ECGR 3183 Project 4: Datapath Elements

By: Christopher Lowe

4/19/2022

**Program Counter**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.std\_logic\_arith.all;

use IEEE.std\_logic\_unsigned.all;

entity program\_counter is

Port ( pcIn : in STD\_LOGIC\_VECTOR (63 downto 0);

clk : in std\_logic;

pcOut : out STD\_LOGIC\_VECTOR (63 downto 0));

end program\_counter;

architecture Behavioral of program\_counter is

begin

process (clk) is

begin

if rising\_edge(clk) then

pcOut <= pcIn;

end if;

end process;

end Behavioral;

A screenshot of a computer

Description automatically generated

**Figure 1: Program Counter Output**

On the rising edge of every clock cycle, the output (pcOut) will be updated with the current contents of the input (pcIn)

**Instruction Memory**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity instruction\_memory is

Port ( read\_addr : in STD\_LOGIC\_VECTOR (63 downto 0);

instruction : out STD\_LOGIC\_VECTOR (31 downto 0));

end instruction\_memory;

architecture Behavioral of instruction\_memory is

type rom is array (0 to 63) of std\_logic\_vector(7 downto 0);

constant instr\_data : rom := (

0 => "11111000",

1 => "01000000",

2 => "00000001",

3 => "01000010",

4 => "11111000",

5 => "01000000",

6 => "10000001",

7 => "01000011",

8 => "11001011",

9 => "00000010",

10 => "00000000",

11 => "01100100",

12 => "10001011",

13 => "00000010",

14 => "00000000",

15 => "01100101",

16 => "10110100",

17 => "00000000",

18 => "00000000",

19 => "01000001",

20 => "10110100",

21 => "00000000",

22 => "00000000",

23 => "01000000",

24 => "11111000",

25 => "01000000",

26 => "00000001",

27 => "01000010",

28 => "10101010",

29 => "00000011",

30 => "00000000",

31 => "01000110",

32 => "10001010",

33 => "00000011",

34 => "00000000",

35 => "01000111",

36 => "11111000",

37 => "00000000",

38 => "10000000",

39 => "11100100",

40 => "00010100",

41 => "00000000",

42 => "00000000",

43 => "00000010",

44 => "11111000",

45 => "01000000",

46 => "10000001",

47 => "01000011",

48 => "10001011",

49 => "00000001",

50 => "00000000",

51 => "00001000",

others => "00000000");

begin

process (read\_addr) is

begin

instruction(7 downto 0) <= instr\_data(to\_integer(unsigned(read\_addr)) +0);

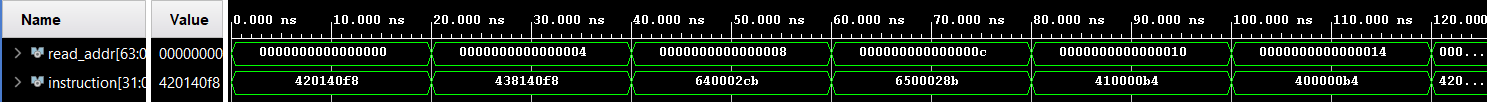
instruction(15 downto 8) <= instr\_data(to\_integer(unsigned(read\_addr)) +1);

instruction(23 downto 16) <= instr\_data(to\_integer(unsigned(read\_addr)) +2);

instruction(31 downto 24) <= instr\_data(to\_integer(unsigned(read\_addr)) +3);

end process;

end Behavioral;



**Figure 2: Instruction Memory Output**

When the instruction memory receives the address of the next instruction (read\_addr), it reads the memory (variable ram seen in above code) stored at that address and outputs it (instruction).

**Register File**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity register\_file is

Port ( readRegA : in STD\_LOGIC\_VECTOR (4 downto 0); --regASel

readRegB : in STD\_LOGIC\_VECTOR (4 downto 0); --regBSel

writeReg : in STD\_LOGIC\_VECTOR (4 downto 0); --writeRegSel

writeData : in STD\_LOGIC\_VECTOR (63 downto 0); --input

writeEnable : in STD\_LOGIC; --writeEnable

clk : in STD\_LOGIC;

readDataA : out STD\_LOGIC\_VECTOR (63 downto 0); --outA

readDataB : out STD\_LOGIC\_VECTOR (63 downto 0)); --outB

end register\_file;

architecture Behavioral of register\_file is

TYPE registerFile IS ARRAY(0 TO 31) OF STD\_LOGIC\_VECTOR(63 DOWNTO 0);

SIGNAL registers : registerFile := (OTHERS => X"0000000000000000");

begin

readDataA <= registers(to\_integer(unsigned(readRegA)));

readDataB <= registers(to\_integer(unsigned(readRegB)));

process (clk) is

begin

--detects rising clock edge

if rising\_edge(clk) then

if writeEnable = '1' and to\_integer(unsigned(writeReg)) /= 31 then

registers(to\_integer(unsigned(writeReg))) <= writeData;

end if;

end if;

end process;

end Behavioral;

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 3: Register File Output**

When the write enable (writeEnable) is set to ‘true’, the current input (writeData) will be written to the selected register (using writeReg), unless the targeted register is register #31 (the XZR register), in which case nothing will occur. When the two read signals are asserted (readRegA and readRegB), the contained data will be given as output (readDataA and readDataB).

**Sign Extend**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity sign\_extend is

Port ( s\_ext\_in : in STD\_LOGIC\_VECTOR (31 downto 0);

s\_ext\_out : out STD\_LOGIC\_VECTOR (63 downto 0));

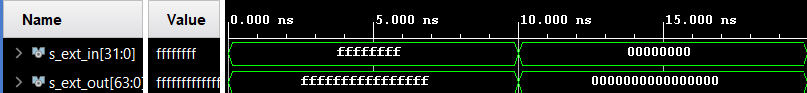
end sign\_extend;

architecture Behavioral of sign\_extend is

begin

s\_ext\_out <= std\_logic\_vector(resize(signed(s\_ext\_in), 64));

end Behavioral;



**Figure 4: Sign Extend Output**

When a 32-bit input is received (s\_ext\_in), the component performs a sign extension to transform the input into a 64-bit output (s\_ext\_out).

**ALU**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.std\_logic\_signed.all;

entity ALU\_main is

Port ( alu\_input\_a : in STD\_LOGIC\_VECTOR (63 downto 0);

alu\_input\_b : in STD\_LOGIC\_VECTOR (63 downto 0);

alu\_control : in STD\_LOGIC\_VECTOR (3 downto 0);

alu\_result : inout STD\_LOGIC\_VECTOR (63 downto 0);

zero : out STD\_LOGIC);

end ALU\_main;

architecture Behavioral of ALU\_main is

signal alu\_val : std\_logic\_vector (63 downto 0);

begin

process(alu\_input\_a, alu\_input\_b, alu\_control)

begin

case(alu\_control) is

when "0000" => alu\_val <= alu\_input\_a and alu\_input\_b;

when "0001" => alu\_val <= alu\_input\_a or alu\_input\_b;

when "0010" => alu\_val <= alu\_input\_a + alu\_input\_b;

when "0110" => alu\_val <= alu\_input\_a - alu\_input\_b;

when "0111" => alu\_val <= alu\_input\_b;

when "1100" => alu\_val <= alu\_input\_a nor alu\_input\_b;

when others => null;

end case;

end process;

zero <= '1' when alu\_result = x"0000000000000000" else '0';

alu\_result <= alu\_val;

end Behavioral;

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 5: ALU Output**

The ALU receives two inputs (alu\_input\_a and alu\_input\_b). Once received, the component then performs an operation on the inputs depending on the received opcode (alu\_control), and then gives an output (alu\_result). If the result is equal to zero, the zero flag is asserted (zero).

**Data Memory**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity data\_memory is

Port ( mem\_in : in STD\_LOGIC\_VECTOR (63 downto 0);

mem\_address : in STD\_LOGIC\_VECTOR (63 downto 0);

mem\_write : in STD\_LOGIC;

mem\_read : in STD\_LOGIC;

mem\_out : out STD\_LOGIC\_VECTOR (63 downto 0));

end data\_memory;

architecture Behavioral of data\_memory is

signal address : std\_logic\_vector(7 downto 0);

type memory is array (0 to 63) of std\_logic\_vector(7 downto 0);

signal ram : memory := (others => "00000000");

begin

write : process(mem\_write, mem\_address, mem\_in) is

begin

if (mem\_write = '1') then

ram(to\_integer(unsigned(mem\_address)) +0) <= mem\_in(7 downto 0);

ram(to\_integer(unsigned(mem\_address)) +1) <= mem\_in(15 downto 8);

ram(to\_integer(unsigned(mem\_address)) +2) <= mem\_in(23 downto 16);

ram(to\_integer(unsigned(mem\_address)) +3) <= mem\_in(31 downto 24);

ram(to\_integer(unsigned(mem\_address)) +4) <= mem\_in(39 downto 32);

ram(to\_integer(unsigned(mem\_address)) +5) <= mem\_in(47 downto 40);

ram(to\_integer(unsigned(mem\_address)) +6) <= mem\_in(55 downto 48);

ram(to\_integer(unsigned(mem\_address)) +7) <= mem\_in(63 downto 56);

end if;

end process write;

read : process(mem\_read, mem\_address) is

begin

if (mem\_read = '1') then

mem\_out(7 downto 0) <= ram(to\_integer(unsigned(mem\_address)) +0);

mem\_out(15 downto 8) <= ram(to\_integer(unsigned(mem\_address)) +1);

mem\_out(23 downto 16) <= ram(to\_integer(unsigned(mem\_address)) +2);

mem\_out(31 downto 24) <= ram(to\_integer(unsigned(mem\_address)) +3);

mem\_out(39 downto 32) <= ram(to\_integer(unsigned(mem\_address)) +4);

mem\_out(47 downto 40) <= ram(to\_integer(unsigned(mem\_address)) +5);

mem\_out(55 downto 48) <= ram(to\_integer(unsigned(mem\_address)) +6);

mem\_out(63 downto 56) <= ram(to\_integer(unsigned(mem\_address)) +7);

end if;

end process read;

end Behavioral;

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 6: Data Memory Output**

When the write enable (mem\_write) is set, the component takes an input (mem\_in) and stores it in a specified address (mem\_address). When the read enable (mem\_read) is set, the component gives the memory stored at a specified address as an output (mem\_out).