EE 466 Computer Architecture

Fall 2019

Instructor: Dr. Chen Liu

Project 3: Forwarding Units

Name: Lewis Collum

Student ID: 0621539

Major: EE & CE

Email Address: colluml@clarkson.edu

Date of the report: 12/13/2019

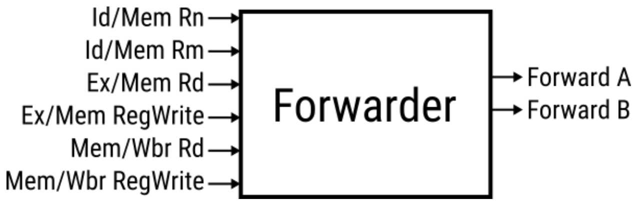
Contents

Abstract

We will be implementing a pipeline forwarding unit for a data-path.

1 Design Details

A forwarding unit is meant to reduce the amount of stalling that occurs on the data-path due to instruction dependency. We are designing one in VHDL with the following inputs and outputs



The outputs `forward A` and `forward B` can be summed in the following table.

Mux Control	Source	Explanation
ForwardA = 00	ID/EX	1st ALU operand comes from the register file
ForwardA = 10	EX/MEM	1st ALU operand is forwarded from the prior ALU result
ForwardA = 01	MEM/WB	1st ALU operand is forwarded from data memory or an earlier ALU result
ForwardB = 00	ID/EX	2nd ALU operand comes from the register file
ForwardB = 10	EX/MEM	2nd ALU operand is forwarded from the prior ALU result
ForwardB = 01	MEM/WB	2nd ALU operand is forwarded from data memory or an earlier ALU result

The inputs are handled conditionally as shown in the pseudo-code below

```
if isExecutionHazardFromN then
  forwardA <= "10";
elsif isMemoryHazardFromN then
  forwardA <= "01";
else
  forwardA <= "00";
end if;

if isExecutionHazardFromM then
  forwardB <= "10";
elsif isMemoryHazardFromM then
  forwardB <= "01";
else
  forwardB <= "00";
end if;

function isExecutionHazardFromN return boolean
  return exMemRegWrite = '1' and
    exMemD /= 31 and
    exMemD = n;
```

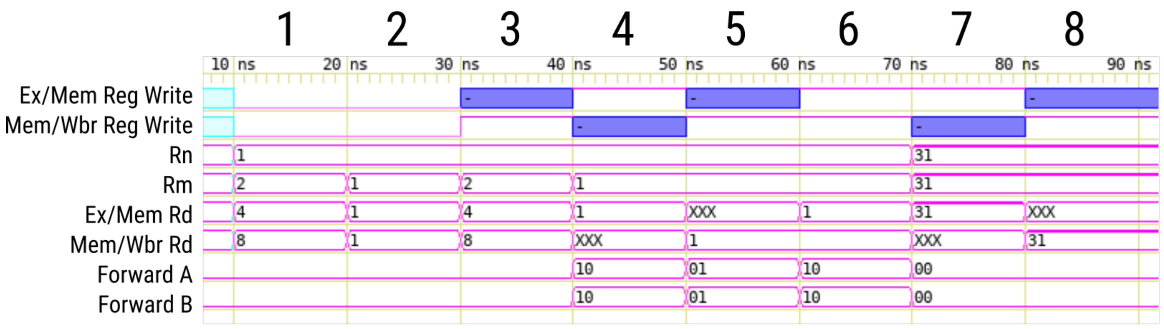
```
function isExecutionHazardFromM return boolean
  return exMemRegWrite = '1' and
    exMemD /= 31 and
    exMemD = m;

function isMemoryHazardFromN return boolean
  return memWbrRegWrite = '1' and
    memWbrD /= 31 and
    not (exMemRegWrite = '1' and exMemD /= 31 and exMemD = n) and
    memWbrD = n;

function isMemoryHazardFromM return boolean
  return memWbrRegWrite = '1' and
    memWbrD /= 31 and
    not (exMemRegWrite = '1' and exMemD /= 31 and exMemD = m) and
    memWbrD = m;
```

Further explanation of the conditionals are included in the Waveform section.

2 Waveform



Each test, for our forwarder, is illustrated on the waveform and is labeled with a numbered. We will discuss the results of each test and include a LegV8 equivalent test.

1. Both register write flags are off which means we are not going to write to the register file and we do not need to forward. Thus, `forward A` and `forward B` are off.
2. Once again both register write flags are off which means we are not going to write to the register file and we do not need to forward, regardless of the fact that the operand registers match. Thus, `forward A` and `forward B` are off.
3. Now the Mem/Wbr register write flag is set, but the operand registers do not match, which means forwarding does not occur.
4. The Ex/Mem register write flag is set and the operand register match, so the first and second ALU operands are forwarded from the prior ALU result.
5. The Mem/Wbr register write flag is set and the operand registers match, so the first and second ALU operands are forwarded from data memory or and earlier ALU result.
6. Both the Mem/Wbr and Ex/Mem register write flags are set. Since execution happens first, the first and second ALU operands are forwarded from the prior ALU result.
7. This time the operand registers equal 31 (XZR). Since we expect this register to be equal to 0, we do not want to forward it since it may forward a non-zero result.
8. Again, the operand registers equal 31 (XZR). Since we expect this register to be equal to 0, we do not want to forward it since it may forward a non-zero result.

3 Self-Evaluation

Struggled with weeding through the vhdl logic for the forwarder. As a solution I extracted the if-else boolean logic into VHDL functions.

4 Appendix: Code

4.1 Forwarder

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Forwarder is
  port(
    n, m: in unsigned(4 downto 0);

    exMemD: in unsigned(4 downto 0);
    exMemRegWrite: in std_logic;

    memWbrD: in unsigned(4 downto 0);
    memWbrRegWrite: in std_logic;

    forwardA: out unsigned(1 downto 0);
    forwardB: out unsigned(1 downto 0));
end entity;

architecture behavioral of Forwarder is
begin
  process(n, m, exMemD, memWbrD, exMemRegWrite, memWbrRegWrite)
    impure function isExecutionHazardFromN return boolean is
    begin
      return exMemRegWrite = '1' and
        exMemD /= 31 and
        exMemD = n;
    end function;

    impure function isExecutionHazardFromM return boolean is
    begin
      return exMemRegWrite = '1' and
        exMemD /= 31 and
        exMemD = m;
    end function;

    impure function isMemoryHazardFromN return boolean is
    begin
      return memWbrRegWrite = '1' and
        memWbrD /= 31 and
        not (exMemRegWrite = '1' and exMemD /= 31 and exMemD = n) and
        memWbrD = n;
    end function;

    impure function isMemoryHazardFromM return boolean is
    begin
      return memWbrRegWrite = '1' and
        memWbrD /= 31 and
        not (exMemRegWrite = '1' and exMemD /= 31 and exMemD = m) and
        memWbrD = m;
    end function;

  begin
    if isExecutionHazardFromN then
      forwardA <= "10";
    elsif isMemoryHazardFromN then
      forwardA <= "01";
    else
      forwardA <= "00";
    end if;

    if isExecutionHazardFromM then
      forwardB <= "10";
    elsif isMemoryHazardFromM then
      forwardB <= "01";
    else
      forwardB <= "00";
    end if;

  end process;
end architecture;
```

4.2 Testbench

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity TestForwarder is
end entity;

architecture test of TestForwarder is
  type test_Forwarder is record
    n, m: unsigned(4 downto 0);
    exMemD: unsigned(4 downto 0);
    exMemRegWrite: std_logic;
    memWbrD: unsigned(4 downto 0);
    memWbrRegWrite: std_logic;
    forwardA: unsigned(1 downto 0);
    forwardB: unsigned(1 downto 0);
  end record;

  signal forwarder: test_Forwarder;
begin
  process
  begin
    wait for 10 ns;

    --Test (a)
    forwarder.exMemRegWrite <= '0';
    forwarder.memWbrRegWrite <= '0';
    forwarder.n <= "00001";
    forwarder.m <= "00010";
    forwarder.exMemD <= "00100";
    forwarder.memWbrD <= "01000";
    wait for 10 ns;
    assert forwarder.forwardA = "00";
    assert forwarder.forwardB = "00";


    --Test (b)
    forwarder.exMemRegWrite <= '0';
    forwarder.memWbrRegWrite <= '0';
    forwarder.n <= "00001";
    forwarder.m <= "00001";
    forwarder.exMemD <= "00001";
    forwarder.memWbrD <= "00001";
    wait for 10 ns;
    assert forwarder.forwardA = "00";
    assert forwarder.forwardB = "00";
```

```
--Test (c)
forwarder.exMemRegWrite <= '-';
forwarder.memWbrRegWrite <= '1';
forwarder.n <= "00001";
forwarder.m <= "00010";
forwarder.exMemD <= "00100";
forwarder.memWbrD <= "01000";
wait for 10 ns;
assert forwarder.forwardA = "00";
assert forwarder.forwardB = "00";

--Test (d)
forwarder.exMemRegWrite <= '1';
forwarder.memWbrRegWrite <= '-';
forwarder.n <= "00001";
forwarder.m <= "00001";
forwarder.exMemD <= "00001";
forwarder.memWbrD <= (others => '-');
wait for 10 ns;
assert forwarder.forwardA = "10";
assert forwarder.forwardB = "10";

--Test (e)
forwarder.exMemRegWrite <= '-';
forwarder.memWbrRegWrite <= '1';
forwarder.n <= "00001";
forwarder.m <= "00001";
forwarder.exMemD <= (others => '-');
forwarder.memWbrD <= "00001";
wait for 10 ns;
assert forwarder.forwardA = "01";
assert forwarder.forwardB = "01";

--Test (f)
forwarder.exMemRegWrite <= '1';
forwarder.memWbrRegWrite <= '1';
forwarder.n <= "00001";
forwarder.m <= "00001";
forwarder.exMemD <= "00001";
forwarder.memWbrD <= "00001";
wait for 10 ns;
assert forwarder.forwardA = "10";
assert forwarder.forwardB = "10";

--Test (g)
forwarder.exMemRegWrite <= '1';
forwarder.memWbrRegWrite <= '-';
forwarder.n <= to_unsigned(31, 5);
forwarder.m <= to_unsigned(31, 5);
forwarder.exMemD <= to_unsigned(31, 5);
forwarder.memWbrD <= (others => '-');
wait for 10 ns;
assert forwarder.forwardA = "00";
assert forwarder.forwardB = "00";

--Test (h)
forwarder.exMemRegWrite <= '-';
forwarder.memWbrRegWrite <= '1';
forwarder.n <= to_unsigned(31, 5);
forwarder.m <= to_unsigned(31, 5);
forwarder.exMemD <= (others => '-');
forwarder.memWbrD <= to_unsigned(31, 5);
wait for 10 ns;
```

```
    assert forwarder.forwardA = "00";
    assert forwarder.forwardB = "00";

    wait for 10 ns;
    wait;
  end process;

  unit: entity work.Forwarder
    port map(
      n => forwarder.n,
      m => forwarder.m,
      exMemD => forwarder.exMemD,
      exMemRegWrite => forwarder.exMemRegWrite,
      memWbrD => forwarder.memWbrD,
      memWbrRegWrite => forwarder.memWbrRegWrite,
      forwardA => forwarder.forwardA,
      forwardB => forwarder.forwardB);
end architecture;
```