EC 413 Computer Organization – Fall 2019
Problem Set 3 Solution

Important guidelines: Always state your assumptions and clearly explain your answers. Please upload your solution document (PDF or TXT) to Blackboard. Your submission should be a single file titled *[your name]_pset3.pdf* (example: JohnDoe_pset3.pdf) or *[your name]_pset3.txt*.

Part 1: Instruction Encoding and Decoding and Bit manipulation

Exercise 1: [Instruction Encoding]

a. 0x01162223
b. 0x009A0C93
c. 0x00B68533

Exercise 2: [Instruction Decoding]

a. sw ra, 12(sp) OR sw x1, 12(x2)
b. andi s0,sp,16 OR andi x8, x2, 16
c. lw a2, 4(s0) OR lw x2, 4(x8)
d. add a3,a2,a3 OR add x13, x12, x13

Exercise 3: [Bit manipulation]

a. 0x1236FEF8
b. 0x00000545

Part 2: Single Cycle CPU

Exercise 1:

a. ALUSrc = 0; ALUOp = 000; MemtoReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; Branch = 0; Jump = 0
   Although mathematically the MemRead control could be a don't care, practically, it should be set to 0 (false) to prevent causing a segmentation fault or a cache miss.
b. Registers, ALUSrc mux, MemtoReg mux (also I-mem, PC, and PC increment adder).
c. All blocks produce some output, except for the data memory, since MemRead=0 and MemWrite=0. The output of the Sign Extender is not used.

Exercise 2:

a. 25% + 10% = 35%. Only *Load* and *Store* use Data memory.

b. 100%. Every instruction must be fetched from the instruction memory before it can be executed.
c. 28% + 25% + 10% + 11% + 2% = 76%. Only R-type instructions do not use the Sign extender.
d. The sign extend produces an output during every cycle. If its output is not needed, it is simply ignored.

Exercise 3:

a. Only *Load* is broken. *MemtoReg* is either 1 or don't care for all other instructions.
b. *Load*, *Store* are all broken.
c. *Store* and *SB-type* are all broken.

Exercise 4:
The encoded instruction is *sw x12, 20(x13)*

a. ALUop: 101
   Opcode: 0100011
   Funct3: 010
b. The new PC is the old PC+4. This signal goes from the PC, through the "PC+4"adder, through the "branch" mux, and back to the PC.
c. For each mux, specify the values of its inputs and outputs. List the values that are register outputs at Reg [Xn].
   ALUSrc: Inputs : Reg[x12] and 0x0000000000000014;
   Output: 0x0000000000000014
   MemtoReg: Inputs: Reg[x13] + 0x14 and <undefined>;
   Output: <undefined>
   Branch: Inputs: PC+4 and 0x000000000000000A
d. ALU Inputs: Reg[x13] and 0x0000000000000014;
   PC+4 adder inputs: PC and 4
   Branch adder inputs: PC and 0x000000000000000A
e. Read Addr 1 : 13
   Read Addr 2 : 12
   Write Addr: 12
   Write data: undefined. Either Reg[x13]+20, or the undefined output of data memory, depending on the value of the MemtoReg control line.

Part 3: CPU Pipeline

Exercise 1:

a. Pipeline: 350; non-pipeline: 1250
b. Pipeline: 1250; non-pipeline: 1250
c. Split the ID stage. This reduces the clock-cycle time to 300ps.
d. 35%
e. 65%

Exercise 2:

a. The energy for the two designs is the same: I-Mem is read, two registers are read, and a register is written. We have: 140pJ + 2*70pJ + 60j = 340pJ.
b. The instruction memory is read for all instructions. Every instruction also results in two register reads (even if only one of those values is actually used). A load instruction results in a memory read and a register write; a store instruction results in a memory write; all other instructions result in at most a single register write. Because the sum of memory read and register write energy is larger than memory write energy, the worst-case instruction is a load instruction. For the energy spent by a load, we have: 140pJ + 2*70pJ + 60pJ + 140pJ = 480pJ.
c. Instruction memory must be read for every instruction. However, we can avoid reading registers whose values are not going to be used. To do this, we must add RegRead1 and RegRead2 control inputs to the Registers unit to enable or disable each register read. We must generate these control signals quickly to avoid lengthening the clock cycle time. With these new control signals, a load instruction results in only one register read (we still must read the register used to generate the address), so our change saves 70pJ (one register read) per load. This is a savings of 70/480 = 14.6%.
d. *jal* will benefit, because it need not read any registers at all. *I-type* instructions will also benefit because they need only read one register. If we add logic to detect x0 as a source register, then instructions such as *beqz* (i.e. *beq* x0, …) and *li* (addi xn, x0, …) could benefit as well.
e. Before the change, the Control unit decodes the instruction while register reads are happening. After the change, the latencies of Control and Register Read cannot be overlapped. Tis increases the latency of the ID stage and could affect the processor's clock cycle time if the ID stage becomes the longest-latency stage. However, the sum of the latencies for the register read (90ps) and control unit (150ps) are less than the current 250ps cycle time.
f. If memory is read in every cycle, the value is either needed (for a load instruction), or it does not get past the WB Mux (for a non-load instruction that writes to a register), or it does not get written to any register (all other instructions, including branches and stalls). Tis change does not affect clock cycle time because the clock cycle time must already allow enough time for memory to be read in the MEM stage. It can affect overall performance if the unused memory reads cause cache misses.
The change also affects energy: A memory read occurs in every cycle instead of only in cycles when a load instruction is in the MEM stage. This increases the energy consumption by 140pJ during 75% of the 250ps clock cycles. This

corresponds to a consumption of approximately 0.46 Watts (not counting any energy consumed as a result of cache misses).