STEP-1: Create python class for SimObject

In order to create a SimObject we first create a python class because every SimObject has a python file associated with it. We have to make this python file in the src folder which can be found in the gem5 folder.

For simplicity I have created the python file with name VectorOperations.py in gem5/src/Assignment2.

VectorOperations.py: It will contain the python class. We also have to mention the parameters in case we are taking input from the user. So, we are taking 2 vectors 3x1 and 3 times to wait for variables for each event.

- type: The type is the C++ class that you are wrapping with this Python SimObject.
- cxx_header: The cxx_header is the file that contains the declaration of the class used as the type parameter.
- cxx_class: The cxx_class is an attribute specifying the newly created SimObject is declared within the gem5 namespace.

```
type = 'VectorOperations'
cxx_header = "Assignment2/vector_operations.hh"
cxx_class = "gem5::VectorOperations"
wait1= Param.Latency("150")
wait2 = Param.Latency("1500")
wait3 = Param.Latency("15000")
Vector_1=VectorParam.Int([2.0,5.0,6.0])
Vector_2=VectorParam.Int([1.0,2.0,3.0])
```

STEP-2: Implement the SimObject in C++

vector_operations.hh: contains all the declarations.

- gem5 wraps all header files in #ifndef /#endif with the name of the file and the directory it's in so there are no circular includes.SimObjects should be declared within the gem5 namespace.

Have to include these in the header file to use the debug flags declared in SConscipt

- #include "base/trace.hh"
- #include "debug/VECTOR.hh"
- #include "debug/RESULTCROSS.hh"
- #include "debug/NORMALIZE.hh"
- #include "debug/RESULTSUB.hh"

- We have to declare all the private variables for our SimObject which is VectorOperations in this case.. The name for this parameter type is generated automatically from the name of your object. For our “VectorOpeartions” the parameter type’s name is “VectorOpeartionsParams”. So , all the event declarations with event wrappers along with cycles variables are declared in private of class.In public we will do the declaration of the startup() function which is used to schedule the events.

Vector_operations.cc : contains the body of the events and all computation along with DPRINTF statements which are used to output results with corresponding flags.

- We will initially schedule the event in the startup() function we added to the VectorOpeartoins class. The startup() function is where SimObjects are allowed to schedule internal events.These are the events which will start depending on the input given by the user.
  ```
  void VectorOperations::startup()
  {
     schedule(vector_cross_product, CrossCycles);
     schedule(normalize_vector, NormalizeCycles);
     schedule(vector_subtraction, SubtractionCycles);
  }
  ```

- It will be the body of the events i.e VectorCrossProduct, NormalizeVector and Vector Subtraction. Also the decode flags usage to print the desired result with use of the corresponding flag.
- A. DEBUG flag “VECTOR” will display the vectors.
- B. DEBUG flag “RESULTCROSS” will display the resultant value from VectorCrossProduct.
- C. DEBUG flag “NORMALIZE” will display the resultant vectors from NormalizeVector.
- D. DEBUG flag “RESULTSUB” will display the resultant vector from VectorSubtraction.

Step 3: Register the SimObject and C++ file

- In this step we have to make a Sconscript file which will register the SimObject and c++ file . In the SConscript file, there are a number of functions automatically defined after you import them. In this file, you have to declare the SimObject and the .cc file. Source('vector_operations.cc')
- We also have to mention the DECODE flags in this . In our assignment we have created 4 DECODE FLAGS.

- SimObject('VectorOperations.py', sim_objects=['VectorOperations'])
- DebugFlag('VECTOR')
- DebugFlag('RESULTCROSS')
- DebugFlag('NORMALIZE')
- DebugFlag('RESULTSUB')

Step 4: (Re)-build gem5

- To compile and link your new files you simply need to recompile gem5.
  Command: **scons build/X86/gem5.opt**

Step 5: Create the config scripts to use your new SimObject

- All gem5 instances require a Root object. We also have to take the user input to run all SimObject with user inputs.
- We need to call instantiate on the m5 module and actually run the simulation!
  m5.instantiate()

- print("Beginning simulation!")
- exit_event = m5.simulate()
- print('Exiting @ tick {} because {}'
        .format(m5.curTick(), exit_event.getCause()))

command**:build/X86/gem5.opt**
--debug-flags=NORMALIZE,RESULTCROSS,RESULTSUB,VECTOR
configs/assignment2/run.py