RTL Model of a Two-Stage MIPS Processor

6.884 Laboratory 1
February 4, 2005 - Version 20040215

1 Introduction

For the first lab assignment, you are to write an RTL model of a two-stage pipelined MIPS processor using Verilog. The lab assignment is due at the start of class on Friday, February 18. You are free to discuss the design with others in the class, but you must turn in your own solution.

The two-stage pipeline should perform instruction fetch in the first stage, while the second pipeline stage should do everything else including data memory access. The 32-bit instruction register should be the only connection from the first stage to the second stage of the pipeline. You should find that the two-stage pipeline makes it easy to implement the MIPS branch delay slot.

If you need to refresh your memory about pipelining and the MIPS instruction set, we recommend "Computer Organization and Design: The Hardware/Software Interface", Second Edition, by Patterson and Hennessey.

For this assignment, you should focus on writing clean synthesizable code that follows the coding guidelines discussed in lecture. In particular, place logic only in leaf modules and use pure structural code to connect the leaf modules in a hierarchy. Avoid tricky hardware optimizations at this stage, but make sure to separate out datapath and memory components from control circuitry.

The datapath diagram in Figure 5 can be used as an intial template for your SMIPS cpu implementation, but please treat it as a suggestion. Your objective in this lab is to implement the SMIPS ISA subset, not to implement the datapath diagram so feel free to add new control signals, merge modules, or make any other modification to the datapath diagram.

2 CPU Interface

Your processor model should be in a module named `mips_cpu`, and must have the interface shown in Figure 1. We will provide a test rig that will drive the inputs and check the outputs of your design, and that will also provide the data and instruction memory. We have provided separate instruction and data memory ports to simplify the construction of

the two stage pipeline, but both ports access the same memory space. The memory ports can only access 32-bit words, and so the lowest two bits of the addresses are ignored (i.e., only `addr[31:2]` and `iaddr[31:2]` are significant). Notice that the data write bus is a separate unidirectional bus from the data read bus. Bidirectional tri-state buses are usually avoided on chip in ASIC designs.

```
module mips_cpu
(
   input clk,                    // Clock input
   input reset,                  // Reset input
   input int_ext,                // External interrupt input

   input  [7:0]  fromhost,       // Value from test rig
   output [7:0]  tohost,         // Output to test rig

   output [31:0] addr,           // Data memory address
   output        wen,            // Data memory write enable
   output [31:0] write_data,     // Data to write to memory
   input  [31:0] read_data,      // Data read back from memory

   output [31:0] iaddr,          // Instruction address
   input  [31:0] inst            // Instruction bits
);
```

Figure 1: Interface to SMIPS CPU.

3 Implemented Instructions

The SMIS instruction set is a simplified version of the full MIPS instruction set. Consult the "SMIPS Processor Specification" for more details about the SMIPS architecture. For this first lab assignment, you will only be implementing a subset of the SMIPS specification. Figures 2 and 3 show the instructions that you must support.

For this first assignment there are only 35 distinct instructions to implement. The instructions we have removed from the SMIPS specification for this lab are: byte and halfword loads and stores, all multiply and divide instructions (you do not need to implement the `hi` and `lo` registers), the branch likely instructions, the branch and link instructions (BLTZAL, BGEZAL), the instructions that can cause arithmetic overflows (ADD, SUB, ADDI), and other instructions related to trap handling (SYSCALL, BREAK).

You do not need to support any exceptions or interrupt handling (apart from reset). The only piece of the system coprocessor 0 you have to implement are the `tohost` and `fromhost` registers, and the MTC0 and MFC0 instructions that access these registers. These registers are used to communicate with the test rig. The test rig drives `fromhost`, while you should implement an 8-bit register in COP0 which drives the `tohost[7:0]` port on the `mips_cpu` module interface.

Opcode

31...29 \ 28...26	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	*	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*
4	*	*	*	LW	*	*	*	*
5	*	*	*	SW	*	*	*	*
6	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*

SPECIAL function

5...3 \ 2...0	0	1	2	3	4	5	6	7
0	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1	JR	JALR	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*
4	*	ADDU	*	SUBU	AND	OR	XOR	NOR
5	*	*	SLT	SLTU	*	*	*	*
6	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*

REGIMM rt

20...19 \ 18...16	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	*	*	*	*	*	*
1	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*

Figure 2: SMIPS CPU Instruction Subset for Lab 1.

COP0 rs

25...24 \ 23...21	0	1	2	3	4	5	6	7
0	MFC0	ξ	ξ	ξ	MTC0	ξ	ξ	ξ
1	ξ	ξ	ξ	ξ	ξ	ξ	ξ	ξ
2	CO0							
3								

Figure 3: SMIPS CP0 Instruction Subset for Lab 1.

4 Test Rig

We are providing a test rig to connect to your CPU model. The test rig loads in a hex memory dump of instructions to fill the memory. You should use the smips-gcc toolchain to build verilog memory dump versions of your SMIPS assembly test programs. The test rig will clock the simulation until it sees a non-zero value coming back on the `tohost` register, signifying that your CPU has completed a test program.

The simplest test program is shown in Figure 4.

```
# 0x1000: Reset vector.
        addiu r2, r0, 1     # Load constant 1 into register r2
        mtc0 r2, r21        # Write tohost register in COP0
loop:   beq r0, r0, loop    # Loop forever
        nop                 # Branch delay slot
```

Figure 4: Simple test program.

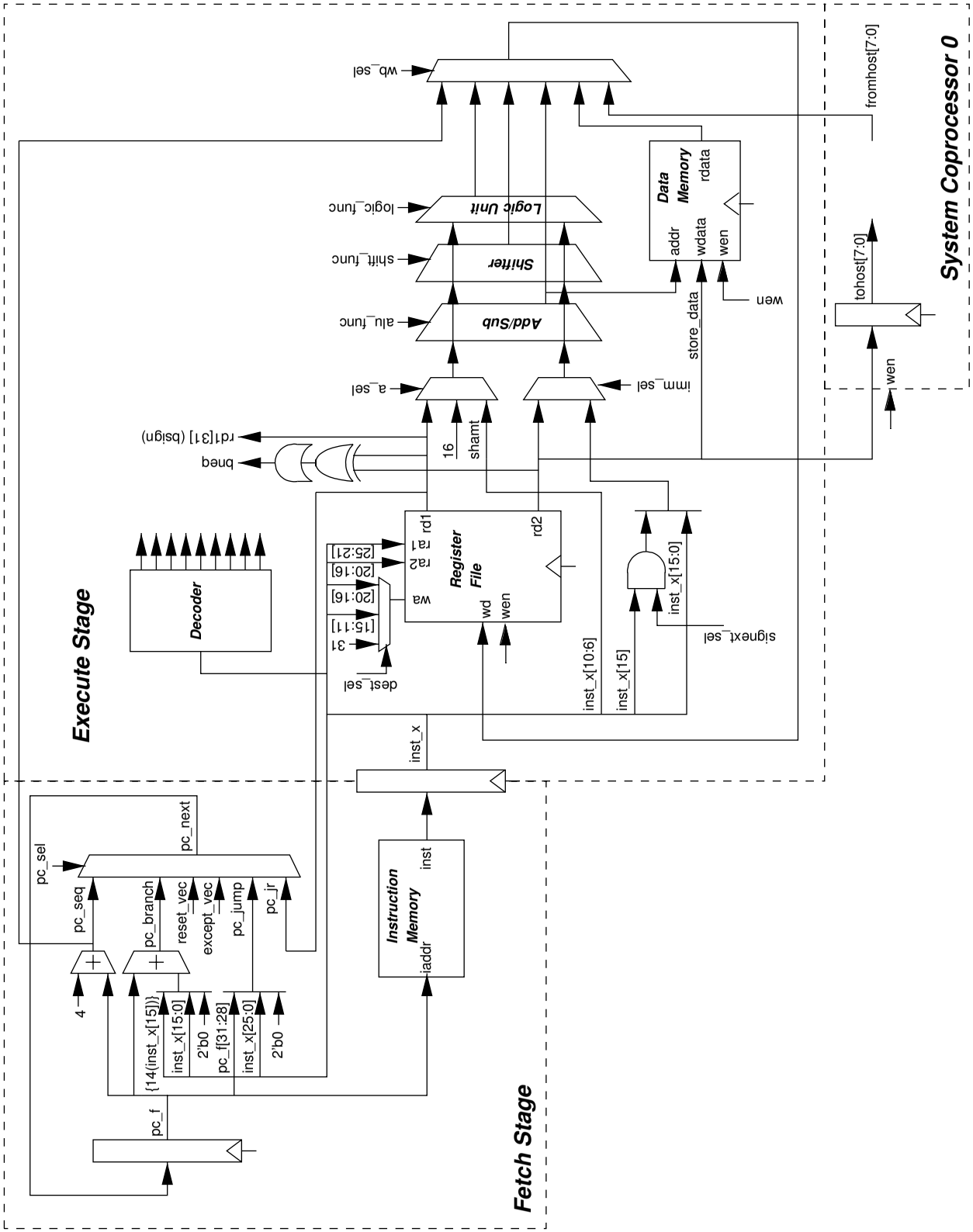

Figure 5: SMIPS 2-Stage Pipeline Datapath for Lab 1.

31–26	25–21	20–16	15–11	10–6	5–0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type
Load and Store Instructions						
100011	base	dest	signed offset			LW rt, offset(rs)
101011	base	dest	signed offset			SW rt, offset(rs)
I-Type Computational Instructions						
001001	src	dest	signed immediate			ADDIU rt, rs, signed-imm.
001010	src	dest	signed immediate			SLTI rt, rs, signed-imm.
001011	src	dest	signed immediate			SLTIU rt, rs, signed-imm.
001100	src	dest	zero-ext. immediate			ANDI rt, rs, zero-ext-imm.
001101	src	dest	zero-ext. immediate			ORI rt, rs, zero-ext-imm.
001110	src	dest	zero-ext. immediate			XORI rt, rs, zero-ext-imm.
001111	00000	dest	zero-ext. immediate			LUI rt, zero-ext-imm.
R-Type Computational Instructions						
000000	00000	src	dest	shamt	000000	SLL rd, rt, shamt
000000	00000	src	dest	shamt	000010	SRL rd, rt, shamt
000000	00000	src	dest	shamt	000011	SRA rd, rt, shamt
000000	rshamt	src	dest	00000	000100	SLLV rd, rt, rs
000000	rshamt	src	dest	00000	000110	SRLV rd, rt, rs
000000	rshamt	src	dest	00000	000111	SRAV rd, rt, rs
000000	src1	src2	dest	00000	100001	ADDU rd, rs, rt
000000	src1	src2	dest	00000	100011	SUBU rd, rs, rt
000000	src1	src2	dest	00000	100100	AND rd, rs, rt
000000	src1	src2	dest	00000	100101	OR rd, rs, rt
000000	src1	src2	dest	00000	100110	XOR rd, rs, rt
000000	src1	src2	dest	00000	100111	NOR rd, rs, rt
000000	src1	src2	dest	00000	101010	SLT rd, rs, rt
000000	src1	src2	dest	00000	101011	SLTU rd, rs, rt
Jump and Branch Instructions						
000010	target					J target
000011	target					JAL target
000000	src	00000	00000	00000	001000	JR rs
000000	src	00000	dest	00000	001001	JALR rd, rs
000100	src1	src2	signed offset			BEQ rs, rt, offset
000101	src1	src2	signed offset			BNE rs, rt, offset
000110	src	00000	signed offset			BLEZ rs, offset
000111	src	00000	signed offset			BGTZ rs, offset
000001	src	00000	signed offset			BLTZ rs, offset
000001	src	00001	signed offset			BGEZ rs, offset
System Coprocessor (COP0) Instructions						
010000	00000	dest	cop0src	00000	000000	MFC0 rt, rd
010000	00100	src	cop0dest	00000	000000	MTC0 rt, rd

Table 1: SMIPS instruction subset for Lab 1.