CENG3420 Homework 2

Due: Mar. 21, 2021

Please submit PDF or WORD document directly onto blackboard.
DO NOT SUBMIT COMPRESSED ZIP or TARBALL.

Solutions

Q1 (10%)

- Write down the bit pattern in the fraction of value 1/3 assuming a floating-point format that uses binary numbers in the fraction. Assume there are 24 bits, and you do not need to normalize. Is this representation exact?

A1.1
1. 0101 0101 0101 0101 0101 0101
2. No

- Write down the binary representation of the decimal number 63.25 assuming it was stored using the single precision. IBM format (base 16, instead of base 2, with 7 bits of exponent).

A1.2
1. $63.25 = 10^0 = 111111.012^0 = 3F.4016^0$
2. move hex point two to the left
3. $0.3F4016^2$
4. $sign = positive, exp = 64 + 2$
5. Final bit pattern: 01000010001111110100000000000000

Q2 (15%) Examine the difficulty of adding a proposed `lwi.d rd, rs1, rs2` ("Load With Increment") instruction to RISC-V. Interpretation: `Reg[rd]=Mem[Reg[rs1]+Reg[rs2]]`

1. Which new functional blocks (if any) do we need for this instruction?
2. Which existing functional blocks (if any) require modification?
3. Which new data paths (if any) do we need for this instruction?
4. What new signals do we need (if any) from the control unit to support this instruction?

A2

1. No new functional blocks are needed
2. Only the control unit needs modification
3. No new data paths are needed
4. No new data paths are needed
5. Final bit pattern: 01000010001111110100000000000000

Q3 (15%) Add NOP instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

```
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
add x15, x13, x12
```

A3

1. addi x11, x12, 5
2. NOP
3. NOP
4. add x13, x11, x12
5. addi x14, x11, 15
6. NOP
7. add x15, x13, x12

Q4 (15%) In this exercise, we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word: 0x00c6ba23.

1. What are the values of the ALU control unit inputs for this instruction?
2. What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.
3. For each mux, show the values of its inputs and outputs during the execution of this instruction. List values that are register outputs at Reg [xn].
4. What are the input values for the ALU and the two add units?

A4

1. $ALUop = 00, ALUControlLines = 0010$
2. 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.
3. ALUsrc : Inputs: Reg[x12] and 0x0000000000000014; Output:0x0000000000000014
4. MemToReg : Inputs: Reg[x13] + 0x14 and <undefined>; output: <undefined>
5. Branch: Inputs: PC+4 and 0x000000000000000A
6. ALU inputs: Reg[x13] and 0x0000000000000014; PC + 4 adder inputs: PC and 4; Branch adder inputs: PC and 0x0000000000000028

Q5 (15%) If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. As a result, the MEM and EX stages can be overlapped and the pipeline has only four stages.

1. How will the reduction in pipeline depth affect the cycle time?
2. How might this change improve the performance of the pipeline?
3. How might this change degrade the performance of the pipeline?

A5

1. The clock period won't change because we aren't making any changes to the slowest stage.
2. Moving the MEM stage in parallel with the EX stage will eliminate the need for a cycle between loads and operations that use the result of the loads. This can potentially reduce the number of stalls in a program.
3. Removing the off set from ld and sd may increase the total number of instructions because some ld and sd instructions will need to be replaced with a addi/ld or addi/sd pair

Q6 (30%)

Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add x15, x12, x11
ld x13, 4(x15)
ld x12, 0(x2)
or x13, x15, x13
sd x13, 0(x15)
```

1. If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.
2. Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register x17 can be used to hold temporary values in your modified code.
3. If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?
4. If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in the figure.

A6 1.
- add x15, x12, x11
- nop
- nop
- ld x13, 4(x15)
- ld x12, 0(x2)
- nop
- or x13, x15, x13
- nop
- nop
- sd x13, 0(x15)

2. It is not possible to reduce the number of NOPs.
3. The code executes correctly. We need hazard detection only to insert a stall when the instruction following a load uses the result of the load. That does not happen in this case.
4. (Correctd) Because there are no stalls in this code, PCWrite and IF/IDWrite are always 1 and the mux before ID/EX is always set to pass the control values through.

Table 1: Answer of Q6-4

Cycle	1	2	3	4	5	6	7	8	
add	IF	ID	EX	ME	WB				
ld		IF	ID	EX	ME	WB			
ld			IF	ID	EX	ME	WB		
or				IF	ID	EX	ME	WB	
sd					IF	ID	EX	ME	WB

1. ForwardA = X; ForwardB = X (no instruction in EX stage yet)
2. ForwardA = X; ForwardB = X (no instruction in EX stage yet)
3. ForwardA = 0; ForwardB = 0 (no forwarding; values taken from registers)
4. ForwardA = 2; ForwardB = 0 (base register taken from result of previous instruction)
5. ForwardA = 0; ForwardB = 0 (base register taken from registers)
6. ForwardA = 0; ForwardB = 1 (rs1 = x15 taken from register; rs2 = x13 taken from result of 1st ld – two instructions ago)
7. ForwardA = 0; ForwardB = 0 (base register taken from register file)