LSU EE 4720

Homework 4 Solution

In the problems below a new MIPS instruction, integer fmadd, (hypothetical of course) is to be added to our pipelined MIPS implementation. A simpler implement-the-instruction problem was the subject of Fall 2010 Homework 3, in which a shift unit is added to MIPS to implement shift instructions. The 2010 problem is simpler because the shift unit occupies just one stage, while the fmadd for this assignment spans multiple stages. For past assignments in which integer arithmetic hardware spans several stages see 2020 Homework 2, 3, and 4 and 2020 midterm exam Problem 5. In these 2020 problems an integer multiply instruction was to be implemented.

Problem 1: A fused multiply/add instruction, such as fmadd r1, r2, r3, r4, computes $r_1 = r_2r_3 + r_4$. Such instructions are useful for both floating-point and integer calculations, and integer version is considered here. The goal in this problem is to extend MIPS with an integer multiply/add instruction, fmadd. The new fmadd instruction will be encoded in MIPS Format R with the SA field being used to specify the third source register, r4 in the example.

	Opcode		RS	RT		RD		SA	Function	
MIPS R:		0	2		3		1	4	fmade	d
	31	26	25 21	20	16	15	11	10 6	4 (0

The hardware to compute the multiply/add will consist of two types of units: a carry-save multiplier (CSM) and integer adders (labeled ADD). The connection of these two types of units needed to compute a multiply/add are shown below.



The CSM takes 0.9 clock cycles to compute a result and each adder takes 0.45 clock cycles, so the critical path through the hardware shown above is 1.8 clock cycles. Because the critical path is greater than one clock cycle the hardware cannot be placed in one stage. (Unless the clock frequency were to be decreased from ϕ to $\phi/1.8$, which would slow everything down and so of course we don't want to do it.)

Note: For the three parts below a single hardware solution can be provided. That is, a correct solution to part c also can be a correct solution to parts b and a, and so there is no need to draw three hardware designs.

(a) Add the CSM and ADD units to the MIPS implementation below to efficiently implement the fmadd instruction. For this sub-problem provide the hardware needed so that fmadd can execute without stalls when there are no nearby dependencies, such as in the execution below.

```
# There are no dependencies in this code fragment.
# Cycle
                       0 1 2 3 4
                                     5 6 7 8
                       IF ID EX ME WB
add r1, r2, r3
sub r4, r5, r6
                          IF ID EX ME WB
fmadd r7, r8, r9, r10
                             IF ID EX ME WB
fmadd r11, r12, r13, r14
                                IF ID EX ME WB
xori r15, r16, 17
                                   IF ID EX ME WB
# Cycle
                       0
                             2 3 4
                                     5
                                         6 7
                         1
                                              8
```

Put your solution on the larger diagram several pages ahead.



Put your solution on the larger diagram several pages ahead.

7 Add the CSM and ADD units to the implementation above so that the can implement the fmadd instruction.

 \checkmark Provide the datapath needed so that operands can reach the CSM and ADD units and \checkmark the result can reach the register file.

- Don't forget that this instruction has three source operands.
- \checkmark Do not increase the critical path.
- As always, consider cost. Assume that an n-bit register costs twice as much as an n-bit, 2-input multiplexor.
- \uparrow fmadd should execute without stalls when there are no nearby dependencies.
- **Do not** design control logic for this assignment.

The solution and its discussion appear several pages ahead.

(b) In the code fragments below the fmadd depends on prior instructions.

Add bypass paths to the fmadd implementation so that all of the executions below are possible.

The solution and its discussion appear on the next page.

 \checkmark

 \checkmark

```
# Fragment A
                       0 1 2 3 4 5
# Cycle
                                        6
                       IF ID EX ME WB
add R1, r2, r3
                          IF ID EX ME WB
sub R4, r5, r6
                             IF ID EX ME WB
fmadd r7, R1, R4, r9
# Fragment B
# Cycle
                       0 1 2 3 4 5
                                        6 7
                          IF ID EX ME WB
sub R9, r5, r6
fmadd R7, r1, r4, R9
                             IF ID EX ME WB
fmadd r2, r3, r5, R7
                                IF ID EX ME WB
# Cycle
                       0 1 2 3 4 5 6 7
# Fragment C
# Cycle
                       0 1 2 3 4 5 6
add R1, r2, r3
                       IF ID EX ME WB
lw R9, 0(r10)
                          IF ID EX ME WB
fmadd r7, R1, r4, R9
                             IF ID EX ME WB
```

(c) Using additional ADD unit(s) modify the implementation so that it can execute Fragments L and D correctly. This will require some tricky bypassing. Note that stalls will be needed when the dependent instruction following the fmadd does not use the adder, such as in Fragment E. Note: In the original problem just one adder was to be used. That is probably impossible without critical path impact.

Add a second adder and bypass paths so that fragments L and D execute as shown.

The solution and its discussion appear on the next page.

```
# Fragment L
# Cycle
                       0 1 2 3 4
                                      5
                                         6
fmadd R7, r1, r4, r9
                       IF ID EX ME WB
lw r10, 16(R7)
                          IF ID EX ME WB
                                             # No stall!
# Fragment D
# Cycle
                       0 1 2 3 4 5 6
fmadd R7, r1, r4, r9
                       IF ID EX ME WB
add r2, R7, r3
                          IF ID EX ME WB
                                             # No stall!
# Fragment E
                       0 1 2 3 4
# Cycle
                                      5
                                         6
                       IF ID EX ME WB
fmadd R7, r1, r4, r9
                          IF ID -> EX ME WB # A stall. :-(
or r2, R7, r3
```

Use the diagram below for your solution, or download

https://www.ece.lsu.edu/ee4720/2023/mpipei3.svg and edit with your favorite SVG editor. (The diagram was drawn with Inkscape.)

Solution appears below. The changes for parts a and b are shown in blue and the changes for part c are shown in green.

For part a, the most important thing was not to put CSM and an ADD in the same stage, because the delay of CSM already used most of the clock cycle. Notice that CSM's inputs are obtained from the ALU's multiplexors, so that for the multiplier and multiplicand operands of fmadd no further changes are needed for bypassing.

The fmadd instruction uses a third source operand, sa. For this operand a third read port is added to the register file (in ID). In EX the sa value uses the rtv path to reach ME, saving the need for an sav pipeline latch between EX and ME. The output of the second ADD is the result of the fmadd instruction, it is connected to a new mux at the top of ME where it joins the path leading to the register file input.

The changes described above provide the bypass paths needed for Fragment A. For fragments B and C a bypass is needed for the sa value, for example R9 in Fragment C. That bypass is provided by the mux in the lower part of ME. Other bypasses needed for the sa value are provided by the existing mux that provides a value for the ME.rtv pipeline latch.

For part c two ADD units are used, these are shown in green. (The original problem was to use just one, which is probably impossible without critical path impact.)

