CPSC 321 Computer Architecture
Fall Semester 2004
Single Cycle MIPS–Lite Processor in Verilog
Due Date: Tuesday, 11-16-2004, at 23:59:59, via turnin.
Work in groups of up to two students

1 Objective

In this project you will build a single-cycle processor. You will need to verify that it executes a given subset of the MIPS instruction set.

2 Datapath Design

Implement the needed components in Verilog. You may use behavioral or a combination of behavioral and structural Verilog features. All Verilog models should correspond to realistic components, such as, registers, comparators, shifters, etc. *No super-composite components, eg, a branch unit that takes in the opcode, operands, and PC, and outputs a new PC, or something similar to that.* You will be given a number of basic datapath components which you *may* use or extend. All storage elements are negative-edge triggered.

2.1 Datapath Components

There are a few components you will need for your processor, including the following. We suggest you start by designing these sub-components:

ALU 32–bit ALU,

Extender sign/zero extender,

Multiplexers m16x2, m32x2, m32x3, m32x5, m5x2: multiplexers with various widths and number of inputs,

Register reg32: 32–bit register,

Register File regfile: register file,

Shifter Arithmetic logical, multi-bit shifting unit.

Instruction Memory 2K–words deep, 32–bit wide, (using "SRAM"), initial content will be loaded from within the Verilog code.

Data Memory 4K–words deep, 32–bit wide, no initial contents.

There should be an arithmetic/logical multi-bit (32) shifter *external* to the ALU. Furthermore, instructions such as `slt` should have effect outside the ALU, as specified in the textbook. `slt $rd, $rs, $rt` should subtract the two operands, and then generate the appropriate result value for the destination register. Your datapath should then store this result back in the destination register.

2.2 Provided Components

A number of datapath components are provided as examples of building larger behavioral constructs in Verilog. **You may use these components**. The following files are available, from the class website:

- `SingleCycleProc.v` A skeleton module for your processor. You need to fill in the missing parts of this file to complete the processors.

- `ALU_behav.v` A behavioral implementation of the ALU you would need for this project.

- `signextend.v` A sign extender.

- `lshift2.v` A left-shift-by-two module.

- `IdealMemory.v, DMeminit.v, IMeminit.v` A behavioral implementation of the ideal data and instruction memories, and files to initialize their contents. We will provide additional memory contents files later for evaluation and test purposes.

- `mux.v` Multiplexors of various sizes.

2.3 Component Propagations Delays

All of your Verilog components must incorporate "reasonable" propagation delays. For each component, think about how that component would be implemented using discrete gates. Use this mental discrete gate implementation to estimate the delays to use. There is no exact right or wrong answer for the delays to use. However, if the delays are too large or too small, you will be asked to justify your decision. The delay models may be kept simple, ie, one delay value can be used for an entire component (which should be the maximum worst-case delay). For the datapath controller you should use a delay of $20\Delta T$.

Please look through the Verilog code of the provided components for examples of how to specify propagation delays for the various components and how to support edge-triggered clocking. You should try to use reasonable delays for the blocks with storage elements.

2.4 Resetting and Initializing the Data-paths

Your data-paths must employ an active-low, master reset signal, called `Reset_L`. This signal should be kept at the de-asserted level at all times, except when the processor is being reset. To reset the

processor, drive `Reset_L` low for at least 10 integral clock cycles and then reset `Reset_L` to high level. The master reset should feed into all reset types of signals of the datapath, such as those of the register file. You have to provide a unit that is external to the main control units that contains at least one register called `IPC`. This register supplies the first Program Counter (PC) that your processor should execute after "reset". `IPC` and all other input control signals should be driven by the top level test module of your model.

3 Testing and Diagnostic Programs

You should write diagnostic programs to test your MIPS-Lite processor. These programs should exercise all instructions your datapath supports. Remember, a single cycle processor does not have delay slots in the `lw $rt, imm16($rs)` nor in the conditional branch instructions.

Your program should be written such that the result of the test is "displayed" on the STD-OUT. In general, you can display the internal state of a module by using appropriately coded `$display("...");` statements from within the module. For instance, you can use `$display("Reg[rt] = %b ...", Reg[rt])` statements in a module to print on the STDOUT the contents of a register "rt". It is important to write good diagnostic programs to test your hardware because it will simplify your life later on when you have to debug a more complicated processor.

4 Single-Cycle Data-Path

Develop an implementation in Verilog of a single-cycle processor for a subset of the MIPS architecture given below.

4.1 Supported Instructions

Build a single-cycle datapath that implements the following subset of MIPS instructions.

Type	Instructions
arithmetic (unsigned)	addu, subu, addiu
arithmetic (signed)	add, sub, addi
logical	and, andi, or, ori, xor, xori
shift	sll, sra, srl
compare	slt, slti, sltu
control	beq, bne, bgez, bltz, j, jr, jal
data transfer	lw, sw, lui

Make sure to verify the coding of instructions such as `bgez` in Appendix A. Note that the "rt" field is actually used to distinguish instructions `bgez` and `bltz`. Use the actual MIPS machine language instruction formats, given in the back cover of your textbook.

4.2　Verilog Controller for the Datapath

Build the Control Unit (CU) for the single-cycle datapath. You may use any behavioral Verilog instruction available to develop the CU. This CU should take the exact bits from the 32-bit instruction and generate the control signals. You must add any control signals necessary to read instructions off the instruction memory, and the "next instruction logic" to generate the Program Counter (PC) of the instruction which needs to be fetched next.

The Verilog code that specifies the behavior of your CU must be well structured. Use the Verilog `'define SYMBOL code_to_substitute` as much as possible and specify upper bounds for units as `parameter n = ....` **Note**: Recall that in the Single-Cycle Processor, everything should take place in one clock cycle T_{CC}. Make sure that you are using sufficiently (but not excessively) long clock period T_{CC}, for your Single-Cycle Processor. *Turn in a copy of the output matrix for the control unit, processor schematic, diagnostic program(s), and a simulation log that shows the correct execution of the test MIPS programs.*

You will have to include detailed justification for the selection of T_{CC} in your Single-Cycle Processor. You need to be prepared to demonstrate evidence for this justification, in terms of longest combinational circuit paths in your design. Hint: use Verilog to measure the times it takes to compute signal propagations within your data-path.

5　Deliverables

Design Notebook (30pts)
Please keep a design notebook that documents your design process and all the steps in the program development; it should document the progress that you made while writing your code (keep track of date and time). Write the design notebook while you are developing the software, do not write it after the fact.

Your notebook should keep a log of all the errors that you make; it should be handwritten with a pen that is not erasable; printouts documenting your progress should be glued in. The grading of your design notebook will take into account the clarity, regularity, legibility and organization of your annotations.

You also need to turnin the control matrix for the control unit.

Verilog Program (40pts)
Turn in your verilog program via the *turnin* command. Ensure that your program is clearly commented.

Project Demonstration (30pts)
Project demonstrations should be completed within a week after submission. Failure to demonstrate will result in a Fail grade on the project.

6   Dishonesty

Make sure that you complete the assignment by yourself. Do not copy the code from others, nor provide others with your code. Refrain from copying and modifying the code from other sources.