

CSE321: COMPUTER ARCHITECTURE

Lab #3: Simple 12

Modifying the ISA and Control Path Design

Due: October 6,7 2004

1 Introduction

This is the second half of the Simple 12 lab that you started two weeks ago. In the first part you wrote two assembly programs and finished off the datapath design in VHDL. In this part, you will create two new instructions, rewrite your Bubble Sort program, and redo the control logic for your Simple 12.

Below is the same figure that was in the first Simple 12 lab. It is included here for your convenience.

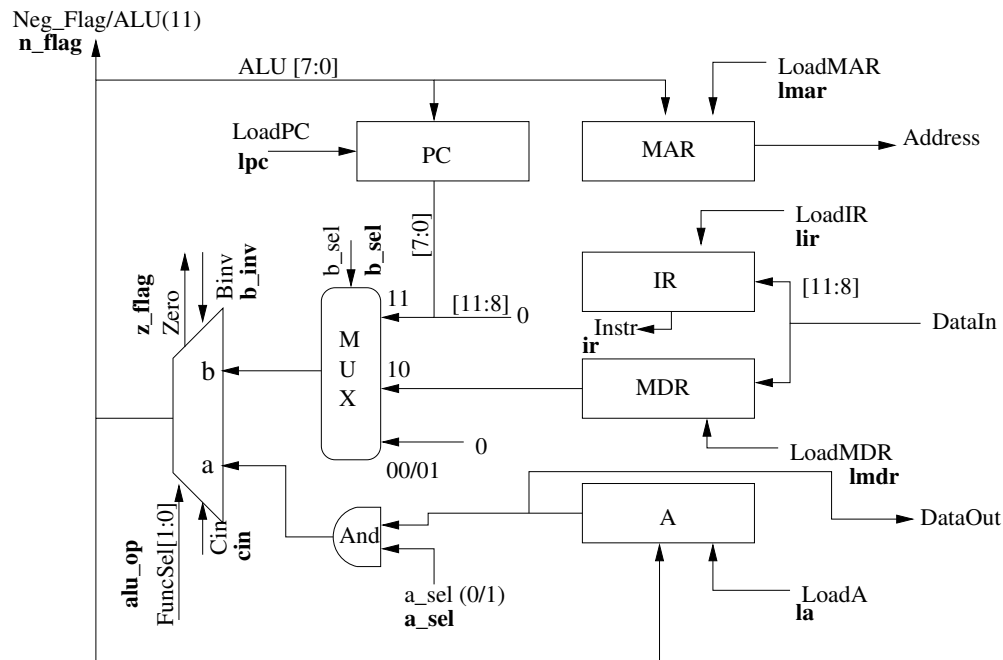


Figure 1: Simple 12 Datapath, with Control Signals

2 Add Load Indirect/Store Indirect Instructions

Previously, you may have found that programming the Simple 12 was difficult because it required the use of self-modifying code. Self-modifying code was needed because there was no effective way to load from or store to an array without changing the instruction itself. One solution to this is to add a new instruction to the Simple 12 which performs a “Load indirect.” A load or store indirect accesses the memory stored in another piece of memory (aka: a pointer). Or, in RTL:

$$LOADI : A \leftarrow M(M(X)) \tag{1}$$

$$STOREI : M(M(X)) \leftarrow A \tag{2}$$

It is possible to add Load and Store indirect (LOADI and STOREI) instructions to the Simple 12 processor *without* changing the datapath. Your first task is to develop a detailed RTL design for the LOADI and STOREI instructions. This can be done without modifying the datapath that was used in Lab 2.

Having created the RTL for these instructions, rewrite the Bubble Sort program to utilize these instructions. Do not delete your original version as you will want to compare the operation of the two Bubble Sorts.

2.1 What to Turn In

- State diagrams for LOADI and STOREI.
- The assembly code for your new Bubble Sort.

3 Microcode the Simple 12

In the course directory, under `lab3`, you will find a directory titled `s12_vhdl` which contains the VHDL code for a Simple 12. Unlike last lab, this Simple 12 has a complete data path but lacks a control path. The file `control.vhd` contains the skeleton code for a microcode-based Simple 12 control path. A few entries in the ROM table have already been filled in. **Complete the rest of the ROM table (including the STOREI and LOADI instructions)**. You may find it helpful to do a few things before extending the table:

- Analyze the entries in the table which are already given.
- Refamiliarize yourself with the S12 datapath, and the signal names relevant to the control logic which are in **bold** (see Fig. 1).
- Look over the format for the ROM table entries (see Table 1).
- Look at the `control.vhd` from last week which uses a “hardwired” state machine.

Signal	Bit(s)
Condition Select	23 to 21
Address Select	20
Next Address	19 to 14
la	13
lpc	12
lmar	11
lmdr	10
b_inv	9
cin	8
alu_op	7 to 6
b_sel	5 to 4
a_sel	3
we	2
lir	1
unused	0

Table 1: ROM Format.

This ROM table is similar to the one presented in class, but with a few small changes.

- The “Read” and “Write” signals have been combined into a “write enable (WE)” signal.
- Load IR signal is included.
- Because there is some “lag” as signals propagate from the control path to the data path, some control signals appear in an earlier state than in the table presented in class. For example, because we need the IR register to have the current instruction in the IFetch state in order to decide where to branch to, we have to load the IR register in the previous state (such as EXECUTE for ALU operations).

3.1 What to Turn In

- ROM table for Simple 12 (with LOADI and STOREI).
- Comparison of microcoded vs. hardwired control logic (use last lab’s `control.vhd` for comparison).

4 Simulate

First, create a new directory for these simulations and then copy the VHDL files from `<course_dir>/labs/lab3/` into this new directory. Next, since a `datapath.vhd` file was not included in these files, copy the one you created in the previous lab into this directory. Lastly, invoke `vsim` and compile the files in the same order that you did in the last lab. Simulate the two Bubble Sort programs (also try your Fibonacci numbers program, but do not hand in waveforms for it here). Additionally, compare the two Bubble Sorts and look at the following issues:

- Code Size: How much memory did the instructions for the code take?
- Execution Time: How many cycles did the code take to execute?
- Programming Ease: Subjectively, how difficult was it to write the code? Objectively, how much time was spent coding?

4.1 What to Turn In

- Waveforms for both Bubble Sorts (and Fibonacci if you want).
- Discussion comparing the two Bubble Sort programs.

5 Ponder the significance of it all

As technology has changed, computers have changed. From the colossal multi-million dollar vacuum tube-based machines of the 50s to the inexpensive commodity laptops of today, the role of computers has also changed. As the economic role and technological foundation of computers have changed, the design and architecture has also changed (see Table 2).

Time Period	Dominant Control Technique	Dominant Technology	CPU Size
40s-50s	Hardwired	Vacuum tubes	Room
60s-70s	Microcoded	Transistor	Several Boards
80s - Today	Hardwired	Transistor	Single Chip

Table 2: Trends in control and technology.

Using what you know about control logic, economics, assembly programming, compilers, software design, data structures, history, and anything else you think relevant **construct a theory as to why control has changed from hardwired to microcoded and back, and why/if it may change back.**

6 Conclusion

For this lab you are expected to hand in a formal report (in two weeks). In addition to what was listed in the **What to Turn In** sections, also discuss your experiences with writing Simple 12 assembly code, creating new instructions, microcoding, and using ModelSim.