EECS 151/251A Homework 4

Due Friday, Oct 4^{th}, 2019

Reading

In addition to reviewing the RISC-V ISA and datapath lectures, skim through the RISC-V ISA spec. In particular, focus on the Introduction, Chapter 2 (RV32I Base Integer Instruction Set), and the table on page 130.

Problem 1: RISC-V Manual Assembly

Manually construct the binary instruction for the following assembly instructions. Provide the solution as a 32-bit binary number.

(a) `add x1, x2, x3`

(b) `addi x1, x2, 100`

(c) `lb x1, 4(x2)`

(d) `beq x6, x8, 1024`

Problem 2: RISC-V Assembly Programs

Write down the values of the specified registers after the following programs have run:

(a)
```
li x1, 100
li x2, 200
add x3, x1, x2
sub x3, x3, x1
```

`x1 = _____, x2 = _____, x3 = _____`

(b)
```
li x1, 100
li x0, 200
add x0, x1, x0
```

`x0 = _____, x1 = _____`

(c)
```
li x1, 0xdead
li x2, 0xbeef
li x3, 0x1024
sh x1, 0(x3)
sh x2, 2(x3)
lw x4, 0(x3)
```

x1 = _____, x2 = _____, x4 = _____

(d)
```
li x1, -1
li x2, 1
li x3, 0
bge x1, x2, f1
li x3, 100
f1: addi x3, x3, 100
```

x1 = _____, x2 = _____, x3 = _____

(e) Assume the following instructions start at address 0x0.

```
li x1, 0
jalr x2, x0, 20
addi x1, x1, 100
addi x1, x1, 200
jal x0, end
jalr x3, x2, 0
end: nop
```

x1 = _____, x2 = _____, x3 = _____

Problem 3: RISC-V Psuedo-instructions

Several psuedo-instructions are defined by the RISC-V assembler which translate to sequences of one or more RISC-V base instructions. For each psuedo-instruction, write the RISC-V assembly instructions that implement it. Refer to page 130 of the RISC-V spec for a list of the base RISC-V instructions.

(a) `nop`. Do nothing, don't change the architectural state.

(b) `mv rd, rs`. Move the value in register `rs` to register `rd`.

(c) `li rd, imm`. Load a 32-bit immediate into register `rd`.

(d) `beqz rs, imm`. Branch if `rs` is greater than or equal to zero.

(e) `j imm`. Jump to PC += imm and don't link any register.

(f) `bgt rs1, rs2, imm`. Branch if `rs1` is greater than `rs2`.

Problem 4: RISC-V Instruction Decoder

Consider the complete RV32I datapath drawn in the lecture slides. Write down logical expressions for the following control signals in terms of an instruction's `opcode, funct3`, and `funct7` bits.

You can simplify the expressions by comparing the `opcode` field against the constants in Table 25.1 in the RISC-V ISA manual. For example: `assign sig = (opcode == OP-32) || (opcode == LOAD);`

(a) `WBSel`

(b) `MemRW`, assume `0 = read` and `1 = write`

(c) `PCSel`

(d) `BSel`

Problem 5: RISC-V Memory Decoder

The RV32I ISA defines 5 memory load instructions: `lb, lbu, lh, lhu, lw`. Complete the implementation of a Verilog module which converts the raw output of the data memory to the value to be written to the regfile according to the type of load instruction and the memory address.

```
module load_decoder(
  input [31:0] addr, // the byte-address for the load instruction
  input [31:0] raw_data, // the raw data from the DMEM
  input lb, lbu, lh, lhu, lw, // type of load instruction, only 1 is high at a time
  output [31:0] wb_data // writeback data (to the regfile)
);

    // Your implementation

endmodule
```

Problem 6: RV64I ALU

Refer to Chapter 5 (RV64I Base Integer Instruction Set) in the RISC-V spec. Implement an ALU that supports the `add, sll, sub` 64-bit integer instructions as well as their 'W' suffix variants.

```
`define ALU_ADD 0
`define ALU_ADDW 1
`define ALU_SUB 2
`define ALU_SUBW 3
`define ALU_SLL 4
`define ALU_SLLW 5
`define ALU_SRA 6
`define ALU_SRAW 7
module rv64_alu(
  input [63:0] a,
  input [63:0] b,
  input [2:0] op, // op can be any of values `define'd above
  output [63:0] c,
);

  // Your implementation
```

```
endmodule
```