ECGR 3183 Project 5: Control Units

By: Christopher Lowe

4/25/2022

**ALU Control**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity alu\_control is

Port ( ALUOp : in STD\_LOGIC\_VECTOR (1 downto 0);

opcode : in STD\_LOGIC\_VECTOR (10 downto 0);

ALU\_control\_out : out STD\_LOGIC\_VECTOR (3 downto 0));

end alu\_control;

architecture Behavioral of alu\_control is

signal output : std\_logic\_vector(3 downto 0);

begin

process(ALUOp, opcode) is

begin

case ALUOp is

when "00" => output <= "0010"; --"add" for LDUR/STUR

when "01" => output <= "0111"; --pass input b

when "10" => --r-type

if (opcode = "10001011000") then output <= "0010"; --add

elsif (opcode = "11001011000") then output <= "0110"; --sub

elsif (opcode = "10001010000") then output <= "0000"; --and

elsif (opcode = "10101010000") then output <= "0001"; --orr

else null;

end if;

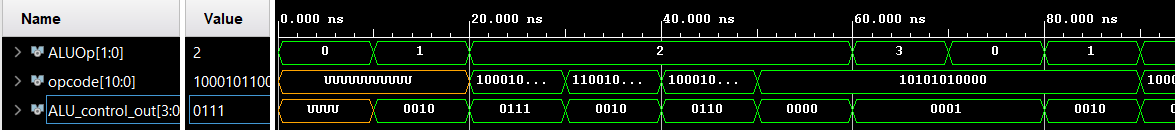
when others => null;

end case;

ALU\_control\_out <= output;

end process;

end Behavioral;



**Figure 1: ALU Control Output**

The ALU control accepts two inputs, a 2-bit ALUOp (ALUOp), and an 11-bit opcode (opcode). Depending on these two inputs, the component gives a 4-bit output (ALU\_control\_out) that will be used by the ALU to determine what operation needs to be performed on the data from the register file. One thing of note is that this component gives the output 1 clock cycle later than the input, even though this component has no clock input

**Main Control**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity main\_control is

Port ( opcode : in STD\_LOGIC\_VECTOR (10 downto 0);

Reg2Loc : inout STD\_LOGIC;

ALUSrc : out STD\_LOGIC;

MemtoReg : inout STD\_LOGIC;

RegWrite : out STD\_LOGIC;

MemRead : out STD\_LOGIC;

MemWrite : out STD\_LOGIC;

Branch : out STD\_LOGIC;

UncondBranch : out STD\_LOGIC;

ALUOp : out STD\_LOGIC\_VECTOR (1 downto 0));

end main\_control;

architecture Behavioral of main\_control is

begin

process (opcode) is

begin

if (opcode = "11111000010") then --LDUR

Reg2Loc <= Reg2Loc;

ALUSrc <= '1';

MemtoReg <= '1';

RegWrite <= '1';

MemRead <= '1';

MemWrite <= '0';

Branch <= '0';

UncondBranch <= '0';

ALUOp <= "00";

elsif (opcode = "11111000000") then --STUR

Reg2Loc <= '1';

ALUSrc <= '1';

MemtoReg <= MemtoReg;

RegWrite <= '0';

MemRead <= '0';

MemWrite <= '1';

Branch <= '0';

UncondBranch <= '0';

ALUOp <= "00";

elsif (opcode(10 downto 5) = "000101") or (opcode(10 downto 4) = "1011010") then --B, CBZ, CBNZ, etc

Reg2Loc <= '1';

ALUSrc <= '0';

MemtoReg <= MemtoReg;

RegWrite <= '0';

MemRead <= '0';

MemWrite <= '0';

Branch <= '1';

UncondBranch <= NOT(opcode(10));

ALUOp <= "01";

else --R-format

Reg2Loc <= '0';

ALUSrc <= '0';

MemtoReg <= '0';

RegWrite <= '1';

MemRead <= '0';

MemWrite <= '0';

Branch <= '0';

UncondBranch <= '0';

ALUOp <= "10";

end if;

end process;

end Behavioral;

Graphical user interface

Description automatically generated

**Figure 2: Main Control Output**

This component takes an 11-bit input (opcode). Once received, it determines which of the eight 1-bit outputs (Reg2Loc, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, UncondBranch) need to be asserted, and also gives another 2-bit output (ALUOp) which is forwarded to the ALU control to determine what operation needs to be taken by the ALU. Which bits are asserted is dependent on whether the instruction is a load, store, branch, or r-type instruction.