Experiment 7 - It's Alive!
Brandon Kelley and Joseph Prachar
CPE 233-04
February 5, 2016

Objective:

The purpose of this lab is to learn how the RAT CPU sends signals to different modules within the architecture to create a state machine that controls the fetching and executing of program instructions.

Procedure:

Part 1:

1. Recreate the RAT CPU architecture diagram using an electronic tool. The diagram will be useful for building the CPU and ensuring all signals are connected to modules correctly.

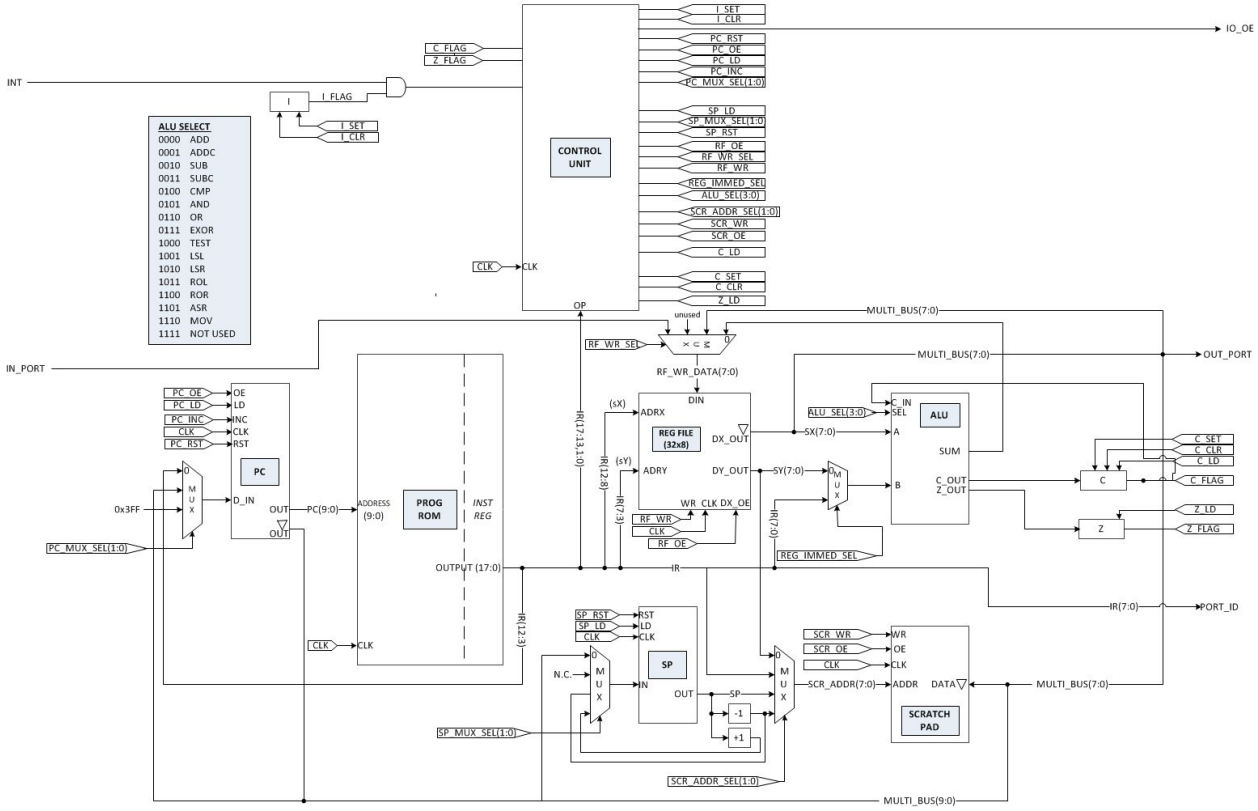

Figure 1: Archetecture diagram provided in class (used to create CPU)

Part 2:

1. Complete the provided control signal table worksheet with the signals that trigger the IN, MOV, EXOR, OUT, and BRN instructions, as well as the fetch and reset states.

2. Use the same approach to fill out the ControlUnit.vhd skeleton with signals as shown in Figure 2.

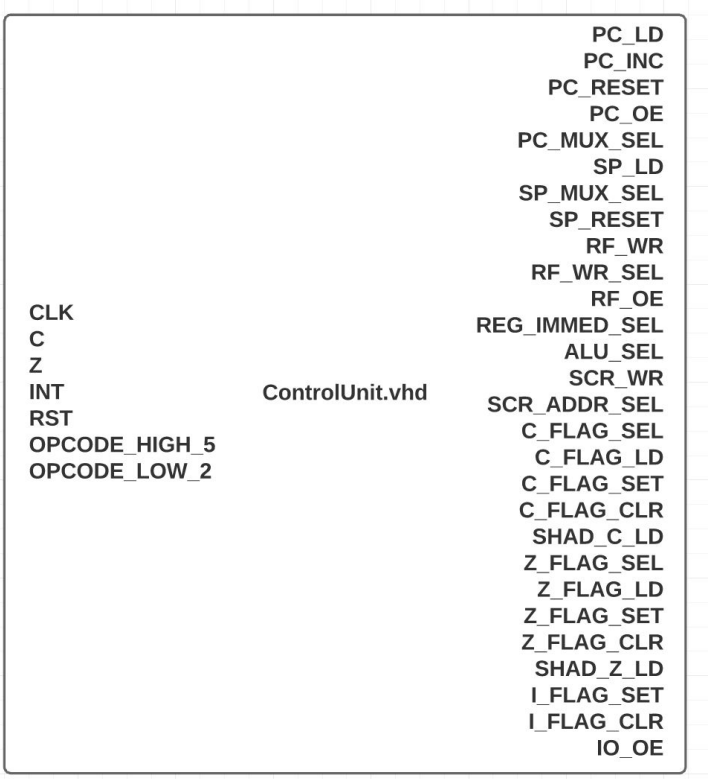

Figure 2: The ControlUnit BBD

Part 3:

1. Create a RAT_CPU module that connects the control unit to the program counter, prog_rom, register file and ALU.

Figure 3: The FlagReg BBD

2. Implement a system like Figure 3 for the zero and carry flag registers.
3. Create signals within the CPU for the flags and assign them with the outputs of the ALU.

Part 4:

1. Create a prog_rom.vhd file using the RAT simulator and the following program.

```
.EQU SWITCH_PORT = 0x20 ; port for switch input
.EQU LED_PORT    = 0x40 ; port for LED output

.DSEG
.ORG 0x00
.CSEG
.ORG 0x10

main:   IN    r10,SWITCH_PORT
        MOV   r11,0xFF
        EXOR  r10,r11
        OUT   r10,LED_PORT
        BRN   main
```

Part 5:

1. Create a wrapper file for the RAT CPU to interface it with the Basys 3 board.
2. Create a testbench and analyze the CPU in iSim.
3. Create a constraints file for the wrapper and load the CPU onto the board.

Testing:

Testing the RAT_CPU was fairly simple. All modules worked as expected and just needed to be connected. There were minor problems/typos in the CPU module and the control unit as described in Table 1. These errors were found by analyzing the timing diagram (Figure 4) extensively for incorrect signals. Once the design simulated correctly, a TA checked off correct functionality performing on the Nexys board.

Problem	Solution
CPU not init-ing correctly	Fix control unit to reset on '1' instead of '0'
CPU still not init-ing correctly	Fix counter to reset value when RST = '1' and not mask output with zeros
Counter IMMED input not connected to instruction signal	Connect IMMED input to correct bits of instruction signal
CPU not outputting correctly	Connect CPU outputs to correct signals inside CPU
No output on IO_OE	Control unit was using RF_OE instead of IO_OE. Replace RF_OE with IO_OE in OUT instruction.
IO_OE not outputting correctly	Turns out both RF_OE and IO_OE are needed in the OUT instruction. So put RF_OE back.

Table 1: Problems discovered while debugging and their solutions

Figure 4: Timing diagram of RAT CPU running assembly program from part 4

Conclusions:

Brandon Kelley:

In this lab, the basic structure of the RAT CPU was completed. Main components, such as the program counter and ALU, were connected into the ControlUnit so that it became possible to execute a program contained in prog_rom.vhd. At this point, support for only a few instructions has been included, so that will need to be added for a more complete CPU. *However, the lab was very useful in giving the users an understanding of how the CPU is composed of internal modules and signals.*

Joseph Prachar:

This experiment was extremely helpful in understanding the way that the RAT CPU operates. Referring to the architecture diagram while connecting all of the previously created modules gave great reinforcement of the overall layout of the CPU. This also helped debugging; once a signal was identified to be faulty, the architecture diagram was analyzed as well as the specific modules in question to find the problem and solution. This gave the participants of the lab much experience debugging a large, multi level design. *The biggest word of advice to future lab participants would be to focus on creating the smaller modules correctly the first time. Once Experiment 7 is reached it is a lot of work to debug every single module. One needs to be confident in the way that each module works before starting this experiment, otherwise debugging will be lengthy and tedious.*

RAT_CPU.vhd:

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

entity RAT_CPU is
    Port (
        IN_PORT : in STD_LOGIC_VECTOR (7 downto 0);
        RST : in STD_LOGIC;
        INT_IN : in STD_LOGIC;
        CLK : in STD_LOGIC;
        OUT_PORT : out STD_LOGIC_VECTOR (7 downto 0);
        PORT_ID : out STD_LOGIC_VECTOR (7 downto 0);
        IO_OE : out STD_LOGIC);
end RAT_CPU;

architecture Behavioral of RAT_CPU is
    component ControlUnit
        Port (
            CLK           : in   STD_LOGIC;
            C             : in   STD_LOGIC;
            Z             : in   STD_LOGIC;
            INT           : in   STD_LOGIC;
            RST           : in   STD_LOGIC;
            OPCODE_HI_5   : in   STD_LOGIC_VECTOR (4 downto 0);
            OPCODE_LO_2   : in   STD_LOGIC_VECTOR (1 downto 0);

            PC_LD         : out  STD_LOGIC;
            PC_INC        : out  STD_LOGIC;
            PC_RESET      : out  STD_LOGIC;
            PC_OE         : out  STD_LOGIC;
            PC_MUX_SEL    : out  STD_LOGIC_VECTOR (1 downto 0);
            SP_LD         : out  STD_LOGIC;
            SP_MUX_SEL    : out  STD_LOGIC_VECTOR (1 downto 0);
            SP_RESET      : out  STD_LOGIC;
            RF_WR         : out  STD_LOGIC;
            RF_WR_SEL     : out  STD_LOGIC_VECTOR (1 downto 0);
            RF_OE         : out  STD_LOGIC;
            REG_IMMED_SEL : out  STD_LOGIC;
            ALU_SEL       : out  STD_LOGIC_VECTOR (3 downto 0);
            SCR_WR        : out  STD_LOGIC;
            SCR_OE        : out  STD_LOGIC;
            SCR_ADDR_SEL  : out  STD_LOGIC_VECTOR (1 downto 0);
            C_FLAG_SEL    : out  STD_LOGIC_VECTOR (1 downto 0);
            C_FLAG_LD     : out  STD_LOGIC;
            C_FLAG_SET    : out  STD_LOGIC;
            C_FLAG_CLR    : out  STD_LOGIC;
            SHAD_C_LD     : out  STD_LOGIC;
            Z_FLAG_SEL    : out  STD_LOGIC_VECTOR (1 downto 0);
```

```
        Z_FLAG_LD     : out  STD_LOGIC;
        Z_FLAG_SET    : out  STD_LOGIC;
        Z_FLAG_CLR    : out  STD_LOGIC;
        SHAD_Z_LD     : out  STD_LOGIC;
        I_FLAG_SET    : out  STD_LOGIC;
        I_FLAG_CLR    : out  STD_LOGIC;
        IO_OE         : out  STD_LOGIC);
end component;

component counter
    Port (  FROM_IMMED : in STD_LOGIC_VECTOR (9 downto 0);
        FROM_STACK : in STD_LOGIC_VECTOR (9 downto 0);
        INTERRUPT : in STD_LOGIC_VECTOR (9 downto 0);
        PC_MUX_SEL : in STD_LOGIC_VECTOR (1 downto 0);
        PC_OE : in STD_LOGIC;
        PC_LD : in STD_LOGIC;
        PC_INC : in STD_LOGIC;
        RST : in STD_LOGIC;
        CLK : in STD_LOGIC;
        PC_COUNT : out STD_LOGIC_VECTOR (9 downto 0);
        PC_TRI : out STD_LOGIC_VECTOR (9 downto 0));
end component;

component prog_rom
    Port (  ADDRESS : in std_logic_vector(9 downto 0);
        INSTRUCTION : out std_logic_vector(17 downto 0);
        CLK : in std_logic);
end component;

component RegisterFile
    Port (  D_IN   : in     STD_LOGIC_VECTOR (7 downto 0);
        DX_OUT : out  STD_LOGIC_VECTOR (7 downto 0);
        DY_OUT : out    STD_LOGIC_VECTOR (7 downto 0);
        ADRX   : in     STD_LOGIC_VECTOR (4 downto 0);
        ADRY   : in     STD_LOGIC_VECTOR (4 downto 0);
        DX_OE  : in     STD_LOGIC;
        WE     : in     STD_LOGIC;
        CLK    : in     STD_LOGIC);
 end component;

 component alu
    Port (  A : in STD_LOGIC_VECTOR (7 downto 0);
        B : in STD_LOGIC_VECTOR (7 downto 0);
        C_IN : in STD_LOGIC;
        Sel : in STD_LOGIC_VECTOR (3 downto 0);
        SUM : out STD_LOGIC_VECTOR (7 downto 0);
        C_FLAG : out STD_LOGIC;
        Z_FLAG : out STD_LOGIC);
```

```
    end component;

    component FlagReg
       Port (  IN_FLAG  : in  STD_LOGIC;
           LD       : in  STD_LOGIC;
           SET      : in  STD_LOGIC;
           CLR      : in  STD_LOGIC;
           CLK      : in  STD_LOGIC;
           OUT_FLAG : out  STD_LOGIC);
    end component;

    signal MULTI_BUS : STD_LOGIC_VECTOR (9 downto 0);

    -- Program counter signals
    signal PC_MUX_SEL : STD_LOGIC_VECTOR (1 downto 0);
    signal PC_OE, PC_LD, PC_INC, PC_RST : STD_LOGIC;
    signal PC_COUNT : STD_LOGIC_VECTOR (9 downto 0);

    -- Prog-rom signal
    signal Instruction : STD_LOGIC_VECTOR (17 downto 0);

    -- Register file signals
    signal RF_DATA_IN : STD_LOGIC_VECTOR (7 downto 0);
    signal RF_WR_SEL : STD_LOGIC_VECTOR (1 downto 0);
    signal RF_OUT_X, RF_OUT_Y : STD_LOGIC_VECTOR (7 downto 0);
    signal RF_OE, RF_WE : STD_LOGIC;

    -- ALU signals
    signal ALU_IN_B, ALU_OUT : STD_LOGIC_VECTOR (7 downto 0);
    signal ALU_SEL : STD_LOGIC_VECTOR (3 downto 0);
    signal REG_IMMED_SEL : STD_LOGIC;
    signal ALU_OUT_C, ALU_OUT_Z : STD_LOGIC;

    -- C Flag signals
    signal C_FLAG : STD_LOGIC;
    signal C_SET, C_CLR, C_LD : STD_LOGIC;

    -- Z Flag signals
    signal Z_FLAG, Z_LD : STD_LOGIC;

begin
    control : controlUnit PORT MAP (CLK, C_FLAG, Z_FLAG, INT_IN, RST,
Instruction (17 downto 13), Instruction (1 downto 0),
        PC_LD, PC_INC, PC_RST, PC_OE, PC_MUX_SEL,
        open, open, open,
        RF_WE, RF_WR_SEL, RF_OE, REG_IMMED_SEL,
        ALU_SEL,
        open, open, open,
```

```
        open, C_LD, C_SET, C_CLR, open,
        open, Z_LD, open, open, open,
        open, open, IO_OE);

    pc : counter PORT MAP (Instruction (12 downto 3), Instruction (12 downto
3), Instruction (12 downto 3), PC_MUX_SEL, PC_OE, PC_LD, PC_INC, PC_RST, CLK,
PC_COUNT, open);
    progRom : prog_rom PORT MAP (PC_COUNT, Instruction, CLK);

    RF_DATA_IN <= ALU_OUT                when RF_WR_SEL = "00"
             else MULTI_BUS (7 downto 0) when RF_WR_SEL = "01"
             else IN_PORT                when RF_WR_SEL = "11"
             else (others => '0');
    regFile : RegisterFile PORT MAP (RF_DATA_IN, RF_OUT_X, RF_OUT_Y,
Instruction (12 downto 8), Instruction (7 downto 3), RF_OE, RF_WE, CLK);
    MULTI_BUS (7 downto 0) <= RF_OUT_X;

    ALU_IN_B <= RF_OUT_Y when REG_IMMED_SEL = '0'
           else Instruction (7 downto 0);
    aluMod : alu PORT MAP (RF_OUT_X, ALU_IN_B, C_FLAG, ALU_SEL, ALU_OUT,
ALU_OUT_C, ALU_OUT_Z);
    cFlag : FlagReg PORT MAP (ALU_OUT_C, C_LD, C_SET, C_CLR, CLK, C_FLAG);
    zFlag : FlagReg PORT MAP (ALU_OUT_Z, Z_LD, '0', '0', CLK, Z_FLAG);

    PORT_ID <= Instruction (7 downto 0);
    OUT_PORT <= MULTI_BUS (7 downto 0);

end Behavioral;
```

Constraints File:

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports CLK]
    set_property IOSTANDARD LVCMOS33 [get_ports CLK]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports CLK]


### Switches
set_property PACKAGE_PIN V17 [get_ports {SWITCHES[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[0]}]
set_property PACKAGE_PIN V16 [get_ports {SWITCHES[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[1]}]
set_property PACKAGE_PIN W16 [get_ports {SWITCHES[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[2]}]
set_property PACKAGE_PIN W17 [get_ports {SWITCHES[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[3]}]
set_property PACKAGE_PIN W15 [get_ports {SWITCHES[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[4]}]
set_property PACKAGE_PIN V15 [get_ports {SWITCHES[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[5]}]
set_property PACKAGE_PIN W14 [get_ports {SWITCHES[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[6]}]
set_property PACKAGE_PIN W13 [get_ports {SWITCHES[7]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[7]}]

### LEDs
set_property PACKAGE_PIN U16 [get_ports {LEDS[0]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[0]}]
set_property PACKAGE_PIN E19 [get_ports {LEDS[1]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[1]}]
set_property PACKAGE_PIN U19 [get_ports {LEDS[2]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[2]}]
set_property PACKAGE_PIN V19 [get_ports {LEDS[3]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[3]}]
set_property PACKAGE_PIN W18 [get_ports {LEDS[4]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[4]}]
set_property PACKAGE_PIN U15 [get_ports {LEDS[5]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[5]}]
set_property PACKAGE_PIN U14 [get_ports {LEDS[6]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[6]}]
set_property PACKAGE_PIN V14 [get_ports {LEDS[7]}]
      set_property IOSTANDARD LVCMOS33 [get_ports {LEDS[7]}]

###Button
set_property PACKAGE_PIN U18 [get_ports RST]
      set_property IOSTANDARD LVCMOS33 [get_ports RST]
```