Austen Brooks
ECGR 3183 - Control Units

The ALU control takes the 11 bit instruction opcode and the 2 bit aluOp flag from the control as inputs and outputs the the 4 bit ALUcontrol that will be passed to the ALU to determine which operation it will perform.

```
34  entity alu_control is
35      Port ( aluOp : in STD_LOGIC_VECTOR (1 downto 0);
36             opCode : in STD_LOGIC_VECTOR (10 downto 0);
37             aluControl : out STD_LOGIC_VECTOR (3 downto 0));
38  end alu_control;
39
40  architecture Behavioral of alu_control is
41
42  begin
43  process(aluOP, opCode) is
44  begin
45      -- LDUR or STUR
46      if aluOP = "00" then
47          aluControl <= "0010";
48      --CBZ
49      elsif aluOP = "01" then
50          aluControl <= "0111";
51
52      elsif aluOP = "10" then
53          --ADD
54          if opCode = "10001011000" then
55              aluControl <= "0010";
56          --SUB
57          elsif opCode = "11001011000" then
58              aluControl <= "0110";
59          --AND
60          elsif opCode = "10001010000" then
61              aluControl <= "0000";
62          --ORR
63          elsif opCode = "10101010000" then
64              aluControl <= "0001";
65          end if;
66      end if;
67  end process;
68
69  end Behavioral;
70
```

The main control unit takes an 11 bit opcode input from the instruction and will output 8 1 bit flags that will be used in the processor and a 2 bit aluOp flag that will be sent to the ALU. Certain instructions have shorter opcodes, and so only the relevant amount of bits for that opcode will be checked like in CBZ and B instructions. The R-Type instructions have some bits which are don’t cares, but in VHDL the X format for a don’t care didn’t produce the correct result, so I compared each section without the don't cares, and ANDed them together.

```vhdl
entity main_control is
    Port ( opCode : in STD_LOGIC_VECTOR (10 downto 0);
           reg2Loc : out STD_LOGIC;
           aluSrc : out STD_LOGIC;
           memToReg : out 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
    --R type
    if opCode (10 downto 10) = "1" AND opcode (7 downto 4) = "0101" AND opCode (2 downto 0) = "000" then
        reg2Loc <= '0';
        aluSrc <= '0';
        memToReg <= '0';
        regWrite <= '1';
        memRead <= '0';
        memWrite <= '0';
        branch <= '0';
        uncondBranch <= '0';
        aluOP <= "10";
    --LDUR
    elsif opCode = "11111000010" then
        reg2Loc <= 'X';
        aluSrc <= '1';
        memToReg <= '1';
        regWrite <= '1';
        memRead <= '1';
        memWrite <= '0';
        branch <= '0';
        uncondBranch <= '0';
        aluOP <= "00";
    --STUR
```

```
74          aluOP <= "00";
75      --STUR
76      elsif opCode = "11111000000" then
77          reg2Loc <= '1';
78          aluSrc <= '1';
79          memToReg <= 'X';
80          regWrite <= '0';
81          memRead <= '0';
82          memWrite <= '1';
83          branch <= '0';
84          uncondBranch <= '0';
85          aluOP <= "00";
86      --CBZ
87      elsif opCode (10 downto 3) = "10110100" then
88          reg2Loc <= '1';
89          aluSrc <= '0';
90          memToReg <= 'X';
91          regWrite <= '0';
92          memRead <= '0';
93          memWrite <= '0';
94          branch <= '1';
95          uncondBranch <= '0';
96          aluOP <= "01";
97      --B
98      elsif opCode (10 downto 5) = "000101" then
99          reg2Loc <= 'X';
100         aluSrc <= 'X';
101         memToReg <= 'X';
102         regWrite <= '0';
103         memRead <= '0';
104         memWrite <= '0';
105         branch <= '1';
106         uncondBranch <= '1';
107         aluOP <= "XX";
108     end if;
109 end process;
110 end Behavioral;
```