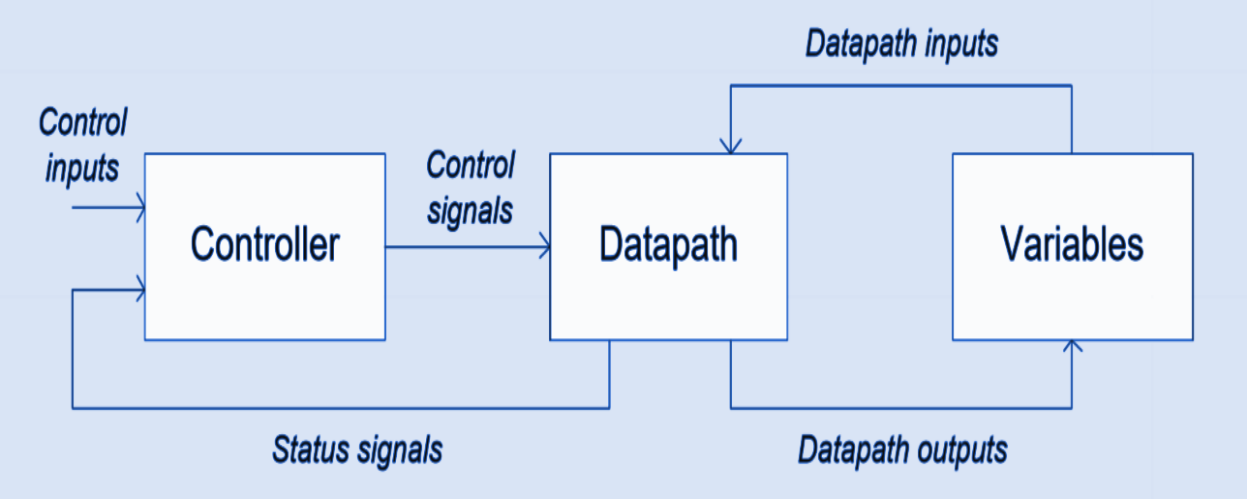
Cyber Systems

Assignment 1:

Implementation of an FSMD simulator in Python



Name	Student number
Irene Sotomayor	s205720
Pablo Escalona Trujillo	s212157
Brendan van de Venter	s204770

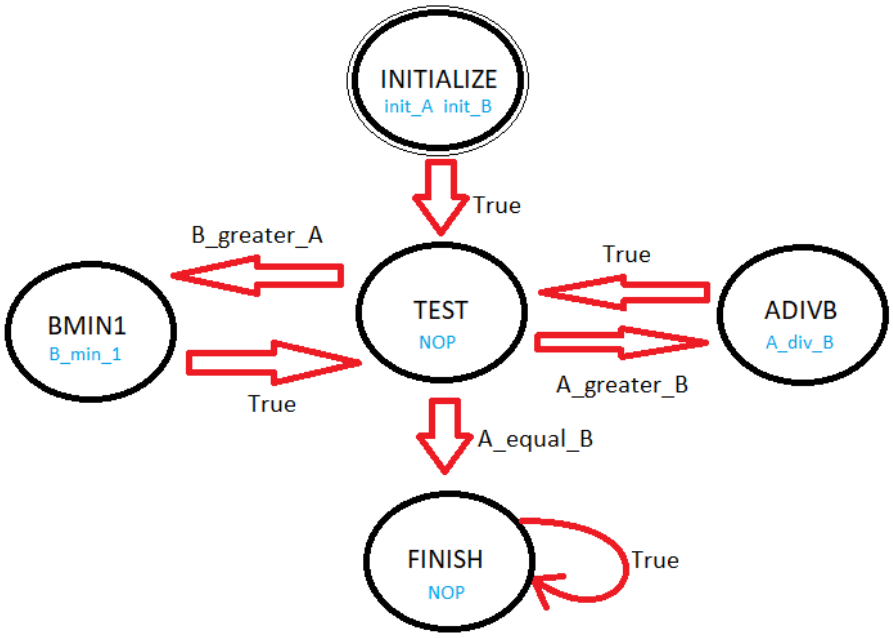
Aim

In this assignment we aim to understand FSMD simulators and to design one in order to get a clear overview on how State Machine models work. Finally, we will test the designed FSMD in the simulator and describe how this was done.

FSMD Description

Test 3

We designed a moore FSMD which has 5 states, as shown in the diagram below.
The variables are initialised in the "Initialize" state. In the "Test" state the two values are compared. If A is greater, it goes into the "ADIVB" state, where A is divided by B. If, on the other hand, B is greater, it goes to the "BMIN1" state, where B is subtracted by 1. After both operations, it always returns to the "Test" state.
When A and B are equal, then it goes to the "Finish" state, where the program ends.



```
<statelist>
   <state>INITIALIZE</state>
   <state>TEST</state>
   <state>ADIVB</state>
   <state>BMIN1</state>
   <state>FINISH</state>
  </statelist>
```

For our FSMD, we have 5 states. We have an *Initialize* state where we declare the variables, operations and conditions. We have the *Test* state that checks the conditions and if found true applies the correct next state. The two states, *ADIVB*, *BMIN1,* apply the instructions to the variables. The *Finish* state tells the FSMD to end.

```
<inputlist>
  </inputlist>

  <variablelist>
   <variable>var_A</variable>
   <variable>var_B</variable>
  </variablelist>
```

Since we did not have a stimulus file we do not have an input list. We then have the list of the variables. We chose to keep our FSMD file simple so we chose to have two variables, A and B.

The FSMD is able to do all tasks by using the conditions we have provided. By using conditions we can change states and apply instructions. If a certain condition is found to be true in the Test state, the corresponding next state will be applied. The FSMD will move to that state, and that is where the actual instruction will take place. Lets run through an example where we look at only condition/operation:

```
<operation>
    <name>A_div_B</name>
    <expression>var_A = var_A / var_B </expression>
</operation>
```

Operations are defined consisting of a name and an expression. This will be the actual function/calculation applied on our variables.

```
<condition>
    <name>A_greater_B</name>
    <expression>var_A &gt; var_B</expression>
</condition>
```

Next the conditions are defined so that the expression will identify and match the correct condition.

```
<TEST>
    <transition>
      <condition>A_equal_B</condition>
      <instruction>NOP</instruction>
      <nextstate>DONE</nextstate>
    </transition>
    <transition>
      <condition>A_greater_B</condition>
      <instruction>NOP</instruction>
      <nextstate>ADIVB</nextstate>
    </transition>
    <transition>
      <condition>B_greater_A</condition>
      <instruction>NOP</instruction>
      <nextstate>BMIN1</nextstate>
    </transition>
</TEST>
```

Inside the FSMD transition table we have the test state, which applies the nextstate according to the condition. In this case it would apply the “ADIVB” nextstate.

```
<ADIVB>
    <transition>
      <condition>True</condition>
      <instruction>A_div_B</instruction>
      <nextstate>TEST</nextstate>
    </transition>
</ADIVB>
```

We enter the ADIVB state, where the instruction is performed and once again the nextstate TEST is applied.
The conditions will always set the nextstate to be TEST, this creates a ‘loop’ that only breaks when the condition A equals B is True. The condition A equals B will apply the DONE nextstate, taking us to the DONE state where FSMD is complete.

Code snippets

```
Future_state = False

while cycle <= iterations:
```

Before entering the while loop, we have created the variable future state which will be the variable helping us transition between states. In this case we set it up to false because we do not want to switch state as soon as we enter the loop.

```
if Future_state:
    state = transition["nextstate"]
    Future_state = False
```

To switch states, we check whether the future state is true or false. If true then the state will be updated to the next state, and the future state is set up to false again to ensure the future state changes only when a condition is true.

```
for transition in fsmd[state]:
    if evaluate_condition(transition["condition"]) == True:
        execute_instruction(transition["instruction"])
```

In this part what we iterate over the list of states which contains dictionaries in which the conditions and the instructions for that particular state can be found. The functions used - evaluate_condition() and execute_condition() - are previously defined and what they do is to check whether a condition is true or false and if true it executes the instruction given by the state which could either be A_div_B or B_min_1.

```
if transition["nextstate"] != state:
    Future_state = True
    break
```

When the next state is not equal to the current state it will then set the future state to true so that it will meet the condition of being true and can move on to the next state. The break statement is just to break the for loop and go to the if statements.