CoE 113 Lab Exercise 4

March 21, 2017

1 Introduction

This lab exercise seeks to implement a pipelined processor implementation of a subset of the MIPS instruction set consisting only of 12 instructions: ADD, SUB, SLT, ADDI, SLTI, LW, SW, BEQ, BNE, J, JAL, and JR.

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 9 core instructions that must be supported in stage 1 of the MP. 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.

Instruction and data memory will be separated into different blocks. The interface to the instruction memory will only consist of the instruction addr (inst_addr), which is computed by the processor, and the actual instruction fetched from memory (inst). This is primarily because access to the instruction memory is always just a read, and never a write.

In contrast, the interface to the data memory requires two-way communication. The address of the memory element to be read is specified in the data_addr port. The corresponding contents associated with that address are placed by the memory at the data_in port of the processor. To write to the memory, data to be written is placed by the processor at the data_out port of the processor, and the data_wr signal is set to 1. If the data_wr signal is set to 0, no data should be written and the state of the memory should be preserved.

Debug signals for verifying pipelined operation are also added to the external interface. These signals (pc_IF, pc_ID, pc_EXE, pc_MEM, pc_WB) specify the address of the instruction currently executing in the corresponding stage of the pipeline. (functionally, pc_IF should be the same as inst_addr, but we add it nonetheless for uniformity).

In cases where an instruction is flushed, just let the values of these debug signals proceed through the pipeline unchanged, even though the instruction will never complete execution because it was flushed.

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.

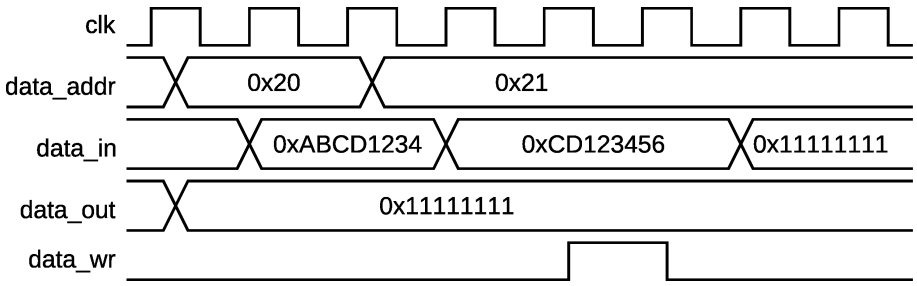

Figure 1: Memory interface timing

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. The following subsections will discuss the functionality of each stage.

2.4.1 Instruction fetch (IF)

The IF stage is the first stage of the pipeline. It is responsible for generating the address of the instruction to be fetched from instruction memory. Take note that unlike the single-cycle implementation of the previous lab exercise, the instruction is provided by memory on the rising edge of the clock (in the previous lab exercise, the instruction is provided combinationally, in the same clock cycle). The address of the instruction currently associated with the IF stage should be output on the pc_IF signal (which is technically equal to inst_addr).

2.4.2 Instruction decode (ID)

The ID stage is the second stage of the pipeline. It is responsible for generating the needed operands by the ALU in the next stage (EXE). Operands may either be from the register file or generated from the immediate field of certain instructions. Register file specifications are identical to those found in lab exercise 2. To keep things simple, this exercise will not impelement hazard detection. It is assumed that the program provided is scheduled appropriately such that data hazards are avoided. Details on spacing appropriate spacing of instructions to avoid data hazards will be discussed in the WB stage subsection. The address of the instruction currently associated with the ID stage should be output on the pc_ID signal.

Since the instruction can already be decoded at this stage, any jump instruction (J, JAL, JR) should also be resolved now. A jump instruction is unconditional (always taken), hence resulting in a control hazard with a penalty of 1 clock cycle, as shown in figure 2.

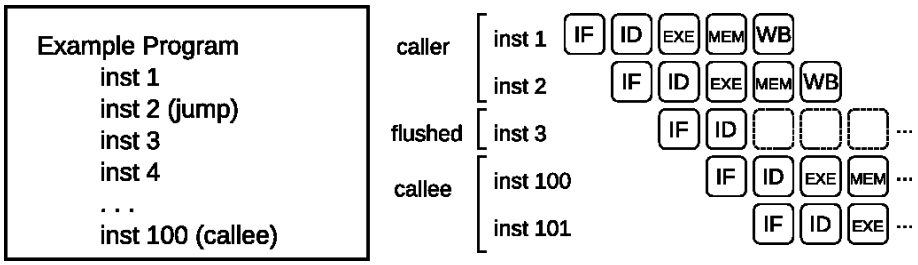

Figure 2: Jump control hazard behavior

2.4.3 Execute (EXE)

The EXE stage is the third stage of the pipeline. All results of arithmetic operations are computed in this stage. Branch target computation is also done in this stage. Aside from these, branch evaluation is performed in the EXE stage. Control signals for correctly updating the PC in the IF stage are generated combinationally in this stage of the pipeline. Therefore, evaluating a taken branch will result in a control hazard, resulting in a 2 clock cycle penalty effectively flushing the two instructions fetched after the branch as shown in figure 3. The address of the instruction currently associated with the EXE stage should be output on the pc_EXE signal.

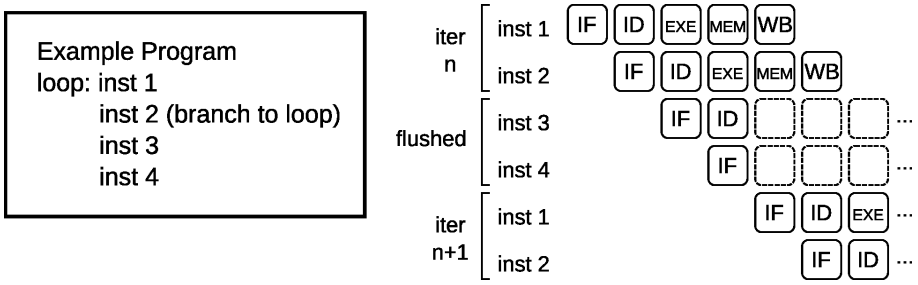

Figure 3: Branch control hazard behavior (after taking the branch)

2.4.4 Memory (MEM)

The MEM stage is the fourth stage of the pipeline. All memory accesses are done in this stage. Appropriate control signals for reading or writing memory are generated here. For memory reads (loads), data from memory will only be made available in the next stage (WB) because of the synchronous nature of reads in data memory (therefore, the WB stage is necessary). The address of the instruction currently associated with the MEM stage should be output on the pc_MEM signal.

2.4.5 Write Back (WB)

The WB stage is the last stage of the pipeline. Necessary control signals for writing to the register file are generated in this stage. This means that instructions which are dependent on the result being written back in the current clock cycle must read their operands (ID stage) in the next clock cycle, as shown in figure 4. The address of the instruction currently associated with the WB stage should be output on the pc_WB signal.

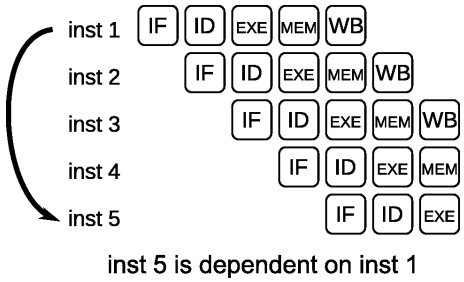

Figure 4: Spacing of dependent instructions

Aside from R-type and I-type arithmetic instructions, the JAL instruction also writes to the register file (register 31) at this stage of the pipeline.

2.4.6 Example Execution

Figure 5 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
```

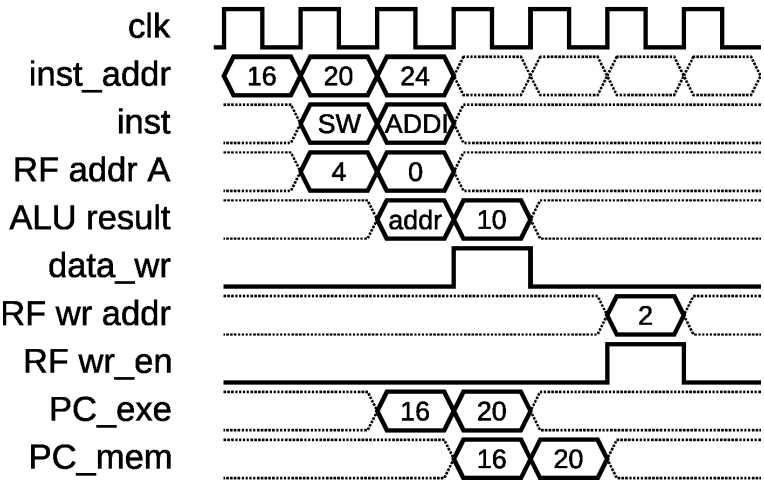

Figure 5: Example pipelined execution

2.5 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 (65%)
  - 30% in-class demonstration
  - 10% 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)
  - 5% Special application (all instructions required)
- Synthesized Model (35%)
  - 10% 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)
  - 5% Special application (all instructions required)

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

Deadline for checking of this lab exercise will be on April 4, 5, and 6 for the Tuesday, Wednesday, and Thursday lab class. The in-class demonstration part of the grade will be done on the deadline. You will be instructed on the deadline date on what instructions to run on your machine in order to successfully demonstrate operation. The other parts 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 snapdensing@gmail.com, with the e-mail subject set to "CoE 113 lab 4 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.