RAT Assignment #2
Counters as Program Counters

Objectives:

- ✓ To understand how to modify a generic counter module to serve more specific purposes
- ✓ To understand how to use an n-bit register (counter) and MUX to construct a counter, which can implement the features typically required of a "program counter".

The Big Picture:
While there are many different computer architectures that include many different modules, one common module in most all architectures is the Program Counter (PC). The basic definition of a computer is a circuit that executes instruction stored in memory to give a desired result. The PC is the component that provides the address of the instruction in program memory scheduled for execution. But, as important as it sounds, the PC is a relatively simple device.

General Notes:
A counter is a special form of register that performs operations typically associated with counting. Counters are generally operated synchronously (changes in output are synchronized to an active clock edge) but they also can have functions that are asynchronous. Modeling counters in VHDL is relatively simple as a VHDL counter models can be viewed as "n-bit flip-flops". In this manner the single-bit flip-flop is model is extended to a vector which allows for parallel operation on all the bits in counter.

This experiment includes extra external loading control for the PC with the addition of a MUX. When the PC is operating in a computer, it normally operates by incrementing the count in order to access the next instruction from memory. In addition, the PC must be able to parallel load values to support computer instructions involving executing instructions that are not the next instruction in the program sequence. This functionality is often associated with the PC is actually separate from the "counter" portion of the PC.

Vector Addition:
You should not need to port map in modules such as ripple-carry adders, rather your design should utilize VHDL's capability to add vectors as signed and unsigned numbers. You will need to include the IEEE.NUMERIC_STD.ALL library as shown:

```
use IEEE.NUMERIC_STD.ALL;        -- defines signed / unsigned vectors

signal inA, inB, outSum : std_logic_vector (3 downto 0);
signal A, B, Sum : unsigned (3 downto 0);

A <= unsigned(inA);   B <= unsigned(inB);
Sum <= A + B;
outSum <= std_logic_vector(Sum);
```

Notes on VHDL Structure and Signal Assignment in a Process Statement
Here are a couple of important rules regarding signals and their use within VHDL.

1.) Signals can be used globally throughout a body of VHDL code. This means that you can declare them at the top of an architecture, assigned within a process, then used anywhere in the entity, including outside of the process where they were assigned.

2.) When assigned within a process, signals are only assigned or updated after the process statement is over. This means that you can't assign a signal within a process statement and then use the "updated" value within the same process statement.

The implications of the above two rules for use within the program counter module being developed are that a signal that is your internal program counter value (PC) is required to contain the value of the PC that is loaded, incremented, reset, etc. This signal, which can be named PC_sig, will be updated within the process statement and then used outside the process statement to drive the PC_COUNT port.

Assignment:
Design a PC and associated input selection MUX. Figure 1 shows the PC while Figure 2 shows the PC and the associated input selection MUX. Figure 1 shows the PC while Figure 2 shows the PC and the associated input selection MUX. Use at least two modules in your design, one for the PC and one for the MUX. These modules can be either two separate entities (two-level design) or two processes in the same entity (one-level design).

Circuit Details:
Figure 1 shows the top-level black box model of the PC you'll implement. Table 1 provides an overview of the PC's operation in the context of the signals in Figure 1.

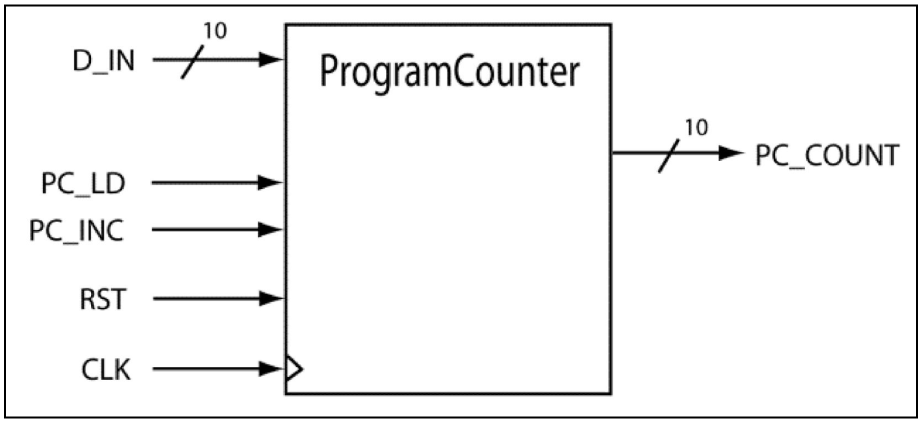

Figure 1: Program Counter Black Box Diagram

Signal	Comment
PC_COUNT	The current value in the PC; this value is used as an address to access values in the program memory.
RST	An active-high asynchronous reset; when RST = ‘1’, the output of the PC becomes all zeros.
PC_LD	An active high signal that synchronously loads D_IN into the PC. This signal has a higher precedence than the PC_INC input.
PC_INC	An active high signal that synchronously increments the value in the PC.
CLK	Synchronizes all “non-asynchronous” PC operations

Table 1: Tabular explanation of the program counter.

Figure 2 shows that the PC can be loaded from various sources. PC. The MUX allows the PC to change according to the current instruction needing execution. The MUX inputs include:

- FROM_IMMED: Branch and Call instructions obtain the new value of the PC from the immediate value included as part of the individual instruction formats.
- FROM_STACK: Return instructions (return from subroutine) obtain the new PC value from the stack.
- 0x3FF: Interrupts (when they are acted upon) set the PC to the interrupt vector: 0x3FF.

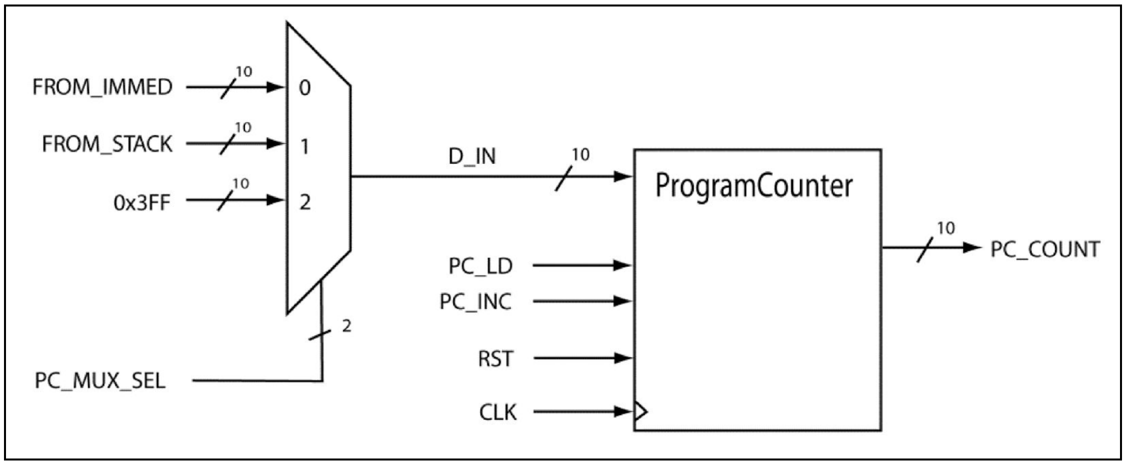

Figure 2: Program Counter with input selection MUX.