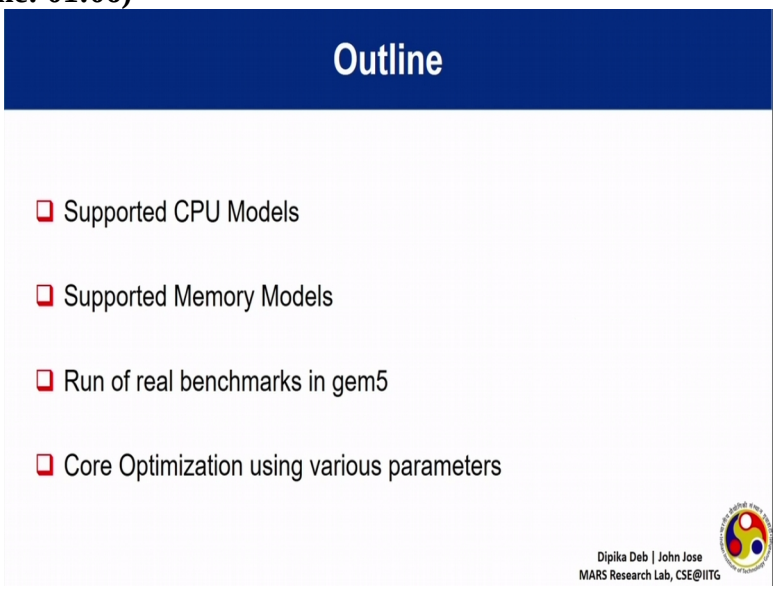
**Gem5-Core Optimization**
**Dipika Deb, Phg.D Scholar**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Guwahati, Assam**

**Tutorial 5**
**MARS Research Lab**

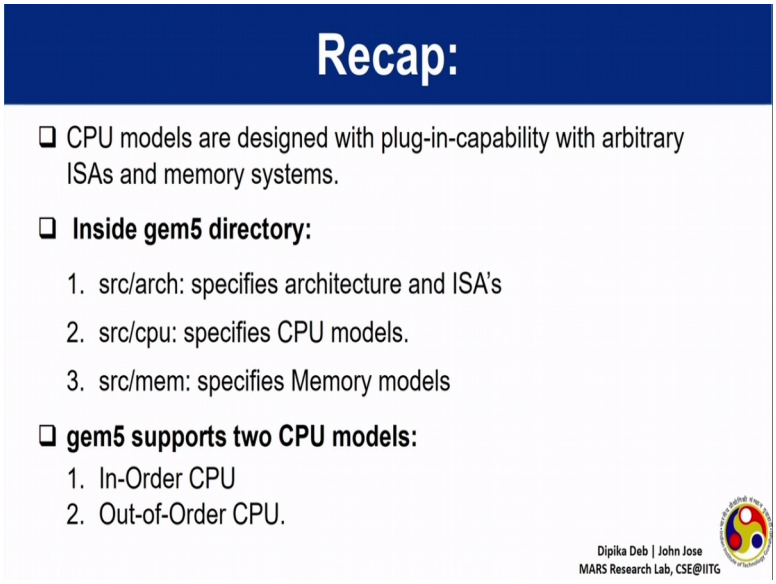
Welcome to the advanced computer architecture course. I am Dipika Deb. And in this video, I will explain how processors are implemented in gem 5. I will also give some insights on how to solve the first assignment of gem 5. Assignment 1 deals with processor or core optimizations. The optimizations are provided as parameters into the gem 5 simulator and we can evaluate the performance of the simulated system.

**(Refer Slide Time: 01:06)**




Moving further, this is the outline of today's video, I initially discuss about the available CPU models in gem 5 followed by the memory models. Next I will discuss how to run and configure real benchmarks in gem 5. And at last I will provide some insight on solving the assignment 1 in gem 5. As a quick recap, of our previous gem 5 videos, I have already discussed about the CPU models that are available.

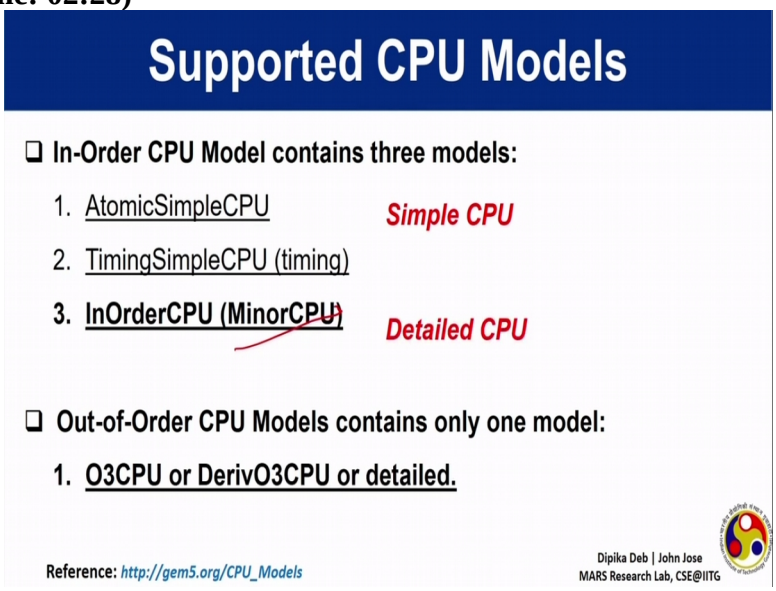
**(Refer Slide Time: 01:41)**




CPU models are designed with plugin capability with arbitrary ISAs that is instruction set architectures and memory systems. By this I mean that different CPU models can be integrated with the available chip memory models as in plugin capabilities. As a tool to the gem5 directory we have seen the first directory specifies the architecture and the instruction set architecture present in gem5.

The second directory specifies the CPU models and the third directory specifies the memory models as a border category of CPU models, gem5 supports 2 such models in-order CPU and out-of-order CPU.

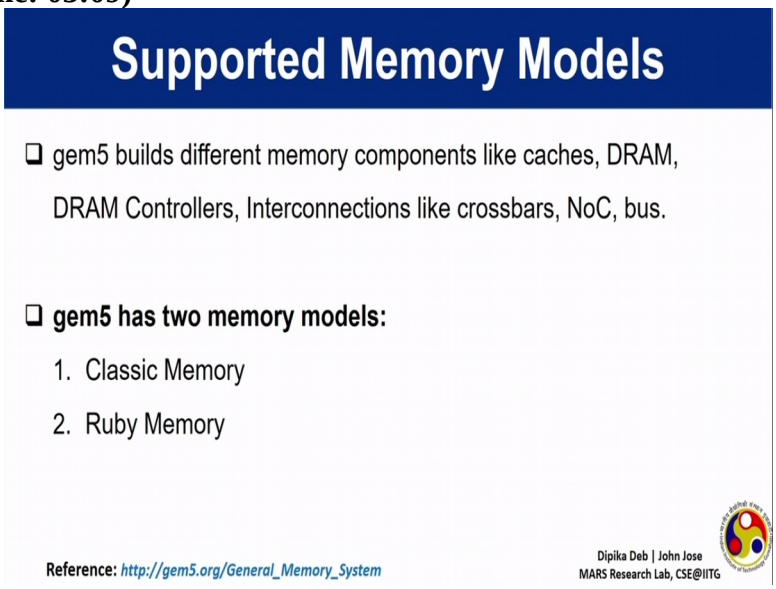
**(Refer Slide Time: 02:28)**




Now I will go in details of the 2 CPU models supported in gem 5 in-order CPU model, which contains 3 such sub models that is AtomicSimpleCPU, TimingSimpleCPU, or Timing. And the next one is in-order CPU. The name given to this model in gem 5 is MinorCPU. While

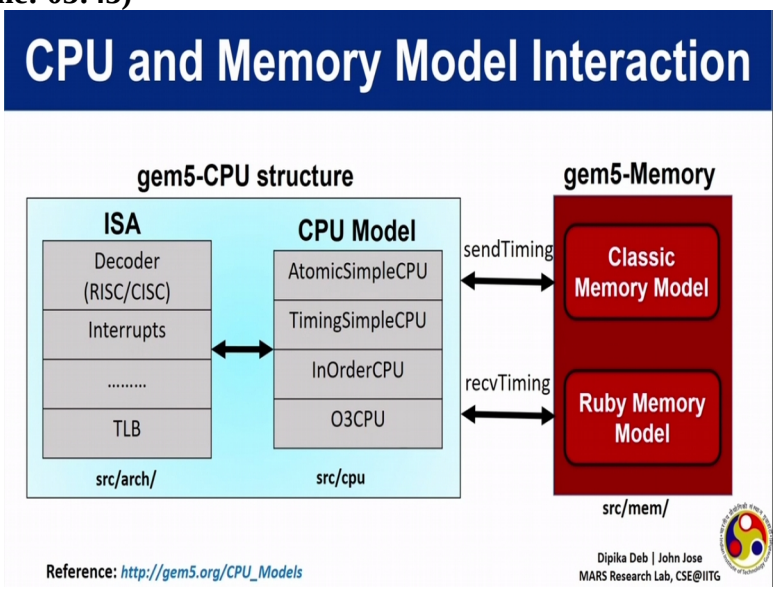
there is just one out of order CPU model, that is O3CPU or DerivO3CPU available in gem5. The first 2 CPU models are also known as simple CPU models. And the next 2 are known as detailed CPU models.

**(Refer Slide Time: 03:09)**



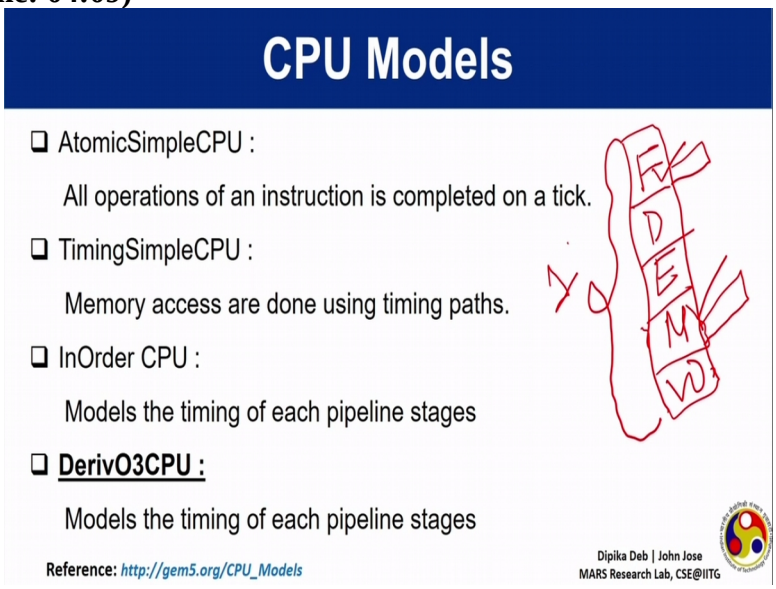

Moving further gem5 builds different memory components like the caches that is L1 cache L2 cache, main memory, DRAM controllers, and also the interconnection backbone. To implement the memory system gem5 uses 2 memory models that is classic memory and ruby memory. I will defer the discussion of ruby memory module in the next video throughout the assignment of gem5 we will use the ruby memory module.

**(Refer Slide Time: 03:43)**




The figure shows the interaction between gem 5 CPU structure and the memory modules. In this video, I will describe only on the available CPU models specially O3CPU or out of order CPU. The implementation of O3CPU can be found in this directory.

**(Refer Slide Time: 04:09)**




In CPU models, the AtomicSimpleCPU and TimingSimpleCPU are implemented as the conventional MIPS 5 stage pipeline that is the pipeline has a fetch stage, a decode stage, a execute stage, a memory stage and a writeback stage. The only difference between atomic SimpleCPU and TimingSimpleCPU is that for an AtomicSimpleCPU, all the operations related to an instruction is completed within a single cycle tape.

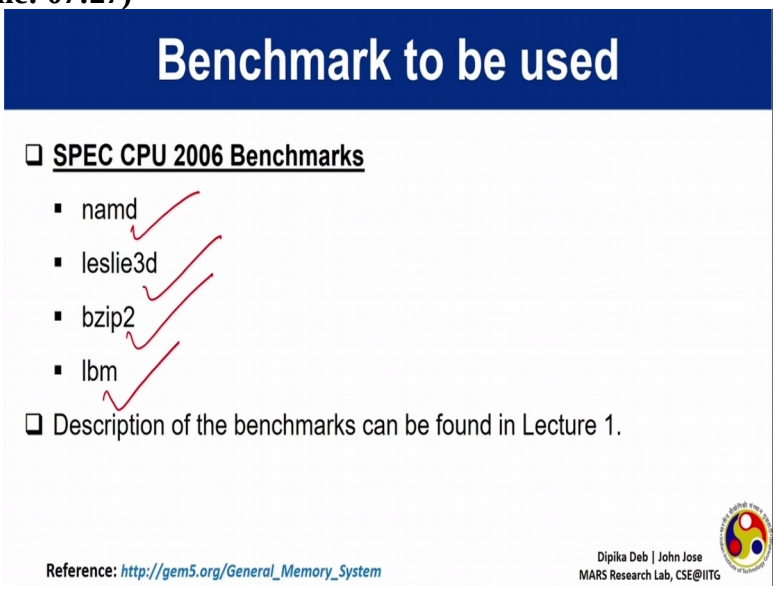
On the other hand, in TimingSimpleCPU, the timing behaviour of memory accesses are taken care of that is the memory are been accessed in fetch stage and in memory stage. In fetch stage the fetch delay, where the CPU brings instructions and data from the memory is taken care of, while the memory delay during the MEM stage is also accurately being processed in TimingSimpleCPU. Hence, it is also known as timing module.

In Minor CPU, and out of order CPU, each pipeline stages are modelled with accurate timings as in real system. The difference between these 2 models are the InOrder CPU is modelled as a 4 stage pipeline, that is the first stage is fetch 1, the second stage is fetch 2 the third stage is decode stage and the fourth stage is execute stage. In InOrder CPU or Minor CPU, the program execute and fetches in InOrder and it also executes in InOrder.

On the other hand, the out of order CPU is modelled as a 7 stage pipeline. That is, it has a fetch stage or decode stage, a rename stage, an issue stage, execute, right back and commit. Here the program fetch occurs in in-order. The execution occurs in out of order and the commit occurs in InOrder. In all the 7 stages in gem5 branch predictors are implemented in the decode stage.

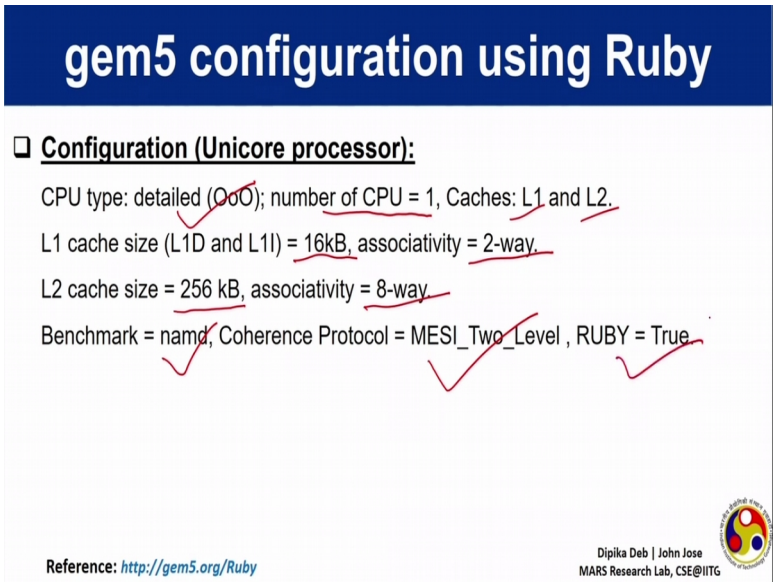
We have reorder buffers implemented in rename stage and the out of order execution is taken care by score boarding that is implemented in execution stage. For further details of the O3CPU and the other CPU models, you can go through this reference. We will use the O3CPU for assignment one.

**(Refer Slide Time: 07:27)**




For all the assignments on gem5, we will use 4 Benchmarks for the multi program SPEC CPU 2006 Benchmarks suit that is namd, leslie3d, bzip2 and lbm. The description of spec benchmarks can be found in lecture 1 of this course.

**(Refer Slide Time: 07:52)**

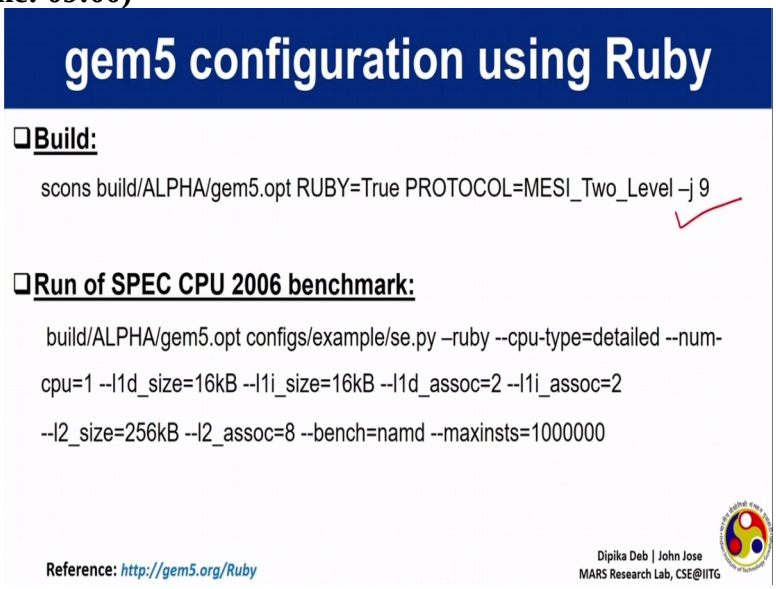



Let us now deal with configuring and running real benchmarks on gem5, the configuration of the simulated machine that we will use for this video is mentioned in the slide. we will model a detailed out of order CPU where the number of core is 1 also known as a unicode processor,

we will have 2 levels of caches L1 and L2, where the L1 cache size including L1 data size and L1 instruction is of 16 kB and 2-way set associative, the L2 cache size is of 256 kB and 8-way set associative.

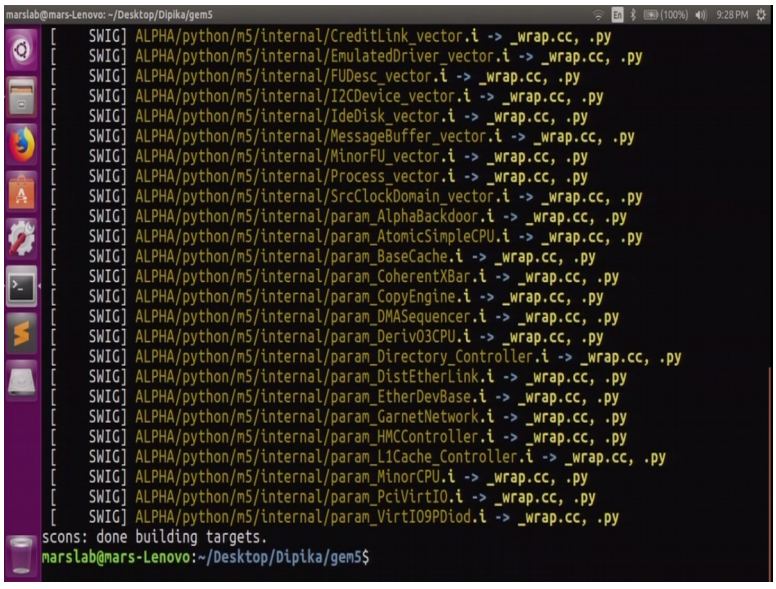
In the example that I will show we will use the benchmark namd, interested readers can also implement the same configuration with different benchmarks that is lbm, bzip2 and the others available. The Coherence protocol that we will use is MESI_Two_Level and since we are using RUBY memory model we will make RUBY is equal to true.

**(Refer Slide Time: 09:00)**




To build a system we will use this command.

**(Refer Slide Time: 09:04)**

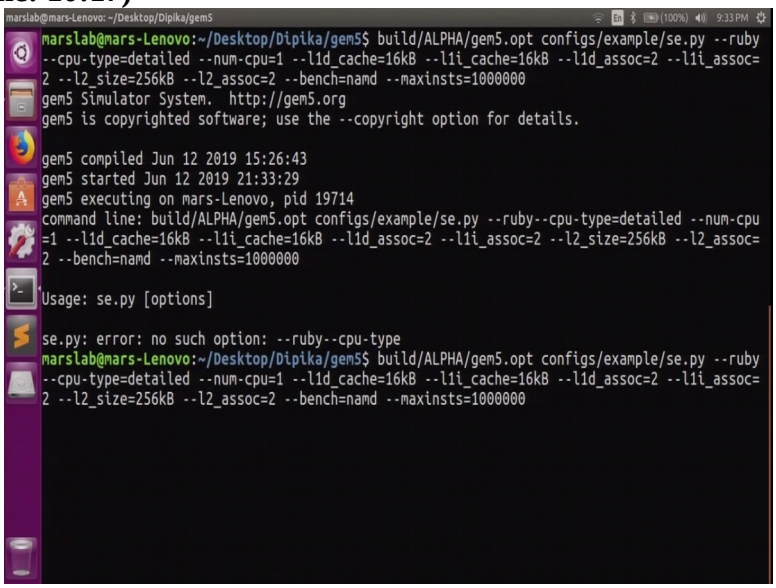



Go in to the gem5 directory, in the gem5 directory type the command: scons build, we will use ALPHA architecture and the binary that we will use is gem5.opt. We will make RUBY as

true and PROTOCOL as MESI_TWO_Level. And then we will specify the number of CPUs that we have. In the first video I have already shown that the command LSCPU in ubuntu terminal will show you the number of CPUs that are available in my system I have 8 CPU.

So I will add number of CPUs + 1, that is 9 after that, this command will build the Ruby memory system in the gem5.

**(Refer Slide Time: 10:17)**

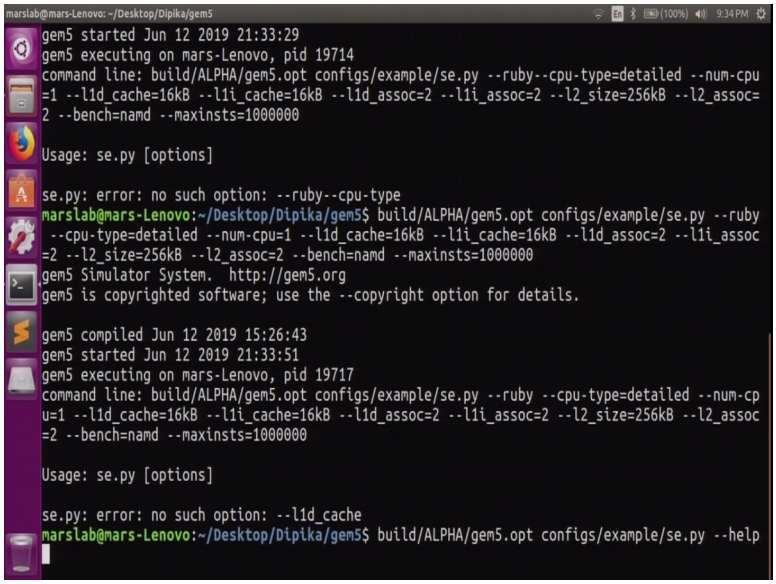



To run the benchmark with the configuration mentioned in the slide, we will type the following command in the terminal. Build will go to the ALPHA architecture that we have built gem5.opt binary configs example se.py we will leave se.py throughout our assignment. I will set --Ruby flag which will set Ruby as true I have already mentioned in the configuration file that the CPU type is detailed are O3.

We have only one CPU with the L1 cache size that is L1 data cache a 16 kB, L1 instruction cache a 16 kB. The associativity for both the cache that is L1 instruction and data is 2 which can be set using this parameters. The L2 cache size is 256 kB and the associativity is 2 we use namd benchmark using this flag and the number of instructions that we are going to simulate is 1 million.

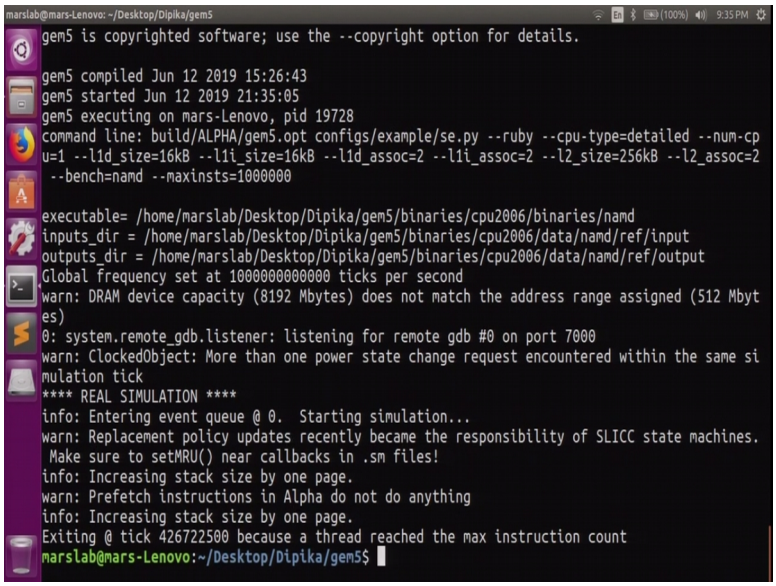
So you can see that this have shown an error. So on getting such an error clearly look what the error is all about. Here the --Ruby and --CPU type should have been separated by a space

**(Refer Slide Time: 12:26)**



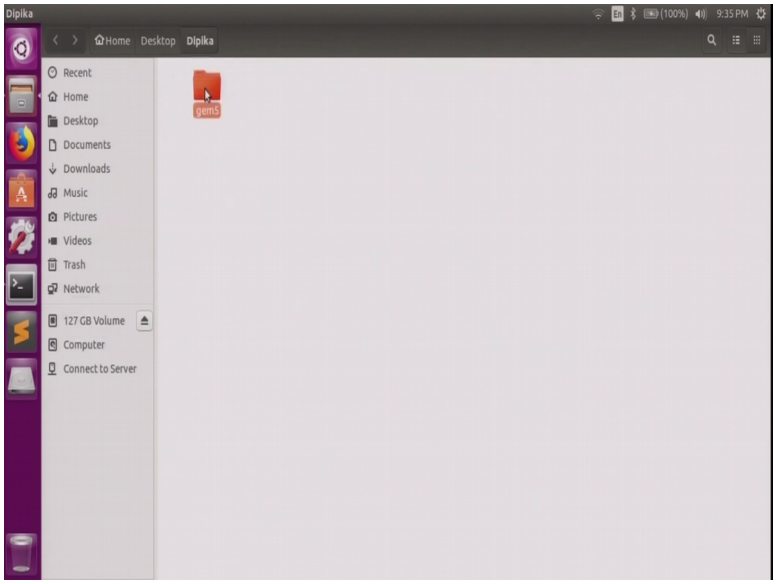

We again have another error as L1-D cache. So, in case if you get such multiple errors and you don it know where the error is all about what you can type is a help command. This is available in gem5.opt. You can go to the simulation script that is se.py and type - - help.

**(Refer Slide Time: 12:54)**

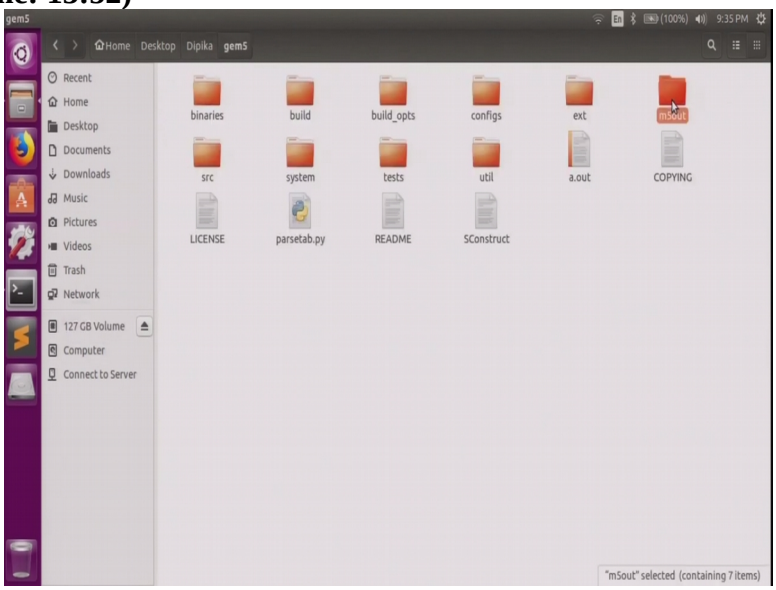



This will show you the available parameters that you can support such as you can see l1d_size, l1i_size, l2_size, l3_size the associativity that we have typed in the terminal, the cache block size, --Ruby and there are various such parameters. So, now we got to know that our error is it is L1-D size and not L1 cache. I have shown this so that you get to know what the error is all about and try to resolve it. This process is simulating our machine and the simulation is completed.

**(Refer Slide Time: 13:49)**

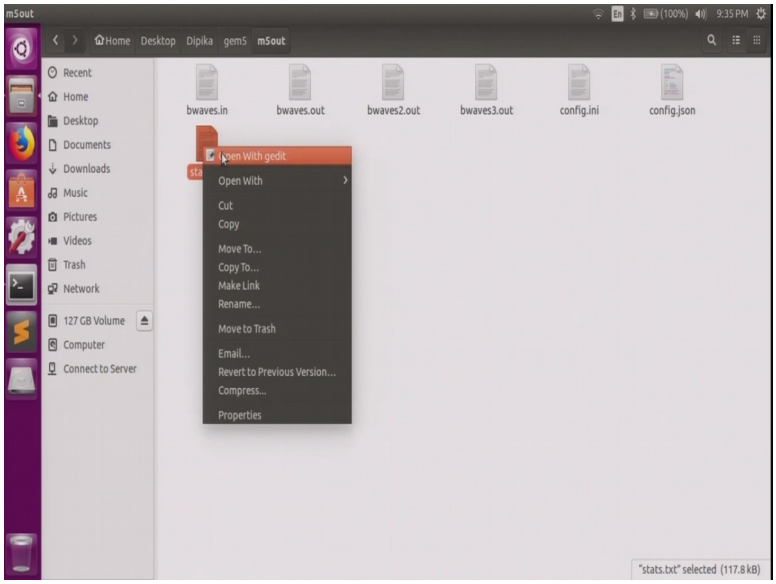



**(Refer Slide Time: 13:52)**

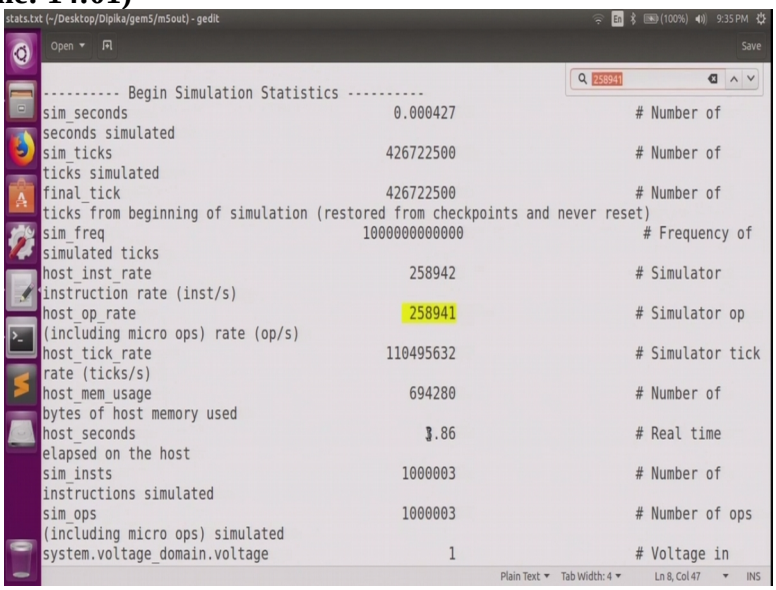



Now you can find the statistics in go inside the gem 5 directory inside the m5 out directory. You will find the stats.txt file. On opening the stats.txt file.

**(Refer Slide Time: 13:55)**

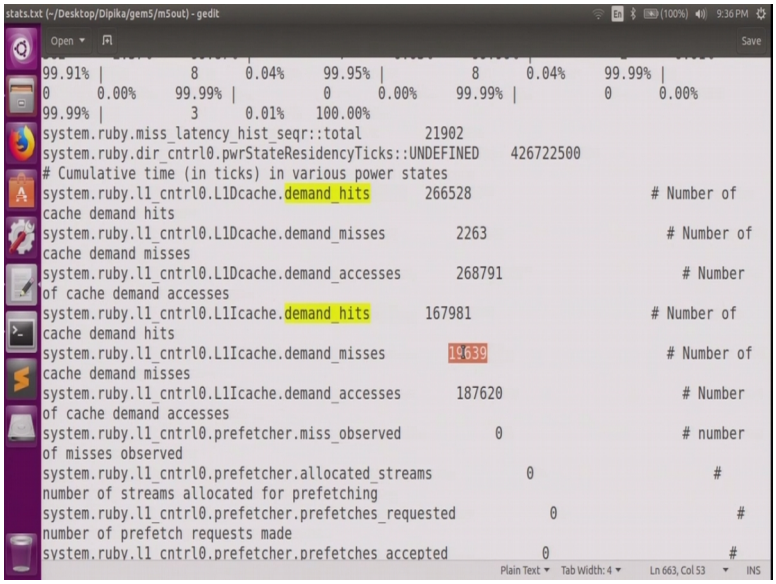



**(Refer Slide Time: 14:01)**



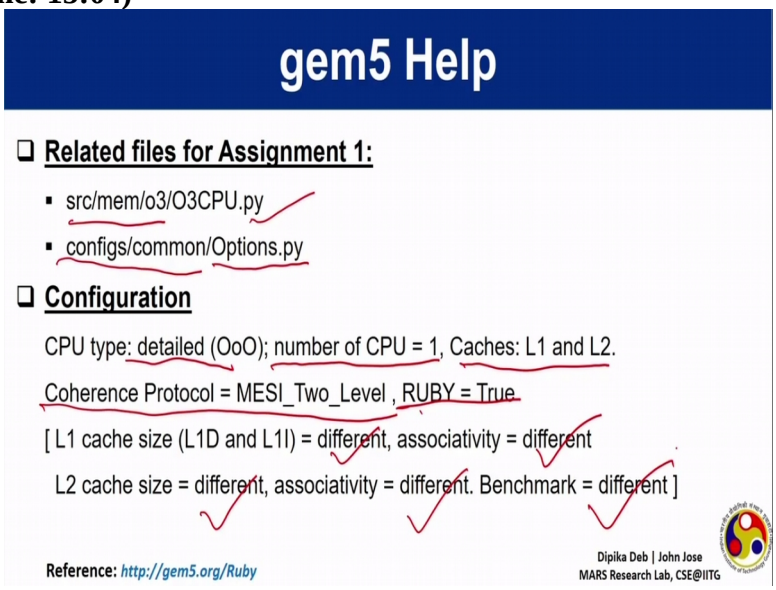

You will find that this is the number of seconds that our script have run on the simulated machine. This was the frequency that we have used that is one giga hertz the instruction rate per second was around this much. We can also see the demand misses that the L1 caches have experienced.

**(Refer Slide Time: 14:32)**




Just type demand hits or misses, you will find that the L1 data cache have experienced around this much number of hits. And it has experienced 2263 number of misses. You can also find that the L1 cache the demand hit was around this much. And for the L1 cache the demand misses. The cache misses were around this much. So this statistics will be useful in calculating various parameters during your assignments.

**(Refer Slide Time: 15:04)**




So the related files for assignment one can be found in this folder. This is this script where you need to make some parameter changes and Options.py contains the default values of L1 cache and L1D cache I cache L2 cache and many more. You can find this in this path. And for assignment one we will use O3CPU, Unicore processor, 2 level of caches L1 and L2. The Coherence Protocol will be MESI_Two_Level and we will set Ruby is true.

The L1 cache size will differ, the associativity will be different and we will also specify what will be the L2 cache size. Its associativity and the benchmarks. By this we come to the end of today is video for any further queries, you can write us in the discussion forum of the course web page. Thank you