COMPARC Milestone 2

January 28, 2017

1 Introduction

This lab exercise seeks to implement a simple single-cycle processor implementation of a subset of the MIPS instruction set consisting only of 7 instructions: ADD, LW, SW, BEQ, BNE, SUB, and SLT. There is no need to implement the MIPS pipeline yet.

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. Each instruction of the processor should complete in exactly one clock cycle. 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 4 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.

Instruction	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
ADD	000000	rs	rt	rd	xxxxx	100000
LW	100011	rs	rt	imm[15:ll]	imm[10:6]	imm[5:0]
SW	101011	rs	rt	imm[15:ll]	imm[10:6]	imm[5:0]
BEQ	000100	rs	rt	imm[15:ll]	imm[10:6]	imm[5:0]
BNE	000101	rs	rt	imm[15:ll]	imm[10:6]	imm[5:0]
SUB	000000	rs	rt	rd	xxxxx	100010
SLT	000000	rs	rt	rd	xxxxx	101010

Table 1: Core instruction set

The **ADD instruction** adds the contents of rs and rt and stores the result to rd. Arithmetic overflow need not be considered (i.e. instruction execution should proceed normally even with overflow). Bits [10:6] of the instruction encoding are ignored, and may take on any value for the ADD instruction. The **SUB instruction** is an R-Type arithmetic instruction, substracts contents of rt from contents of rs and stores it to register specified by rd ([rs] – [rt] = [rd]). The **SLT instruction** set contents of rd to 32'd1, if rs is less than rt. Otherwise, set contents of rd to 32'd0.

The **LW instruction** computes for the address in memory of the item to be loaded by adding the base address and the offset. The register containing the base address is rs, while the offset is encoded in the imm field. The offset encoded in the imm field is a byte address. All memories interfaced to the processor are assumed to be big-endian and byte-addressable. For example, we want to load the constant 0xABCD1234 and its byte address specified in the LW instruction is 0x20, then the location of each byte of the constant in memory is as follows: OxAB at address 0x20, OxCD at address 0x21, 0x12 at address 0x22, and 0x34 at address 0x23. After computing for the address, the corresponding data, which is 4 bytes long (1 word), is fetched from memory and is saved to register rt.

The **SW instruction** is similar to the LW instruction in terms of computing for the memory address. The only difference is that instead of fetching from data memory and storing to register rt, data stored in the register rt is written to the data memory, with the address specified by the computed memory address. Data is also assumed to be 4 bytes long, and should be consistent with the endian-ness described in the LW instruction specification.

The **BEQ instruction** compares the contents of rs and rt. If the contents are not equal, program flow proceeds normally. Otherwise, the next instruction executed is the instruction with address PC + 4 + branch offset, where PC is the address of the currently executing BEQ instruction. Branch offset is specified in the immediate field of the instruction encoding. However, unlike the LW and SW instructions, the immediate field in the BEQ instruction now specifies a word address instead of a byte address. Take note that all memories to be used are byte-addressable, thus you may need to perform additional operations to consider this. The **BNE instruction** is essentially the same as BEQ, except that the branch is taken when the contents of rs and rt are not equal.

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, elk, 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.

You are also required to set the module name to ***single_cyde_mips,*** for automated checking purposes.

Name	Direction	Width
elk	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

Table 2: Processor Ports

2.3 Memory Model

Both instruction and data memories will have combinational read paths. Once the address is issued, its corresponding contents are displayed immediately, without waiting for a clock edge. On the other hand, 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 corresponding contents of address X, X+l, 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+l, [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.

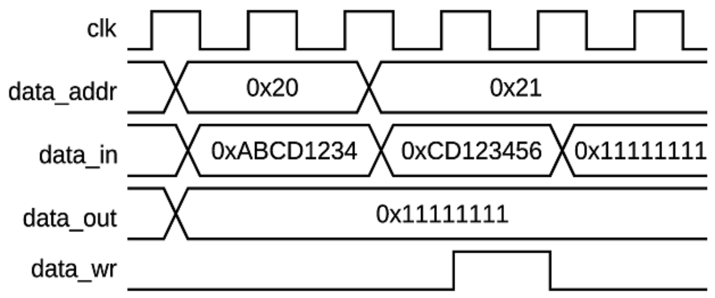

Figure 1: Memory interface timing

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: OxAB at address 0x20, OxCD 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: OxAB at address 0x20, 0x11 at address 0x21, 0x11 at address 0x22, 0x11 at address 0x23, and 0x11 at address 0x24.

2.4 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 50ns. 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 ***single_cycle_mips_mapped.v*** and your SDF file as ***single_cycle_mips_mapped.sdf*** (for automated checking purposes).

3 Submission

Grading will be broken down as follows:

- Behavioral Model (50%)
  - 25% LW, SW
  - 15% ADD, SUB, SLT (requires LW, SW)
  - 10% BEQ, BNE (requires LW, SW)

The ADD and BEQ instructions will only be graded if both LW and SW instructions are fully functional.

Deadline for checking of all parts of the milestone will be done on February 9, 2017. Checking will be done in class, using an automated checker testbench that will be provided on the day of checking. Make sure that your codes are working before having them checked.