ECE/CS 552: INTRODUCTION TO COMPUTER ARCHITECTURE

Project Description – Stage 1

Due on Sunday, March 18, 2018, 23:55

In stage 1, the task is to design and implement a single cycle processor. The implementation should be in Verilog. Either Modelsim or Icrarus should be used as the simulator to verify the design. Below, the WISC-S18 ISA specification will be introduced; and then design details and requirements will be discussed. **You are required to follow the Verilog rules as specified by the rules document uploaded on canvas. NOTE: the only exception to this rule is the required use of *inout* (tri-state logic) in the register file. Do not use *inout* anywhere else in your design.**

Some portions of the project requirements are covered in the homeworks. You are free to reuse those modules in your project.

1. WISC-S18 ISA Specifications

WISC-S18 contains a set of 16 instructions specified for a 16-bit data-path with load/store architecture.

The WISC-S18 memory is byte addressable, even though all accesses (instruction fetches, loads, stores) are restricted to halfword (2 byte), naturally-aligned accesses.

WISC-S18 has a register file, and a 3-bit FLAG register. The register file comprises sixteen 16-bit registers and has 2 read ports and 1 write port. Register \$0 is hardwired to 0x0000. The FLAG register contains three bits: Zero (Z), Overflow (V), and Sign bit (N).

WISC-S18's instructions can be categorized into three major classes: Compute, Memory, and Control.

1.1 Compute Instructions

Six arithmetic and logical instructions belong to this category. They are ADD, PADDSB, SUB, XOR, SLL, SRA, ROR and RED.

The assembly level syntax for ADD, PADDSB, SUB, XOR and RED instructions is

Opcode rd, rs, rt

The two operands are[1] (rs) and (rt) and the result is written to the destination register rd.

The ADD, PADDSB, SUB and RED instructions work on the two operands (rs, rt) in two's-complement representation and save the result in register rd.

The **ADD** and **SUB** instructions will use saturating arithmetic. Meaning if a result exceeds the most positive number (2^15−1), an overflow, then the result is saturated to (2^15−1). Likewise, if the result is smaller than the most negative number (−2^15), an underflow, then the result would be saturated to −2^15.

The **PADDSB** instruction performs four half-byte additions in parallel to realize *sub-word parallelism*. Specifically, each of the four half bytes (4-bits) will be treated as separate numbers stored in a single

[1] (rx) stands for the content of register rx.

word as a byte vector. When PADDSB is performed, the four numbers will be added separately. To be more specific, let the contents in rs and rt are aaaa_bbbb_cccc_dddd, eeee_ffff_gggg_hhhh respectively where a, b, c, d, e, f, g, and h in {0, 1}. Then after execution of PADDSB, the content of rd will be {sat(aaaa+eeee), sat(bbbb+ffff), sat(cccc+gggg), sat(dddd+hhhh)}. The four half-bytes of result should be saturated separately, meaning if a result exceeds the most positive number (2^3-1) then the result is saturated to (2^3-1), and if the result were to underflow the most negative number (-2^3) then the result would be saturated to −2^3.

The **RED** instruction performs reduction on 8 half-byte operations (i.e. 4 half-bytes each from 2 registers). To be more specific, let the contents in rs and rt are aaaa_bbbb_cccc_dddd, eeee_ffff_gggg_hhhh respectively where a, b, c, d, e, f, g, and h in {0, 1}. Then after execution of RED, the content of rd will be the sign extended value of (((aaaa+eeee) + (bbbb+ffff)) + ((cccc+gggg) + (dddd+hhhh))).

The **XOR** instruction performs bitwise XOR, operations on the two operands and save the result in register rd.

The **SLL**, **SRA** and **ROR** instructions perform logical left shift, arithmetic right shift and rotation respectively, of (rs) by number of bits specified in the imm field and saves the result in register rd. For ROR, bits are rotated off the right end are inserted into the vacated bit positions on the left.

They have the following assembly level syntax:

Opcode rd, rs, imm

The imm field is a 4-bit immediate operand in unsigned representation for the SLL, ROR, and SRA instructions.

The machine level encoding for each arithmetic/logic instruction is

0aaa dddd ssss tttt

where aaa represents the opcode (see Table 2), dddd and ssss respectively represent the rd and rs registers. The tttt field represents either the rt register or the imm field.

1.2 Memory Instructions

There are four instructions of this category: LW, SW, LHB and LLB.

The first group of these instructions are LW (load word), and SW (save word). The assembly level syntax for the LW and SW instructions is

Opcode rt, rs, offset

The **LW** instruction loads register rt with contents from the memory location specified by register rs plus the immediate offset. The signed value offset is sign-extended and added to the contents of register rs to compute the address of the memory location to load. The address is always even (the low-order bit will always be zero).

The **SW** instruction saves (rt) to the location specified by the register rs plus the immediate offset. The address of the memory location is computed as in LW. The address is always even (the low-order bit will always be zero).

The machine level encoding of these two instructions is

10aa tttt ssss oooo

where aa specifies the opcode, tttt identifies rt, ssss identifies rs, and oooo is the offset in 2's complement representation, but right-shifted by 1 bit (since the LSB will always be zero, there is no reason to encode that bit in the instruction word). The address is computed as addr = (Reg[ssss] & 0xFFFE) + (oooo << 1).

The next two instructions are of the Load Immediate type: **LHB** (load higher byte), and **LLB** (load lower byte). The assembly level syntax for the LHB and LLB instructions is:

LLB Rx, 0xYY

LHB Rx, 0xYY

Where Rx is the register being loaded, and 0xYY is the 8-bit immediate value.

LHB instruction loads the most significant 8 bits of register rd with the bits in the immediate field. The least significant 8 bits of the register rd are left unchanged. Conversely, LLB loads the least significant 8 bits of rd while the most significant remains unchanged.

Note: These two aren't technically loading from memory but are grouped with memory instructions

The machine level encoding for these instructions is

101a dddd uuuu uuuu

where a is a bit to differentiate LLB vs LHB, dddd specifies the destination registers, and uuuuuuuu is the 8-bit immediate value.

Note that LLB/LHB must not overwrite the upper/lower half of Reg[dddd], and your register file design does not support partial register writes, so you will have to implement them using a read-modify-write register transfer: Reg[dddd] = (Reg[dddd] & 0xFF00) | uuuuuuuu for LLB and Reg[dddd] = (Reg[dddd] & 0x00FF) | (uuuuuuuu << 8) for LHB.

1.3 Control Instructions

There are four instructions belonging to this category: B, BR, PCS, and HLT

The **B** (Branch) instruction conditionally jumps to the address obtained by adding the 9-bit immediate (signed) offset to the contents of the program counter+2 (i.e., address of Branch instruction + 2).

The assembly level syntax for this instruction is

B cond, Label

The machine level encoding for this instruction is

Opcode ccci iiii iiii

where ccc specifies the condition as in Table 1 and iiiiiiiii represents the 9-bit signed offset in two's-complement representation. You will need to left-shit the offset by 1 (since you are accessing half-words i.e. 2 bytes, in a byte-addressable memory). The target is computed as: target = PC + 2 + (iiiiiiiii << 1).

The **BR** (Branch Register) instruction conditionally jumps to the address specified by (rs).

The assembly level syntax for this instruction is

BR cond, rs

The machine level encoding for this instruction is

Opcode cccx ssss xxxx

where ccc specifies the condition as in Table 1 and ssss encodes name of register rs.

Table 1: Encoding for Branch conditions

ccc	*Condition*
000	Not Equal (Z = 0)
001	Equal (Z = 1)
010	Greater Than (Z = N = 0)
011	Less Than (N = 1)
100	Greater Than or Equal (Z = 1 or Z = N = 0)
101	Less Than or Equal (N = 1 or Z = 1)
110	Overflow (V = 1)
111	Unconditional

The eight possible conditions are Equal (EQ), Not Equal (NEQ), Greater Than (GT), Less Than (LT), Greater Than or Equal (GTE), Less Than or Equal (LTE), Overflow (OVFL), and Unconditional (UNCOND). Many of these conditions are determined based on the 3-bit flag N, V, and Z. Instruction that set flags are outlined in the table 2 below:

Table 2: Flags set by instructions

Instruction	Flags Set
ADD	N, Z, V
SUB	N, Z, V
XOR	Z
SLL	Z
SRA	Z
ROR	Z

The True condition corresponds to an unconditional branch. The status of the condition is obtained from the FLAG register (definition of each flag is in section 3.3).

The **PCS** instruction saves the contents of the next program counter (address of the PCS instruction + 2) to the register rd and increments the PC.

The assembly level syntax for this instruction is

PCS rd

The machine level encoding for this instruction is

Opcode dddd xxxx xxxx

where dddd encodes register rd.

The **HLT** instruction essentially freezes the whole machine by stopping advancing PC.

Opcode xxxx xxxx xxxx

The list of instructions and their opcodes are summarized in Table 3 below.

Table 3: Table of opcodes

Instruction	Opcode
ADD	0000
SUB	0001
RED	0010
XOR	0011
SLL	0100
SRA	0101
ROR	0110
PADDSB	0111
LW	1000
SW	1001
LHB	1010
LLB	1011
B	1100
BR	1101
PCS	1110
HLT	1111

2. Memory System

For this stage of the project, the processor will have separate single-cycle instruction and data memory which are bye-addressable. The instruction memory has a 16-bit address input and a 16-bit data output. The data memory has a 16-bit address input, a 16-bit data input, a 16-bit data output, and a write enable signal. If the write signal is asserted, the memory will write the data input bits to the address specified by the address. Both instruction and data memories are implemented as asynchronous memories.

Verilog modules will be provided for both memories.

The instruction memory contains the binary machine code instructions to be executed on your design.

3. Implementation

3.1 Design

As mentioned earlier, in this stage you are about to implement the ISA and design a single cycle processor. On each clock cycle, one instruction is first read from instruction memory, then it is executed and finally the results are stored. So in the single cycle processor, each instruction takes only one cycle to execute. Required design specifications for specific modules are provided below:

1. **ALU Adder**: Carry Look-Ahead adder
2. **Shifter**: Using 3:1 muxes, an extension of the figure from the class slides
3. **Register File**: As specified in the homework
4. **Reduction unit (for RED instruction)**: Using a tree of 4-bit Carry Lookahead Adders. The first level in the tree can be shared with your CLA, but carries are not propagated. The second level has two CLAs, while the third level has one CLA. This tree computes bits $R_{3:0}$ of the reduction. Since 8 4-bit values are being added together (reduced), the final sum requires up to 7 bits to represent. Hence, bits R6:4 need to computed as the sum of the carries from the 7 CLA adders in the reduction tree. You should use a Wallace tree of CSAs (carry save adders) to sum these carry bits (see lecture notes for an example).

3.2 Reset Sequence

WISC-S18 has an active low reset input (rst_n). Instructions are executed when rst_n is high. If rst_n goes low for one clock cycle, the contents of the state of the machine is reset and starts execution at address 0x0000.

3.3 Flags

Flag bits are stored in the FLAG register and are used in conditional jump. There are three bits in the FLAG register: Zero (Z), Overflow (V), and Sign bit (N). Only the arithmetic instructions (except PADDSB, RED) can change all three flags (Z, V, N). The logical instructions (XOR, SLL, SRA, ROR) change the Z FLAG, but they do not change the N or V flag.

The Z flag is set if and only if the output of the operation is zero.

The V flag is set by the ADD and SUB instructions if and only if the operation results in an overflow. Overflow must be set based on treating the arithmetic values as 16-bit signed integers.

The N flag is set if and only if the result of the ADD or SUB instruction is negative.

Other Instructions, including load/store instructions and control instructions, do not change the contents of the FLAG register.

4. Interface

Your top level verilog should be file called ***cpu.v***. It should have a simple 4 signal interface: ***clk***, ***rst_n***, ***hlt*** and ***pc[15:0]***.

Signal Interface of ***cpu.v***		
Signal:	**Direction:**	**Description:**
clk	in	System clock
rst_n	in	Active low reset. A low on this signal resets the processor and causes execution to start at address 0x0000
hlt	out	When your processor encounters the HLT instruction it will assert this signal once it is finished processing the instruction prior to the HLT.
pc[15:0]	Out	PC value over the course of program execution.

5. Submission Requirements

1. You are provided with an assembler to convert your text-level test cases into machine level instructions. You will also be provided with a global test bench and test case. The test case should be run with the testbench and the output (as a .txt file) should be submitted for Phase 1 evaluation.

2. You are also required to submit a zipped file containing: all the Verilog files of your design, all testbenches used and any other support files.