CoE 113 Lab Exercise 5

May 2, 2017

1 Introduction

This lab exercise seeks to implement a pipelined processor implementation of a subset of the MIPS instruction set that is capable of dealing with RAW data hazards.

2 Specifications

The processor implementation should be compliant to the MIPS instruction set, but any exceptions to the programming model will be specified in the specifications. We will also follow the MIPS convention of having register 0 in the register file as a constant 0 which cannot be written. All other registers can be written to. This section defines the core instructions supported in stage 1, the interface of the processor, and the memory models that will be used in simulation.

2.1 Instruction Set

Table 1 shows the instructions that must be supported by the processor. The rs, rt, and rd fields correspond to register addresses (for the register file), while the imm field, which is 16-bits long, correspond to a 16-bit constant which is encoded in signed 2's-complement notation. For details on the execution of each of the first 7 instructions (from a programming point of view), refer to lab exercise 3 and the class web page.

Jump instructions (J, JAL, and JR) branch unconditionally to a certain instruction memory address. For the J instruction, the PC is changed to a value of {PC[31:28], imm[25:0], 2'b00} instead of the normal PC+4. JAL is identical to J, but it also writes the return address (instruction after JAL) to register 31. The JR instruction replaces the PC with the value stored in register rs.

Instruction	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
ADD	000000	rs	rt	rd	XXXXX	100000
SUB	000000	rs	rt	rd	XXXXX	100010
SLT	000000	rs	rt	rd	XXXXX	101010
ADDI	001000	rs	rt	imm[15:11]	imm[10:6]	imm[5:0]
SLTI	001010	rs	rt	imm[15:11]	imm[10:6]	imm[5:0]
LW	100011	rs	rt	imm[15:11]	imm[10:6]	imm[5:0]
SW	101011	rs	rt	imm[15:11]	imm[10:6]	imm[5:0]
BEQ	000100	rs	rt	imm[15:11]	imm[10:6]	imm[5:0]
BNE	000101	rs	rt	imm[15:11]	imm[10:6]	imm[5:0]
J	000010	imm[25:21]	imm[20:16]	imm[15:11]	imm[10:6]	imm[5:0]
JAL	000011	imm[25:21]	imm[20:16]	imm[15:11]	imm[10:6]	imm[5:0]
JR	000000	rs	XXXXX	XXXXX	XXXXX	001000

Table 1: Core instruction set

In cases where the instruction encoding does not match any of the supported instructions, the processor simply performs a NOP (No Operation) and does not change the processor state (no writes to register file and memory), except for the PC which is incremented normallly.

2.2 Processor Interface

Table 2 lists the external interface of the processor. A global clock signal, clk, will govern all of the operations in the processor. An active-low reset signal, rst_n, will also be present. Setting rst_n to 0 will reset the PC and all registers in the register file to 0. The protocol and timing of these signals should follow specifications set in lab exercise 4. You are also required to set the module name to *pipelined_ mips*, for automated checking purposes.

Name	Direction	Width
clk	input	1
rst_n	input	1
inst_addr	output	32
inst	input	32
data_addr	output	32
data_in	input	32
data_out	output	32
data_wr	output	1
pc_IF	output	32
pc_ID	output	32
pc_EXE	output	32
pc_MEM	output	32
pc_WB	output	32

Table 2: Processor Ports

2.3 Memory Model

Both instruction and data memories will have synchronous read paths. Once the address is issued, its corresponding contents are displayed immediately after next rising edge of the clock. All writes are registered. The memory element will be updated upon the next clock edge after issuing the write (by asserting the write enable signal). The memory model will be byte-addressable, but will have word-length read and write ports. Memory organization is assumed to be big-endian. Whenever a byte-address X is issued, the correponding contents of address X, X+1, X+2, and X+3 are placed in the read port of the memory, with the first byte (address X) as the leftmost byte (MSB). For writes, bits [31:24] of the write port are written to address X, [23:16] to address X+1, [15:8] to address X+2, and [7:0] to address X+3. Take note that for the instruction memory, we will only have read ports.

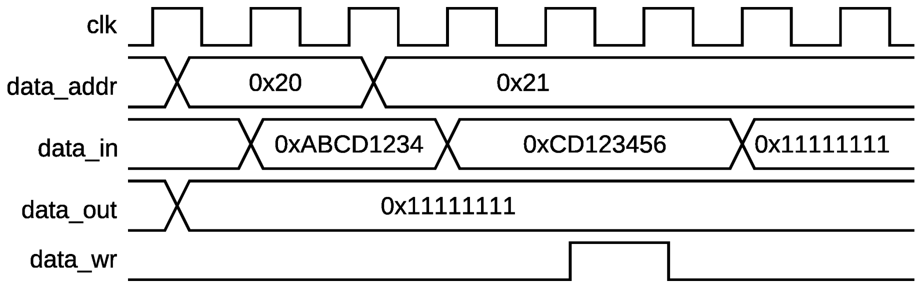

Figure 1: Memory interface timing

Figure 1 shows an example of a read and write operation performed on the memory model. It assumes that the following bytes are initially stored in memory: 0xAB at address 0x20, 0xCD at address 0x21, 0x12 at address 0x22, 0x34 at address 0x23, and 0x56 at address 0x24. The first read is word-aligned, while the second read and the write is not word-aligned. After the write, the state of the memory is: 0xAB at address 0x20, 0x11 at address 0x21, 0x11 at address 0x22, 0x11 at address 0x23, and 0x11 at address 0x24.

2.4 Pipelined operation

You are required to implement the processor in 5 pipeline stages. Functionality should follow specifications set in lab exercise 4.

Figure 2 shows an example of two instructions (SW and ADDI) being executed in the pipeline. Their individual addresses in memory are pre-appended. Only clk, inst_addr, inst, data_wr, PC_exe, and PC_mem are required signals, as discussed earlier. The other signals are "internal" signals which you may choose to implement or not.

Execution starts with IF, where the inst_addr is set to the address of the instruction being fetched. One clock cycle after, the instruction is now at the memory output (inst) and can already be decoded. Being at the

ID stage, this is reflected by RF addr A (register file first source operand) being set to the appropriate value rs of the instruction.

After another clock cycle, the instruction must now be in the EXE stage. In the example, this is shown by the appropriate operation's result being found at the output of the ALU (ALU result). The instruction then proceeds to the next stage, MEM, where memory writes are done by the SW instruction as shown by the assertion of the data_wr signal.

Lastly, writes to the register file are done at the WB by the ADDI instruction as shown by the assertion of the register file write enable signal (RF wr_en). Only after this clock cycle should the result be found in the register file.

The PC_exe and PC_mem signals show at which stage each instruction is in the pipeline. For example, PC_exe has a value of 16 at the clock cycle when the ALU result is the address computation of the memory address for SW. This means that during that clock cycle, the SW instruction is at the EXE stage.

```
16 SW   0(R4),R1
20 ADDI R2,R0,10
```

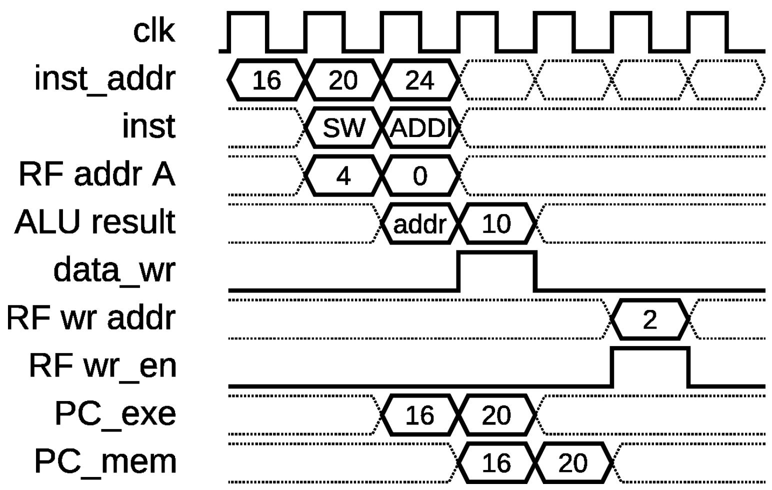

Figure 2: Example pipelined execution

2.5 RAW data hazard handling

The processor should be able to handle RAW data hazards that arise from dependent instructions. At the very least, the processor should stall a dependent instruction at the ID stage until the instruction that produces its result finishes its WB stage. Due to the synchronous nature of memory access, 3 stall cycles will be inserted when consecutive instructions have a RAW hazard, as shown in figure 3. Only 2 stall cycles are inserted for the last ADD instruction in the example since the instruction it is dependent to has another instruction before the last ADD.

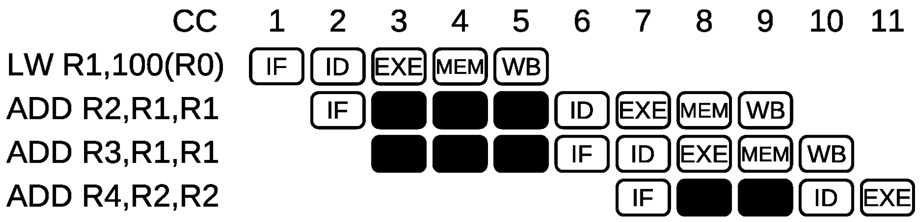

Figure 3: RAW hazard stalls

Full data forwarding implementations completely eliminate stalls from RAW hazards arising from non-load instructions. The previous example now executes in the manner shown in figure 4. Stall cycles for the last ADD instruction are completely eliminated by a forwarding path from the EXE stage pipeline register to the EXE stage. Take note that RAW hazards arising from load instructions cannot be settled by using forwarding since the operand has to go through the MEM stage. For purposes of checking, forwarding will only be verified for non-load instructions. Any RAW hazards resulting from load instructions must be dealt with by inserting stall cycles.

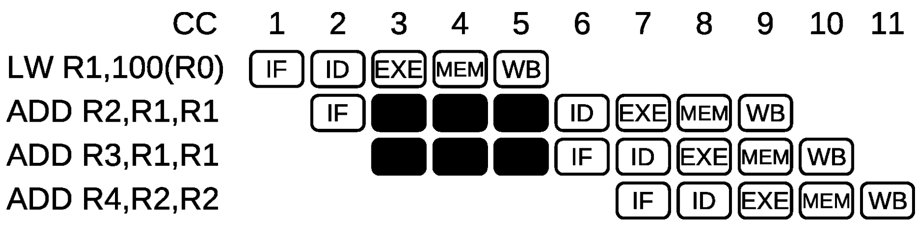

Figure 4: RAW hazard forwarding

2.6 Synthesis and timing constraints

The processor should be synthesized using the generic 90nm standard cell library provided in class. The synthesized designs should operate at a clock period of 20ns. The maximum input delay for all input pins should be set to 25% of the clock period. The maximum output delay for all output pins should be set to 25% of the clock period. Also, make sure to name your mapped Verilog file as *pipelined_mips_mapped.v* and your SDF file as *pipelined_mips_mapped.sdf* (for automated checking purposes).

3 Submission

Grading will be broken down as follows:

- Behavioral Model (25%)
  - 5% LW, SW
  - 5% ADD, SUB, SLT (requires LW, SW)
  - 5% ADDI, SLTI (requires LW, SW)
  - 5% BEQ, BNE (requires LW, SW)
  - 5% J, JAL, JR (requires LW, SW)
- Synthesized Model (25%)
  - 5% LW, SW
  - 5% ADD, SUB, SLT (requires LW, SW)
  - 5% ADDI, SLTI (requires LW, SW)
  - 5% BEQ, BNE (requires LW, SW)
  - 5% J, JAL, JR (requires LW, SW)
- RAW hazard stalling (30%)
  - 15% Behavioral model
  - 15% Synthesized model
- RAW hazard forwarding (20%)
  - 10% Behavioral model
  - 10% Synthesized model

All other instructions will only be graded if both LW and SW instructions are fully functional.

Deadline for submission of this lab exercise will be on May 25, 11:59PM. The processor will be verified using an automated checker testbench. Make sure that your codes are working before having them checked. Submit your codes within in the deadline date by e-mail to chris.densing@eee.upd.edu.ph, with the e-mail subject set to "CoE 113 lab 5 submission". Send all source Verilog files, the synthesized (mapped) Verilog file, and the synthesized (mapped) SDF file archived in single ZIP, TAR or TAR.GZ file, with filename in the format [SURNAME]_[FIRSTNAME] (ex. densing_chrisvincent.tar). Make sure that all the files within the archive are located in a flat structure, with no subdirectories/folders containing the required files. Appropriate demerits will be given to those who fail to follow instructions.