the Moodle page; see the page for detailed instructions on submission.

Your handin should consist of a single file with a name of the form Exercise2.zip. (If you prefer you can use tgz, but other formats, such as rar, are not acceptable.) This file defines a directory named Exercise2. (It is good practice that unpacking your file should produce a *directory* — it is unprofessional and bad style just to dump a lot of files into the user's directory.) The directory should contain the following files:

- StatusReport — your status report. This may be a text file (.txt), a Word document, or a pdf.
- The files needed to run your modified system.

The status report gives general information and documentation about your solution. The opening lines should follow a key-value format, where each keyword is followed by a colon : and a space, and then the value is just text. You should define the keys that are illustrated in the following example of a status report. There should be one author line for each member of your group; the author line contains your name, followed by a comma, followed by your matriculation number.

```
course: CA
exercise: 2
date: 2018-11-29
author: Jane Doe,  9875434
author: John Smith, 12345678
```

After these opening lines, the status report should describe the status of your solution. It should say whether each part of the exercise has been completed. Does it compile? Does it appear to work correctly? How did you test it? Are there any aspects that appear to be not quite right? Are there any especially good aspects you would like to highlight? Explain your approach to solving the problem in the status report.

The status report should contain an extract from the simulation output demonstrating the execution of `loadxi` and `mul`. instruction. Annotate this, showing where key events occur. Don't give the full simulation output, include just the parts where your instruction is executing. Separate the paragraphs in the status report with a blank line.

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. Each part of the submission will be marked in the following categories:

- Documentation (20%). Does the status report state clearly what you did? Have you shown what changes you made to the existing circuits, and why?
- Correct implementation (40%). Have you correctly implemented the new instructions by modifying the circuits?
- Testing (20%). Have you provided test data that provides evidence that your solution works? Does your test data shows normal execution of the instructions, as well as boundary cases?
- Style and quality (20%). Have you followed the submission guidelines? Is your circuit design clear?

You can get partial marks for a partial solution. If you can't get one of the instructions to work, go ahead and document your approach and sketch the circuit and test data for that instruction.