CPE 233: Computer Design and Assembly Language Programming

RAT Assignment #8: Interrupts

By: Roee Landesman and Amir Hashemizad

Black Box Block Diagram

As seen in figure 1, we added an interrupt port to our overarching RAT Wrappe. Figure shows how the interrupt input for the control unit is handled, utilizing a component that will take in the outputs from the control unit and "AND" the result with the interrupt input, this allows us to control when we are able to take in interrupts. Finally, as seen In figure 3, the flags component was upgraded with the to save the state of the flags when we encounter an interrupt.

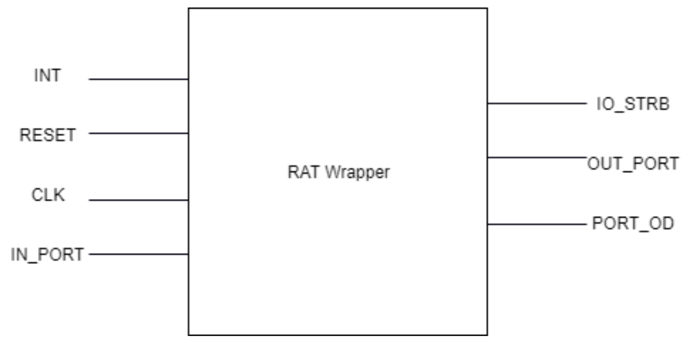


Figure 1: RAT Wrapper Black Box Block Diagram

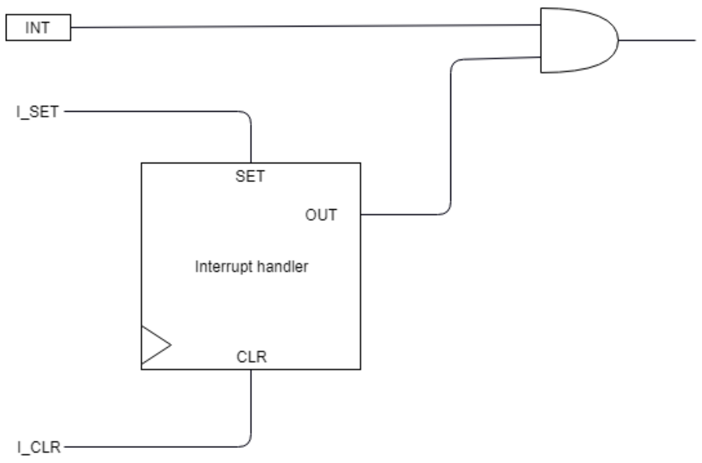


Figure 2: Interrupt Black Box Block Diagram

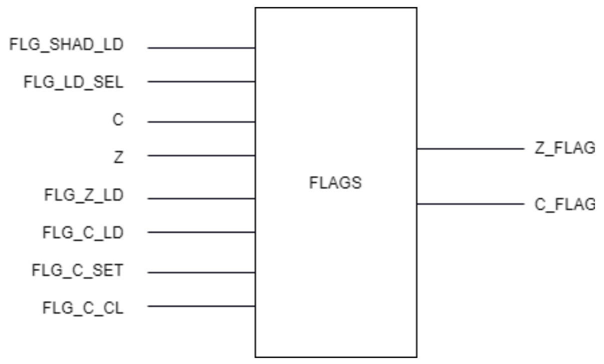


Figure 3: FLAGS Black Box Diagram

Behavior Description

Interrupts signals are external instructions that enter the MCU at an undetermined time, and cause the processor to execute a different set of instructions while "remembering" where it was before the input entered. To accomplish this, we needed to extend our FLAGS component to include shadow flags – a way to store flag data—and include a new schematic which would only allow interrupts in when the interrupt enable is on. Finally, to accomplish the hardware assignment linked to this project, we added a button denounce to prevent multiple inputs.

Structural Design

Figure 4 shows the new RAT Wrapper structural design including the Interrupt controller and debounce components. Figure 5 you can see our interrupt controller showing the different inputs and outputs, but also how the internal signals communicate with different parts of the controller. Figure 6 shows the new FLAGS component which includes two mux's, and two shadow registers.

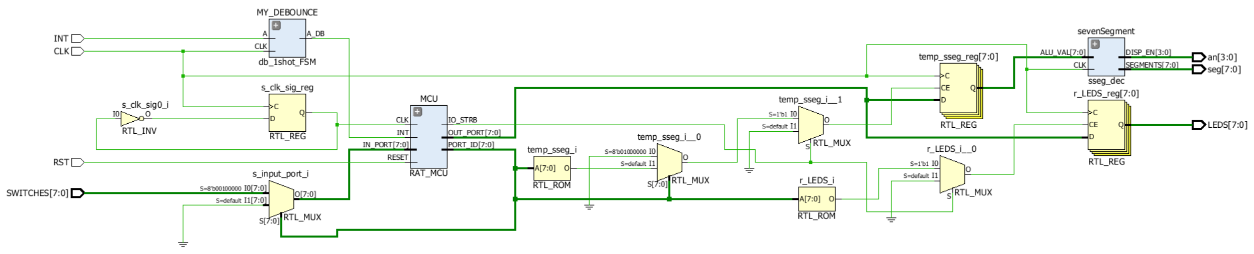


Figure 4: RAT Wrapper Structural Design

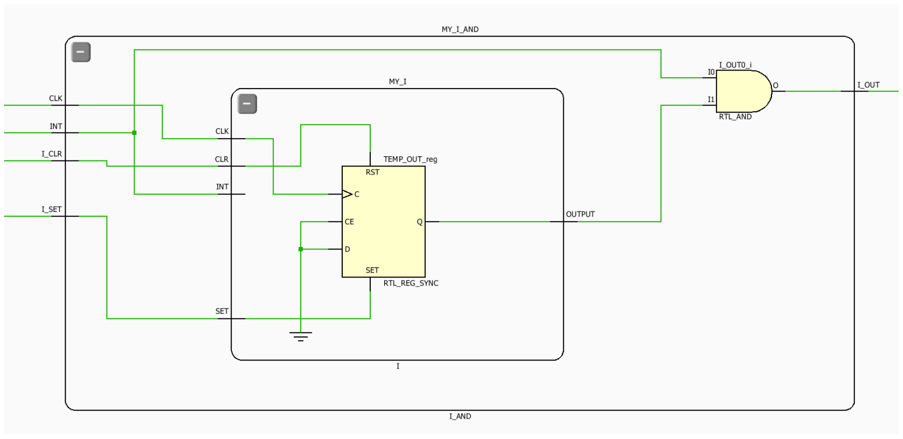


Figure 5: Interrupt Controller Structural Design

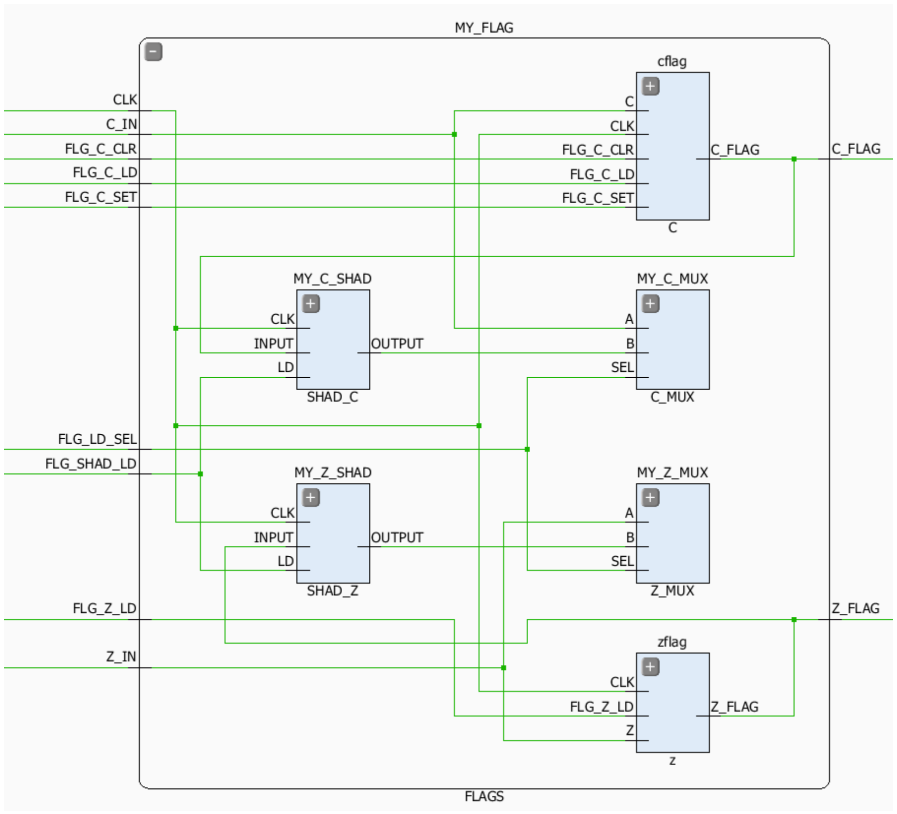
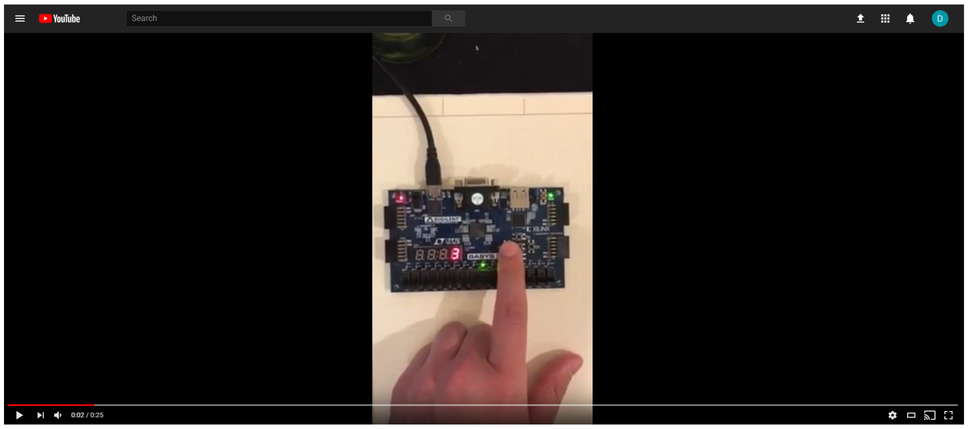


Figure 6: Flags Structural Design

Verification

Below is a link to the demonstration video showing the project working on hardware, when the LED in the middle of the board is lit it indicates that interrupts are enables, while this is happening pressing the left button will create an interrupt and increase the number on the seven-segment display, when the middle LED is off interrupts are disables. After you reach 20 interrupts they become disabled until you hit the reset button (right button)



https://www.youtube.com/watch?v=N85rzZu8d9I

VHDL Source Code

Below is the source code for the control unit (which has the addition of interrupt controls and a new state for interrupts), the interrupt controller, the flag wrapper (which has the addition of muxs' and shadow registers), and the shadow register

```vhdl
-----------------------------------------------------------------------------
-- Company: Cal Poly
-- Engineer: Roee Landesman and Amir Hashemizad
--
-- Create Date: 02/08/2018 11:14:28 PM
-- Module Name: control_unit - Behavioral
-- Project Name: RAT Control Unit
-- Target Devices: Basys3
----------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CONTROL_UNIT is
    Port(
        CLK          : in STD_LOGIC;
        C            : in STD_LOGIC;
        Z            : in STD_LOGIC;
        INT          : in STD_LOGIC;
        RESET        : in STD_LOGIC;
        OPCODE_HI_5  : in STD_LOGIC_VECTOR(4 downto 0);
        OPCODE_LO_2  : in STD_LOGIC_VECTOR(1 downto 0);
        RST          : out STD_LOGIC;
        PC_LD        : out STD_LOGIC;
        PC_INC       : out STD_LOGIC;
        PC_MUX_SEL   : out STD_LOGIC_VECTOR(1 downto 0);
        SP_LD        : out STD_LOGIC;
        SP_INCR      : out STD_LOGIC;
        SP_DECR      : out STD_LOGIC;
        SCR_DATA_SEL: out STD_LOGIC;
        SCR_WE       : out STD_LOGIC;
        SCR_ADDR_SEL: out STD_LOGIC_VECTOR(1 downto 0);
        RF_WR        : out STD_LOGIC;
        RF_WR_SEL    : out STD_LOGIC_VECTOR(1 downto 0);
        ALU_SEL      : out STD_LOGIC_VECTOR(3 downto 0);
        ALU_OPY_SEL  : out STD_LOGIC;
        FLG_C_LD     : out STD_LOGIC;
        FLG_C_SET    : out STD_LOGIC;
        FLG_C_CLR    : out STD_LOGIC;
        FLG_Z_LD     : out STD_LOGIC;
        FLG_SHAD_LD  : out STD_LOGIC;
        FLG_LD_SEL   : out STD_LOGIC;
        I_SET        : out STD_LOGIC;
        I_CLR        : out STD_LOGIC;
        IO_STRB      : out STD_LOGIC);
end CONTROL_UNIT;

architecture Behavioral of CONTROL_UNIT is
```

```
    type state_type is (ST_init, ST_fetch, ST_execute, ST_interrupt);
    signal PS, NS           : state_type;
    signal sig_OPCODE_7     : std_logic_vector (6 downto 0);

begin

    sig_OPCODE_7 <= OPCODE_HI_5 & OPCODE_LO_2;

    next_state: process(CLK, NS, RESET)
    begin
        if (RESET = '1') then
            PS <= ST_init;
        elsif rising_edge(CLK) then
            PS <= NS;
        end if;
    end process next_state;

    main: process (INT, sig_OPCODE_7, PS, NS, C, Z)
    begin

        I_SET           <= '0';
        I_CLR           <= '0';
        ALU_OPY_SEL     <= '0';
        ALU_SEL         <= "0000";
        SP_LD           <= '0';
        SP_INCR         <= '0';
        SP_DECR         <= '0';
        SCR_WE          <= '0';
        SCR_ADDR_SEL    <= "00";
        SCR_DATA_SEL    <= '0';
        RST             <= '0';
        PC_LD           <= '0';
        PC_INC          <= '0';
        PC_MUX_SEL      <= "00";
        RF_WR           <= '0';
        RF_WR_SEL       <= "00";
        FLG_C_SET       <= '0';
        FLG_C_CLR       <= '0';
        FLG_C_LD        <= '0';
        FLG_Z_LD        <= '0';
        FLG_LD_SEL      <= '0';
        FLG_SHAD_LD     <= '0';
        IO_STRB         <= '0';

        case PS is

            when ST_init =>
                RST <= '1';
                NS <= ST_fetch;
            when ST_fetch =>
                NS <= ST_execute;
                PC_INC <= '1';
            when ST_interrupt =>
                NS <= ST_fetch;
                PC_LD <= '1';
                PC_MUX_SEL <= "10";
        FLG_SHAD_LD <= '1';
        FLG_C_CLR <= '1';
        SP_DECR <= '1';
        SCR_WE <= '1';
        SCR_DATA_SEL <= '1';
        SCR_ADDR_SEL <= "11";
        I_CLR <= '1';
    when ST_execute =>
        if (INT = '0') then
            NS <= ST_fetch;
        else
            NS <= ST_interrupt;
        end if;
        PC_INC <= '0';
            case sig_OPCODE_7 is
                when "0000100" => --ADD (reg,reg)
                    RF_WR <= '1';
                    RF_WR_SEL  <= "00";
                    ALU_OPY_SEL <= '0';
                    ALU_SEL <= "0000";
                    FLG_C_LD   <= '1';
                    FLG_Z_LD    <= '1';
                when "1010000" | "1010001" | "1010010" | "1010011" =>
                    RF_WR <= '1';
                    RF_WR_SEL <= "00";
                    ALU_OPY_SEL <= '1';
                    ALU_SEL <= "0000";
                    FLG_C_LD <= '1';
                    FLG_Z_LD <= '1';
                when "0000101" => --ADDC (reg,reg)
                    RF_WR <= '1';
                    RF_WR_SEL  <= "00";
                    ALU_OPY_SEL <= '0';
                    ALU_SEL <= "0001";
                    FLG_C_LD   <= '1';
                    FLG_Z_LD    <= '1';
                when "1010100" | "1010101" | "1010110" | "1010111" =>
                    RF_WR <= '1';
                    RF_WR_SEL <= "00";
                    ALU_OPY_SEL <= '1';
                    ALU_SEL <= "0001";
                    FLG_C_LD <= '1';
                    FLG_Z_LD <= '1';
                when "0000000" => --AND (reg,reg)
                    RF_WR       <= '1';
                    RF_WR_SEL   <= "00";
                    ALU_OPY_SEL <= '0';
                    ALU_SEL     <= "0101";
                    FLG_Z_LD    <= '1';
                when "1000000"|"1000001"|"1000010"|"1000011" =>
                    RF_WR       <= '1';
                    RF_WR_SEL   <= "00";
                    ALU_OPY_SEL <= '1';
                    ALU_SEL     <= "0101";
                    FLG_Z_LD    <= '1';
                when "0100100" => --ASR
                    RF_WR       <= '1';
        RF_WR_SEL  <= "00";
        ALU_OPY_SEL <= '0';
        ALU_SEL <= "1101";
        FLG_C_LD <= '1';
        FLG_Z_LD <= '1';
    when "0010101" => --BRCC
        if (C = '0') then
            PC_LD <= '1';
            PC_MUX_SEL <= "00";
        end if ;
    when "0010100" => --BRCS
        if (C = '1') then
            PC_LD <= '1';
            PC_MUX_SEL <= "00";
        end if ;
    when "0010010" => -- BREQ
        if (Z = '1') then
             PC_LD <= '1';
            PC_MUX_SEL <= "00";
        end if ;
    when "0010011" => --BRNE
        if (Z = '0') then
            PC_LD <= '1';
            PC_MUX_SEL <= "00";
        end if ;
    when "0010000" => --BRN
        PC_LD       <= '1';
        PC_MUX_SEL  <= "00";
    when "0010001" => --CALL
        PC_LD <= '1';
        PC_MUX_SEL <= "00";
        SCR_WE <= '1';
        SCR_DATA_SEL <= '1';
        SCR_ADDR_SEL <= "11";
        SP_DECR <= '1';
    when "0110000" => --CLC
        FLG_C_CLR   <= '1';
    when "0110101" => --CLI
        I_CLR       <= '1';
    when "0001000" => --CMP (reg,reg)
        ALU_OPY_SEL <= '0';
        ALU_SEL <= "0100";
        FLG_C_LD <= '1';
        FLG_Z_LD <= '1';
    when "1100000" | "1100001" | "1100010" | "1100011" =>
        ALU_OPY_SEL <= '1';
        ALU_SEL <= "0100";
        FLG_C_LD <= '1';
        FLG_Z_LD <= '1';
    when "0000010" => --EXOR (reg,reg)
        RF_WR       <= '1';
        RF_WR_SEL   <= "00";
        ALU_OPY_SEL <= '0';
        ALU_SEL     <= "0111";
        FLG_Z_LD    <= '1';
```

```
when "1001000"|"1001001"|"1001010"|"1001011" => --
    RF_WR       <= '1';
    RF_WR_SEL   <= "00";
    ALU_OPY_SEL <= '1';
    ALU_SEL     <= "0111";
    FLG_Z_LD    <= '1';
when "1100100"|"1100101"|"1100110"|"1100111" => --in
    RF_WR       <= '1';
    RF_WR_SEL   <= "11";
when "0001010" => --LD (reg,reg)
    RF_WR <= '1';
    RF_WR_SEL <= "01";
    SCR_ADDR_SEL <= "00";
when "1110000" | "1110001" | "1110010" | "1110011" =>
    RF_WR <= '1';
    RF_WR_SEL <= "01";
    SCR_ADDR_SEL <= "01";
when "0001011" => --ST (reg,reg)
    SCR_ADDR_SEL <= "00";
    SCR_WE <= '1';
when "1110100" | "1110101" | "1110110" | "1110111" =>
    SCR_ADDR_SEL <= "01";
    SCR_WE <= '1';
when "0100000" => --LSL
    RF_WR       <= '1';
    RF_WR_SEL  <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "1001";
    FLG_C_LD <= '1';
    FLG_Z_LD <= '1';
  when "0100001" => --LSR
    RF_WR       <= '1';
    RF_WR_SEL  <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "1010";
    FLG_C_LD <= '1';
    FLG_Z_LD <= '1';
when "0001001" => --MOV (reg,reg)
    RF_WR       <= '1';
    RF_WR_SEL   <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL     <= "1110";
when "1101100"|"1101101"|"1101110"|"1101111" => --MOV
    RF_WR       <= '1';
    RF_WR_SEL   <= "00";
    ALU_OPY_SEL <= '1';
    ALU_SEL     <= "1110";
when "0000001" => --or (reg,reg)
    RF_WR <= '1';
    RF_WR_SEL <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "0110";
    FLG_Z_LD <= '1';
when "1000100" | "1000101" | "1000110" | "1000111" =>
    RF_WR <= '1';
    RF_WR_SEL <= "00";
    ALU_OPY_SEL <= '1';
    ALU_SEL <= "0110";
    FLG_Z_LD <= '1';
when "1101000"|"1101001"|"1101010"|"1101011" => --OUT
    IO_STRB     <= '1';
when "0100110" => --POP
    RF_WR <= '1';
    RF_WR_SEL <= "01";
    SCR_ADDR_SEL <= "10";
    SP_INCR <= '1';
when "0100101" => --PUSH
    SCR_WE <= '1';
    SCR_ADDR_SEL <= "11";
    SP_DECR <= '1';
when "0110010" => --RET
    PC_LD <= '1';
    PC_MUX_SEL <= "01";
    SCR_ADDR_SEL <= "10";
    SP_INCR <= '1';
when "0110110" => --RETID
    PC_LD <= '1';
    PC_MUX_SEL <= "01";
    SCR_ADDR_SEL <= "10";
    SP_INCR <= '1';
    I_CLR <= '1';
    FLG_LD_SEL <= '1';
    FLG_SHAD_LD <= '1';
when "0110111" => --RETIE
    PC_LD <= '1';
    PC_MUX_SEL <= "01";
    SCR_ADDR_SEL <= "10";
    SP_INCR <= '1';
    I_SET <= '1';
    FLG_LD_SEL <= '1';
    FLG_SHAD_LD <= '1';
when "0100010" => --ROL
    RF_WR       <= '1';
    RF_WR_SEL  <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "1011";
    FLG_C_LD <= '1';
    FLG_Z_LD <= '1';
when "0100011" => --ROR
    RF_WR       <= '1';
    RF_WR_SEL  <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "1100";
    FLG_C_LD <= '1';
    FLG_Z_LD <= '1';
when "0000110" => --SUB(reg,reg)
    RF_WR <= '1';
    RF_WR_SEL  <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "0010";
    FLG_C_LD   <= '1';
    FLG_Z_LD    <= '1';
when "0110001" => --SEC
    FLG_C_SET   <= '1';
```

```
when "0110100" => --SEI
    I_SET <= '1';
when "1011000" | "1011001" | "1011010" | "1011011" =>
    RF_WR <= '1';
    RF_WR_SEL <= "00";
    ALU_OPY_SEL <= '1';
    ALU_SEL <= "0010";
    FLG_C_LD <= '1';
    FLG_Z_LD <= '1';
when "0000111" => --SUBC (reg,reg)
    RF_WR <= '1';
    RF_WR_SEL  <= "00";
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "0011";
    FLG_C_LD   <= '1';
    FLG_Z_LD    <= '1';
when "1011100" | "1011101" | "1011110" | "1011111" =>
    RF_WR <= '1';
    RF_WR_SEL <= "00";
    ALU_OPY_SEL <= '1';
    ALU_SEL <= "0011";
    FLG_C_LD <= '1';
    FLG_Z_LD <= '1';
when "0000011" => --TEST (reg,reg)
    ALU_OPY_SEL <= '0';
    ALU_SEL <= "1000";
    FLG_Z_LD <= '1';
when "1001100" | "1001101" | "1001110" | "1001111" =>
    ALU_OPY_SEL <= '1';
    ALU_SEL <= "1000";
    FLG_Z_LD <= '1';
when "0101000" => --WSP
    SP_LD <= '1';
when others =>
    NS          <= ST_fetch;
    I_SET           <= '0';
    I_CLR           <= '0';
    ALU_OPY_SEL     <= '0';
    ALU_SEL         <= "0000";
    SP_LD           <= '0';
    SP_INCR         <= '0';
    SP_DECR         <= '0';
    SCR_WE          <= '0';
    SCR_ADDR_SEL    <= "00";
    SCR_DATA_SEL    <= '0';
    RST             <= '0';
    PC_LD           <= '0';
    PC_INC          <= '0';
    PC_MUX_SEL      <= "00";
    RF_WR           <= '0';
    RF_WR_SEL       <= "00";
    FLG_C_SET       <= '0';
    FLG_C_CLR       <= '0';
    FLG_C_LD        <= '0';
    FLG_Z_LD        <= '0';
    FLG_LD_SEL      <= '0';
    FLG_SHAD_LD     <= '0';
```

```
                              IO_STRB          <= '0';

                        end case;
                end case;
        end process main;
end Behavioral;

------------------------------------------------------------------------------
-- Company: Cal Poly
-- Engineer: Roee Landesman and Amir Hashemizad
--
-- Create Date: 02/08/2018 11:14:28 PM
-- Module Name: Interrupt Controller - Behavioral
-- Project Name:
-- Target Devices: Basys3
------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity I is
    Port(
        INT      : in STD_LOGIC;
        SET      : in STD_LOGIC;
        CLR      : in STD_LOGIC;
        CLK      : in STD_LOGIC;
        OUTPUT   : out STD_LOGIC);
end I;
architecture Behavioral of I is

    signal TEMP_OUT : std_logic := '0';

begin

    process(SET, CLR, CLK)
    begin
        if (rising_edge(CLK)) then
            if (SET = '1') then
                TEMP_OUT <= '1';
            elsif (CLR = '1') then
                TEMP_OUT <= '0';
            else
                TEMP_OUT <= TEMP_OUT;
            end if;
        end if;
    end process;
    OUTPUT <= TEMP_OUT;
end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company: Cal Poly
-- Engineer: Roee Landesman and Amir Hashemizad
--
-- Create Date: 02/08/2018 11:14:28 PM
-- Module Name: FLAGS - Behavioral
-- Target Devices: Basys3
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FLAGS is
    Port(
        CLK         : in STD_LOGIC;
        C_IN        : in STD_LOGIC;
        Z_IN        : in STD_LOGIC;
        FLG_C_SET   : in STD_LOGIC;
        FLG_C_CLR   : in STD_LOGIC;
        FLG_C_LD    : in STD_LOGIC;
        FLG_Z_LD    : in STD_LOGIC;
        FLG_LD_SEL  : in STD_LOGIC;
        FLG_SHAD_LD : in STD_LOGIC;
        C_FLAG      : out STD_LOGIC;
        Z_FLAG      : out STD_LOGIC);
end FLAGS;

architecture Behavioral of FLAGS is

    component C
        port(
            CLK         : in std_logic;
            FLG_C_LD    : in std_logic;
            FLG_C_SET   : in std_logic;
            FLG_C_CLR   : in std_logic;
            C           : in std_logic;
            C_FLAG      : out std_logic);
    end component;

    component C_MUX
        port(
            A           : in std_logic;
            B           : in std_logic;
            SEL         : in std_logic;
            MUX_OUT     : out std_logic);
    end component;

    component SHAD_C
        port(
            LD          : in std_logic;
            INPUT       : in std_logic;
            CLK         : in std_logic;
            OUTPUT     : out std_logic);
    end component;
```

```vhdl
    component Z
        port(
            CLK         : in std_logic;
            FLG_Z_LD    : in std_logic;
            Z           : in std_logic;
            Z_FLAG      : out std_logic);
    end component;

    component Z_MUX
        port(
            A           : in std_logic;
            B           : in std_logic;
            SEL         : in std_logic;
            MUX_OUT     : out std_logic);
    end component;

    component SHAD_Z
        port(
            LD          : in std_logic;
            INPUT       : in std_logic;
            CLK         : in std_logic;
            OUTPUT     : out std_logic);
    end component;

    signal C_MUX_OUT    : std_logic := '0';
    signal Z_MUX_OUT    : std_logic := '0';
    signal C_OUT        : std_logic := '0';
    signal Z_OUT        : std_logic := '0';
    signal C_SHAD_OUT   : std_logic := '0';
    signal Z_SHAD_OUT   : std_logic := '0';

begin

    zflag: Z port map(
        CLK => CLK,
        FLG_Z_LD => FLG_Z_LD,
        Z => Z_IN,
        Z_FLAG => Z_OUT);

    MY_Z_MUX: Z_MUX port map(
        A => Z_IN,
        B => Z_SHAD_OUT,
        SEL => FLG_LD_SEL,
        MUX_OUT => Z_MUX_OUT);

    MY_Z_SHAD: SHAD_Z port map(
        LD => FLG_SHAD_LD,
        INPUT => Z_OUT,
        CLK => CLK,
        OUTPUT => Z_SHAD_OUT);

    cflag: C port map(
        CLK => CLK,
        FLG_C_LD => FLG_C_LD,
        FLG_C_SET => FLG_C_SET,
        FLG_C_CLR => FLG_C_CLR,
        C => C_IN,
        C_FLAG => C_OUT);

    MY_C_MUX: C_MUX port map(
        A => C_IN,
        B => C_SHAD_OUT,
        SEL => FLG_LD_SEL,
        MUX_OUT => C_MUX_OUT);

    MY_C_SHAD: SHAD_C port map(
        LD => FLG_SHAD_LD,
        INPUT => C_OUT,
        CLK => CLK,
        OUTPUT => C_SHAD_OUT);

    process(Z_OUT, C_OUT)
    begin
        Z_FLAG <= Z_OUT;
        C_FLAG <= C_OUT;
    end process;

end Behavioral;
```

---Shadow Flags (Z)---

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SHAD_Z is
    Port (
        LD       : in STD_LOGIC;
        INPUT    : in STD_LOGIC;
        CLK      : in STD_LOGIC;
        OUTPUT   : out STD_LOGIC);
end SHAD_Z;

architecture Behavioral of SHAD_Z is

    signal TEMP_OUT : std_logic := '0';

begin

    process(LD, INPUT, CLK)
    begin
        if (rising_edge(CLK)) then
            if (LD = '1') then
                TEMP_OUT <= INPUT;
            else
                TEMP_OUT <= TEMP_OUT;
            end if;
        end if;
    end process;
    OUTPUT <= TEMP_OUT;

end Behavioral;
```