Course Project – Designing a Pipelined Processor

In this project, you will design and implement a pipelined processor using Verilog. The ISA for this processor is a variant of the LC-3 from CS310. We'll call this machine the LC-3.5; its ISA is given below. It is basically an LC-3 with the indirect load and store instructions omitted, and with some useful arithmetic instructions added.

You will be working on this project for the remainder of the semester, although there will also be some homework and reading assignments to do in parallel with it. You will work in pairs, so each of you will need to select a partner to work with on the project. Accordingly, your first project task is to select a partner. You should email me (fussell@cs.utexas.edu) your selection no later than Thursday October 15, 2009. Each member of the team should send an email so I can verify that you both understand you will be working together. If anyone fails to select a partner, I will appoint one shortly after that date.

The first phase of the project will allow you to get familiar with the LC-3.5. This is due by 11:59pm on Thursday October 29, 2009 and will be counted as homework 6, worth 100 points.

HW 6 - Bug Hunt

Download and examine the unpipelined LC3.5 implementation from the class Verilog page (see below). There are eight bugs hidden in the Verilog code. Your job is to understand the code and find the bugs. You will need to learn all of the ins and outs of the LC3.5 implementation and testbench (.v files). Here are your tasks:

1. Turn in the following electronically. You should be able to complete this even with the buggy Verilog code.
   - A diagram of the LC3.5 datapath with control, to a level of detail similar to that in Figure 4.17 on page 322 in P&H. Please indicate the width of various wire busses in the design. Note that there are a variety of flip-flops in the datapath - please include these in your diagram. For your convenience, a picture of the original LC-3 datapath from Patt and Patel is included below
   - A paragraph or two describing how the control logic works (how the instructions are sequenced in general).
   - A step-by-step description of how the JSRR instruction proceeds through the datapath. Be very detailed here indicating how all of the control signals are set for each cycle required to execute the instruction.

2. Find the bugs! Use your debugging skills to create a suite of assembly programs that help you find the bugs. The most credit will be given to the teams that find all eight bugs. You are to turn in the following:
   - A description of each bug and how it affects the execution of the program.
   - The location of the bug in the Verilog code (filename and line number).
   - A suggested correction.

- A short LC3.5 assembly program that demonstrates the bug. Note that if you find six bugs, you should turn in six different LC3.5 programs, one for each bug. Make sure to comment your bug programs, to include a description of how the bug is exposed.

Turn in an electronic copy:

```
csh> turnin --submit dongli hw6 <bug_i.asm>
```

Note that we have an assembler and simulator for the LC3.5 machine. The second page of this handout shows an example of how to run the assembler and LC3.5 simulator, as well as instructions on how to unpack, compile, and run the Verilog for the buggy LC3.5.

Note: For reasons explained below, you will need to give us advance notice if you plan to use slip days on this project. If you do not notify us in advance, we will assume you will not need them and we will not allow them.

Environment Setup

Following are some instructions to get you started on running LC3.5 programs on the instruction level simulator as well as on the Verilog implementation of the LC3.5. You can download lc35buggy.tar.gz from (http://www.cs.utexas.edu/~fussell/courses/cs352h/assignments/lc35/lc35buggy.tar.gz)

Once downloaded, extract the files as shown below. You should have five directories.

```
csh>  cp <tarfile> .
csh>  tar –xzvf  lc35buggy.tar.gz
csh>  ls
asm/ lc35/ os/ pli/
```

Note that some files assume the directory structure as above. Do not rename or move these five directories.

To setup an environment for this assignment, you should source the Verilog environment as described on the Verilog page (note, this has changed since Homework 5).

If you are using bash:

```
bash> source /p/bin/lc35/class_sh.src
```

If you are using csh,

```
csh> source /p/bin/lc35/class_csh.src
```

Now you have to compile the LC3.5 'operating system'.

```
lc35buggy> cd os
lc35buggy/os> make
```

If all the environments are set correctly, this should succeed without errors and will generate .obj files. This is the end of environment setup.

Sample Program Execution

The following is a set of commands you can execute to assemble and run a simple program (fibonacci). Note that the fibonacci program may not actually work on the buggy Verilog (but it will work on the LC3.5 instruction simulator). You should spend a little time making sure you understand how each of the commands works.

assemble test programs

```
csh> cd asm
csh> lc35_asm fib.asm
```

Run new lc35 simulator, --nox gives you a non-X-windows mode.

```
csh> simxlc3 --nox
(lc3) help
(lc3) load fib.obj
(lc3) go
(lc3) quit
csh> cd ../lc35
csh> make_lc35 lc35.f
csh> lc35_vsim +vhex+../asm/fib.vhex +maxc5000
```

Note that +maxc5000 sets the maximum number of clock cycles executed to 5000. The default is 10000 - this flag is just a way to ensure that the Verilog simulation doesn't accidentally get into an infinite loop. Check out the details in `lc35_top.s`

Pipelined Design in Verilog

Once you have completed the homework 6 bug hunt, you may proceed to the major portion of the project. This will be due on the last class day, Thursday December 3, 2009 by 11:59pm.

For your convenience you will be provided a correct unpipelined Verilog implementation to start with once everyone has turned in the first part of the project (hw 6). The unpipelined version is a multi-cycle implementation in which each instruction proceeds through the same five steps of execution.

You should keep in mind the following hints as you build your pipelined processor:

- Before you start "hacking" at the code, sit down and design the pipelined processor with your partner. Draw datapath diagrams, control state machines, etc.
- Determine what hazards arise in your pipeline and how to avoid them. You are required to use bypass paths wherever possible to limit the negative effect of pipeline bubbles.

- Start with a simple predict not-taken branching scheme. If you have time, you can develop a more sophisticated branch predictor for extra credit.
- You should use the test programs you generated for the bug hunt and augment your suite with more. We have the technology to automatically test your code on programs you haven't seen before...and we will!
- Do not change the external interface to the LC 3.5 as seen by the stimulus (test) file - we will be relying on that interface to be stable so we can run our tests.

To help you along this path, you must to turn in the following:

- A project plan. The ideal plan should include an overview of the design, a listing of the major tasks and your estimate of the hours required per task. The extent and thoroughness of the plan is up to you. You will receive full credit for turning in any sensible attempt at a plan. Your plans should be e-mailed to fussell@cs.utexas.edu no later than Tuesday, Nov. 3rd at 11:59pm. PDF files are preferred. You are more than welcome to discuss your plans with me before or after turning them in. I can schedule additional office hours, if needed.
- On subsequent Tuesdays (11/10, 11/17, and 11/24) you are to turn in (also by e-mail to my account, no later than 11:59pm) a brief report (half a page is fine) indicating your progress to plan, any changes you wish to make to the plan and a statement of how many hours you put into the project the previous week.
- These four reports will account for 20% of your project grade. No late days on these.

You are to turn in the following:

- All of your verilog files (`.v`)
- Any `.h` files you use.
- A top level stim file (similar to or the same as the `lc35_test.v` provided with the unpipelined version).
- An `lc35.f` file, listing all of the files required to build and run your verilog simulator.
- Two test programs (`prog1.asm` and `prog2.asm`) along with two `.out` files (`prog1.out`, `prog2.out`) which show the results of your pipelined processor running your test programs.
- A short report (1-2 pages) describing your design. This can include diagrams, descriptions, etc. - whatever you need to describe how your design works. You should also do a performance comparison of your pipelined design relative to the original unpipelined design. For example, on your test programs determine how many cycles it takes to run the test on both versions.

You should submit your files using the following:

```
csh> turnin --submit dongli project <filename>
```

If you find that you have of extra time and wish to pursue extra credit (and you have thoroughly tested and debugged your pipelined code), then see me. I have a variety of interesting enhancements that you might be interested in tackling, including:

- More sophisticated branch prediction
- Multi-issue/superscalar
- Caching

The LC 3.5 ISA

The LC-3.5 is based on the LC3, which is defined in the second edition of the Patt&Patel text. The LC-3.5 eliminates several instructions (such as the LDI and STI) and replaced them more useful arithmetic instructions, such as SUB, OR, ASH (arithmetic shift), and LSH (logical shift). The table below shows the encodings for all of the LC-3.5's instructions; the salient features are described below:

- LSH and ASH use can accept both positive and negative values to determine how far to shift. Bit 15 of the second operand determine its sign and the low order 4 bits determine the shift magnitude. Bits 5-14 are ignored. A negative shift is a shift right.
- For shifting left, LSH and ASH are identical. For shifting right, ASH replicates the sign bit.
- JSR uses an 11-bit signed offset to compute the target PC (newPC = oldPC + offset)
- The starred instructions modify the condition code registers (NZP). Recall that only one of the N/Z/P bits is set at any one time.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*ADD	0	0	0	1	---DR---			--- SR1 ---			0	0	0	---SR2---		
*ADD	0	0	0	1	---DR---			--- SR1 ---			1	--------imm5--------				
*SUB	1	0	0	0	---DR---			--- SR1 ---			0	0	0	---SR2---		
*SUB	1	0	0	0	---DR---			--- SR1 ---			1	--------imm5--------				
*AND	0	1	0	1	---DR---			--- SR1 ---			0	0	0	---SR2---		
*AND	0	1	0	1	---DR---			--- SR1 ---			1	-------- imm5 --------				
*OR	1	1	0	1	---DR---			--- SR1 ---			0	0	0	---SR2---		
*OR	1	1	0	1	---DR---			--- SR1 ---			1	-------- imm5 --------				
*ASH	1	0	1	0	---DR---			--- SR1 ---			0	0	0	---SR2---		
*ASH	1	0	1	0	---DR---			--- SR1 ---			1	-------- imm5 --------				
*LSH	1	0	1	1	---DR---			--- SR1 ---			0	0	0	---SR2---		
*LSH	1	0	1	1	---DR---			--- SR1 ---			1	-------- imm5 --------				
*NOT	1	0	0	1	---DR---			--- SR ---			1	1	1	1	1	1
*LEA	1	1	1	0	---DR---			---------- PCoffset9 ----------								
*LD	0	0	1	0	---DR---			---------- PCoffset9 ----------								
*LDR	0	1	1	0	---DR---			--- BaseR ---	---------- offset6 ----------							
ST	0	0	1	1	---SR---			---------- PCoffset9 ----------								
STR	0	1	1	1	---SR---			--- BaseR ---	---------- offset6 ----------							
BR	0	0	0	0	n	z	p	---------- PCoffset9 ----------								
JMP	1	1	0	0	0	0	0	--- BaseR ---			0	0	0	1	0	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
JSR	0	1	0	0	1	---------- PCoffset11 ----------										
JSRR	0	1	0	0	0	0	0	--- BaseR ---	---------- offest6 ----------							
TRAP	1	1	1	1	0	0	0	0	---------- trapvect8----------							

LC-3 Datapath