Carnegie Mellon University
18-347 Introduction to Computer Architecture

# Project 1: Single Cycle MIPS

Due: The week of October 13, 2003 prior to the start of Lab
(100 points, may be done in groups of two)

## Objective

In this project, you will become intimately familiar with the MIPS processor and instruction set by implementing a complete single-cycle core in Verilog.

## Project Description

You will be given the skeleton of a single-cycle structural MIPS processor which is capable of performing the ADDI and the SYSCALL instructions. You will complete the design of the single-cycle implementation using the Verilog style implied by the skeleton. This implementation should be a full implementation of the MIPS347; it should run anything you could run on SPIM347 or on the instruction-level Verilog simulator (e.g., supports the instructions in the back cover of P&H).

If you have any questions about the exact behavior of the processor, refer to the behavior of spim347 or xspim347 on the ECE Solaris workstations. These simulators are considered the "golden" processor models for this course, and your design should emulate them as closely as possible. Performance is not a consideration for the single-cycle processor. Correctness counts for the most of the credit in this project.

You are free to use your ALU and 3-ported register file from labs 1 and 2, if you choose. Keep in mind that you will likely be starting subsequent projects from this code base, so it helps to have an implementation that you fully understand. Support for multiply and divide instructions are required for this project. Before you flood our mailboxes, remember that you are permitted (and encouraged) to use behavioral constructs for these operations. We will be supplying you with fully-functional structural models for these operations in later projects.

## Test Cases

For the demo, you will be expected to run a number of supplied programs (some revealed before the demo, some not). In any case, you'll want to build a suite of test programs to test the new capabilities of your implementation as you add them. We have no problem with people sharing these programs, as long as it is available to everyone. If you have an assembly language program that you've used to test a set of instructions, you can publish this program on the bboard. Please state the set of instructions this program requires.

## Diagram

As in the previous labs, you will also have to supply a computer-drawn diagram of your single-

cycle processor. You are encouraged to use decent vector drawing program to diagram your processor. All major structures (i.e., registers, muxes, incrementers) should be drawn, as well as boxes for various control logic blocks. You should label wires with their names and widths. For your sanity, we suggest using different colors (or line styles) to differentiate control and data path connections. We expect that you will base later diagrams off this one, so putting in extra effort to keep this diagram neat will pay off. Don't be afraid to make plenty of white space and span multiple sheets of paper!

## Handin

You should electronically handin all of your Verilog files through the course AFS space. Hand in a paper copy of your diagram at your Lab demo period. During the demo, we will ask your questions about your single-cycle design and test it with a number of input programs.

## Files

The project code that you are starting with is significantly more complicated than in the previous labs. A summary of the supplied modules is listed below:

mips347_struct.v: This file contains the skeleton of the MIPS ISA simulator. You will add constants, additional decode logic, the necessary registers, muxes, and control and data paths to support the full MIPS instruction set listed in the back of your book.

mips_mem.v: a dual-ported multi segmented memory module. Memory is divided into user data, user text, kernel data, kernel text, and stack space. Note that the spim347 program generates all five.dat files necessary to initialize this module. You should not have to change this file.

mips_mem_sync.v: similar to the above memory module, but synchronous. If you have problems using the asynchronous module, you may wish to use this. You should not have to change this file.

mips_defines.v: Verilog defines for various opcodes, instruction mnemonics, and other useful constants.

except.v: a Verilog exception handling unit. Prints errors if you encounter an exception. You should not have to change this file.

testjig_core.v: the file containing your top testjig module with the processor core and memory

All files are available in:

**`/afs/ece/class/ece347/public_html/LABS/project1_files/`**

The addiu.s file can test the supplied addiu and syscall instructions, as soon as you integrate the register file.

## Caveats

- The memory module, in file mips_mem.v, has been modified to provide four write enables per port. Each write enable corresponds to a byte in the memory. If you want to write a single byte in the memory, you must decode the address to enable only that byte during the write

operation.

- There are two modules in mips347_struct.v file that are not synthesizeable. They are the syscall_unit and the exception_unit. You should not have to modify these units. Every other unit is currently built in a synthesizeable way, and every other module should follow a similar style. You should not use any procedural/behavioral structures in those modules.
- Be aware that the multiply, divide, and modulus operators in Verilog (*, /, and %) are always unsigned. If you are performing a signed operation, you need to add some code. You can still use these operators, just take care.
- We provide one test program that currently works given the extremely limited instruction set. In order to assemble this program, you need to use the -notrap flag in spim347. This option prevents the code from including the standard exception handler and start-up code, which you cannot yet execute. Later when you have all instructions implemented, you should not have to use this flag. spim347 -vasm filename.s will generate the complete memory image sets for the memory module.

## Grading

- **Single-cycle MIPS**
  - Proper functionality 80 points
- **Block Diagram**
  - Complete block diagram 20 points
- **Total** **100 points**

## Late Project Policy

Late labs and projects will lose 10 points for each day following your assigned due date and time. The clock stops when all lab materials have been turned in (including Verilog code, diagrams, answers to questions, etc.) and all demos have been completed.