Concurrent Assignment Statements

1. Simple Assignment

signal_name <= expression

e.g.

```
SIGNAL x1, x2, x3, f : STD_LOGIC;
SIGNAL X, Y, S : STD_LOGIC_VECTOR(1 TO 3);

f <= (x1 OR x2) AND x3;

S <= X+Y;
```

Special Assignment Statement:

```
SIGNAL S : STD_LOGIC_VECTOR(1 TO 16);

S <= (OTHERS => '0');
```

2. Selected Signal Assignment

[label:] -- optional label
WITH expression SELECT
signal_name <= expression WHEN constant_value{,
expression WHEN constant_value};

e.g.

```
SIGNAL x1, x2, Sel, f : STD_LOGIC;

WITH Sel SELECT
    f <= x1 WHEN '0',
         x2 WHEN OTHERS;
```

All possible values of the select input Sel must be listed explicitly. In this case, the other possible values of Sel (1, Z, -, etc.) are taken care of by using OTHERS.

3. Conditional Signal Assignment

[label:]
signal_name <= expression WHEN logic_expression ELSE
{expression WHEN logic_expression ELSE}
expression;

e.g.

```
SIGNAL x1, x2, f ,in1, in2, in3 : STD_LOGIC;
SIGNAL g : STD_LOGIC_VECTOR(1 DOWNTO 0);

f <= '1' WHEN x1 = x2 ELSE '0';

g <= "01" WHEN in1 = '1' ELSE
     "10" WHEN in2 = '1' ELSE
     "11" WHEN in3 = '1' ELSE
     "00";
```

These assignments are ordered according to priorities of in1, in2 and in3. An assignment is executed only when the preceding one fails to execute.

4. GENERATE Statement

generate_label:
FOR index_variable IN range GENERATE
    statements;
    {statements;}
END GENERATE;

generate_label:
IF expression GENERATE
    statements;
    {statements;}
END GENERATE;

Note:
The *index_variable* does not have to be declared explicitly; it is a local variable whose scope is limited to the FOR-GENERATE loop.

e.g. An n-bit ripple carry adder

```
ENTITY addern IS
    GENERIC (n : INTEGER := 4);
    PORT (Cin   : IN STD_LOGIC;
          X,Y   : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
          S     : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0);
          Cout : OUT STD_LOGIC);
END addern;

ARCHITECTURE Structure OF addern IS
    SIGNAL C : STD_LOGIC_VECTOR(0 TO n);
BEGIN
    C(0) <= Cin;
    generate_n_fulladders:
    FOR i IN 0 to n-1 GENERATE
        stages: fulladd PORT MAP(C(i), X(i), Y(i), C(i+1));
    END GENERATE;
    Cout <= C(n);
END Structure;
```

Sequential Assignment Statements

In this type of assignments, the order in which the assignment statements appear may affect the meaning of the code.
Sequential assignment statements must be placed inside a PROCESS statement.
We'll consider two kinds of sequential assignment statements:
IF-THEN-ELSE statements and CASE statements.

[process_label:]
PROCESS [(input_signal_name {, input_signal_name})]
    [variable declarations]
BEGIN
        :
    [IF-THEN-ELSE Statements]
    [CASE Statements]
        :
END PROCESS [process_label]

A process statement has a parenthesized list of signals, called the *sensitivity lis*t which includes all input signals used inside the process.

When there is a change in the value of any signal in a process' sensitivity list, then the process becomes active.

Once a process is active, the statements inside the process are evaluated in sequential order. However, any assignments made to signals are hidden from outside the process until ALL the statements in the process have been evaluated.

If there are multiple assignments to the same signal in the process, then only the last assignment will be visible.

1. IF-THEN-ELSE Statement

IF expression THEN
    statement;
    {statement;}
ELSIF expression THEN
    statement;
    {statement;}
ELSE
    statement;
    {statement;}
END IF;

e.g.

```
PROCESS (Sel, x1, x2)
BEGIN
    IF Sel ='0' THEN
        f <= x1;
    ELSE
        f <= x2;
    END IF;
END PROCESS;
```

2. CASE Statement

```
CASE expression IS
    WHEN constant_value =>
        statement;
        {statement;}
    {WHEN constant_value =>
        statement;
        {statement;}}
    WHEN OTHERS =>
        statement;
        {statement;}
END CASE;
```

Note:
A CASE statement must include a WHEN clause for all possible valuation of the selection signal.

e.g. Assume x1, x2 and Sel are STD_LOGIC signals:

```
PROCESS (Sel, x1, x2)
BEGIN
    CASE Sel IS
        WHEN '0' =>
            f <= x1;
        WHEN '1' =>
            f <= x2;
        WHEN OTHERS =>
            f <= 'Z';
    END CASE;
END PROCESS;
```