Computer Architecture
CPSC 321, Fall Semester 2003
Project #1
Due: Monday, November 17, 2003 until 23:59:59
Complete in teams of up to two students

1 Objectives

In this project you are supposed to implement a single cycle processor for a given subset of the MIPS instruction set using the hardware description language Verilog. This processor includes a datapath and a control unit, and is similar to the one discussed in [1, Chapter 5]. You can take advantage of behavioral Verilog features, but you are also allowed to use structural designs. You have to demonstrate the correctness of your design by successful runs of various MIPS test programs.

2 Datapath Design

You may use any combination of behavioral and structural Verilog features; however, your Verilog models should correspond to realistic components, such as registers, comparators, shifters, etc. In particular, you should not use super-composite components, such as a branch unit that takes in the opcode, operands, and PC, and outputs a new PC, or the like. We provide a number of basic datapath components which you can use or extend. You do not have to use these components, but they illustrate what we have in mind. **Use negative-edge triggered storage elements throughout your design.**

Datapath Components. There are a few components that you will need for your processor, including those from the list given below. We suggest you start by designing these sub-components:

ALU 32–bit ALU,

Extender sign/zero extender,

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.

Provided Components. A number of datapath components are provided as examples. **You do not have to use any of these components. Feel free to modify or replace them if necessary**. The following files are available:

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

Finish the implementation of the processor by supplying or completing all necessary modules.

3 Propagations Delays

The single datapath must incorporate reasonable propagation delay amounts for each component. You need to specify propagation delays for both combinational and sequential circuits. Finally, you have to find and use the *minimum possible* clock periods T_{CC} in your single datapaths.

Component Propagations Delays. All of your Verilog components must, in general, incorporate "reasonable" propagation delays. For each component, think about how that component would be implemented using discrete gates, and use this model to estimate the delays to use. There is no exact right or wrong answer for the number of 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, that is, one delay value can be used for an entire component, but this value should be reasonably close to the worst case delay. Use a delay of $20\Delta T$ for the datapath controller. You may use the following normalized delays for individual gates:

Gate	Delay in ΔT
NOT	1
NOR	2
NAND	2
OR	3
AND	3
XOR	2

The provided Verilog components illustrate how to specify propagation delays and how to support edge-triggered clocking. You should try to use reasonable delays for the blocks with storage elements.

Resetting and Initializing the datapaths. Your datapaths 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 T_{CC} 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.

4 Testing and Diagnostic Programs

You should write diagnostic programs to test your MIPS-Lite processor. These programs **should exercise all instructions** that 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 printed to `stdout`. For instance, you can display the internal state of a module using `$display` statements. Recall that you can use statements such as `$display("Reg[rt] = \%b ...", Reg[rt])` to print the contents of a register "`rt`" to `stdout`. It is important to write good diagnostic programs because it will ease the debugging process.

5 Single Cycle Datapath [100 Points]

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

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 of [1]. 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 [1].

Control Unit. Build the control unit for the single cycle datapath of Section 5. You may use any behavioral Verilog instruction available to develop the control unit. The control unit 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 content of the program counter (PC) of the instruction to be fetched next.

The Verilog code that specifies the behavior of your control unit must be well-structured and clear. Use defines and parameters, such as `` `define SYMBOL somecode `` and `parameter n = ...`, to give meaningful names to states, range boundaries, and other constants. Avoid the use of unmotivated constants.

Note: Recall that in the single cycle Processor, everything should take place in one clock cycle T_{CC}. 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.

Hint: Use Verilog to measure the signal propagations delays within your datapath.

Checklist

- ☐ Carefully documented, fully functional Verilog implementation of the single cycle processor, including diagnostic programs for all components.
- ☐ A written report containing a brief justification of the component delays, and a detailed justification for the selection of T_{CC}. Provide evidence in terms of the critical path, that is, the longest combinational circuit path of your design.
- ☐ An output matrix for the control unit.
- ☐ Schematics of your processor.
- ☐ Include in your report a list of know bugs (that is hopefully empty).
- ☐ Demonstrate your program, following the instructions of your TA.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 2nd ed. Morgan-Kaufmann Publishers Inc., 1998, iSBN 1-55860-428-6.

[2] S. Brown and Z. G. Vranesic, *Fundamentals of Digital Logic With Verilog Design.* The McGraw-Hill Co., 2003, [Good logic design text using Verilog.].