Debugging and debug flags

We will cover

- How to enable debug flags (examples of DRAM and Exec) [Not doing this]
- `--debug-help`
- Adding a new debug flag
- Functions other than DPRINTF
- Panic/fatal/assert
- gdb? Mahyar's opinion: does not sound interesting or exclusive to gem5 and hard to teach.

DebugFlags: Debugging and Logging in gem5

IMPORTANT: This slide deck builds on top of what has already been developed in Introduction to SimObjects.

DebugFlags

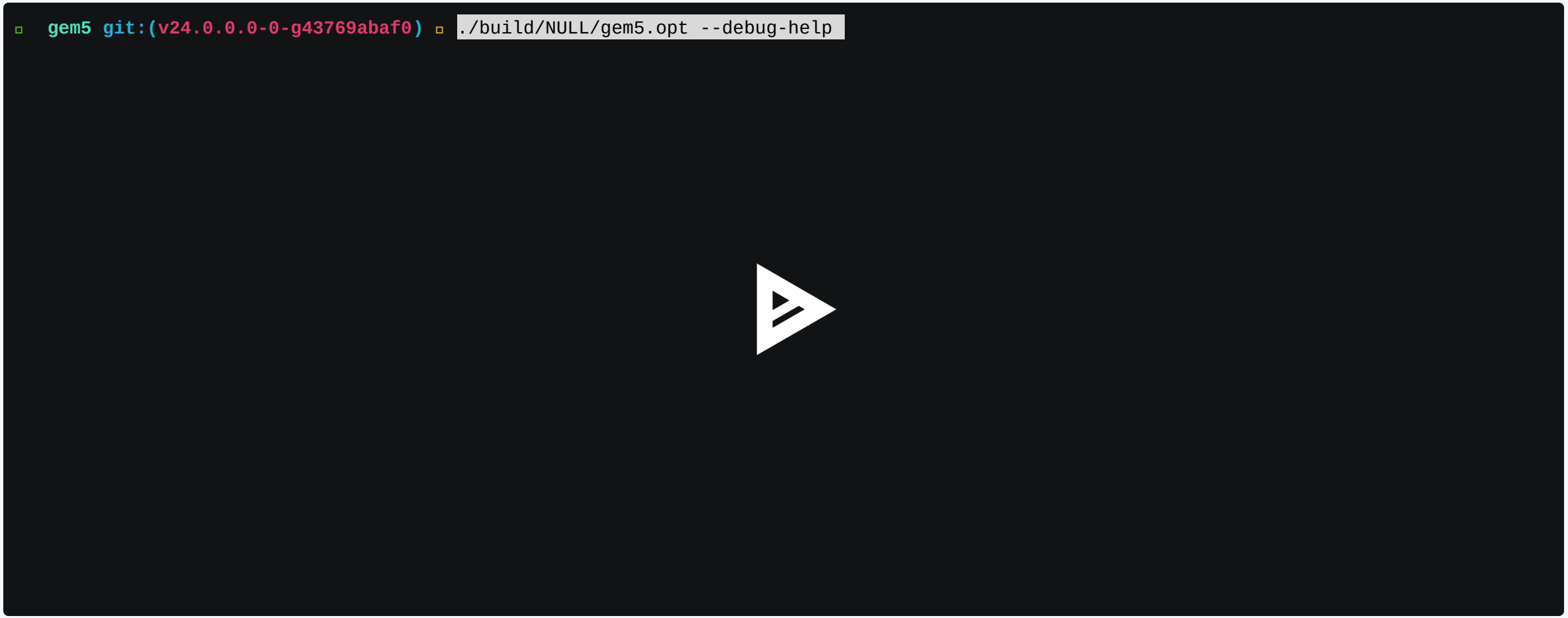
`DebugFlags` help with debug printing. Debug printing is useful for debugging models in gem5 and logging.

Each `DebugFlag` enables printing certain statements within the gem5 code base. Run the following commands to see all the available `DebugFlags` in gem5.

```
cd gem5
./build/NULL/gem5.opt --debug-help
```

This command will show you a list of all the `DebugFlags`. You can choose to use a specific `DebugFlag`, like `Activity`, or you can choose a class of `DebugFlags`, like `Registers`, which will enable the following `DebugFlags`: `IntRegs`, `FloatRegs`, `VecRegs`, `VecPredRegs`, `MatRegs`, `CCRegs`, `MiscRegs`.

In the following slide, you will see the expected output.


gem5 git:(v24.0.0.0-0-g43769abaf0)
./build/NULL/gem5.opt --debug-help

DebugFlags: HelloExampleFlag

To define a new `DebugFlag` in gem5, you just have to define it in **any** `SConscript` in the gem5 directory. However, it is convention that `DebugFlags` are defined in the same `SConscript` that registers `SimObjects` that are relevant to the `DebugFlag`.

To define a new `DebugFlag` that we will use to print debug/log statement in `HelloSimObject`, open `src/bootcamp/hello-sim-object/SConscript` in your editor of choice and add the following line.

```
DebugFlag("HelloExampleFlag")
```

Adding this line will create a new **auto-generated** header file (with the same name as the `DebugFlag`) that defines the `DebugFlag` in C++.

DebugFlags: Using HelloExampleFlag in Code

One of the functions in gem5 that allows for debug printing is `DPRINTF`, which will let you print a formatted string if a certain `DebugFlag` is enabled (more on how to enable `DebugFlags` later). `DPRINTF` is defined in `src/base/trace.hh`. Make sure to include it every time you want to use `DPRINTF`.

Now let's get to actually adding `HelloExampleFlag` in C++. As I mentioned, the header files for `DebugFlags` are auto-generated. For now, trust that the header file for `HelloExampleFlag` will be in `build/NULL/debug/HelloExampleFlag.hh` when we recompile gem5.

Let's include the header files in `hello_sim_object.cc` by adding the following lines. Remember to follow the conventional order of includes!

```
#include "base/trace.hh"
#include "debug/HelloExampleFlag.hh
```

Now let's add a simple `DPRINTF` statement inside the constructor of `HelloSimObject` to print `Hello from ...`. Do it by adding the following line after the `for-loop`. **NOTE**: `__func__` will return the name of the function we're in as a string.

DebugFlags: How Files Look Like

Below is how `src/bootcamp/hello-sim-object/SConscript` should look like after the changes.

```
Import("*")

SimObject("HelloSimObject.py", sim_objects=["HelloSimObject"])

Source("hello_sim_object.cc")

DebugFlag("HelloExampleFlag")
```

Below is how `src/bootcamp/hello-sim-object/hello_sim_object.cc` looks like with changes

```
#include "bootcamp/hello-sim-object/hello_sim_object.hh"

#include <iostream>

#include "base/trace.hh"
#include "debug/HelloExampleFlag.hh"
```

Let's Recompile

Now, let's recompile gem5 with the command below. After compilation is done, you should be able to find the header file in `build/NULL/debug/HelloExampleFlag.hh`.

```
scons build/NULL/gem5.opt -j$(nproc)
```

And here is a snippet of the contents of `build/NULL/debug/HelloExampleFlag.hh`.

```
/**
 * DO NOT EDIT THIS FILE!
 * File automatically generated by
 *   build_tools/debugflaghh.py:139
 */

#ifndef __DEBUG_HelloExampleFlag_HH__
#define __DEBUG_HelloExampleFlag_HH__

#include "base/compiler.hh" // For namespace deprecation
```

DebugFlags: After Adding HelloExampleFlag

Now, our `HelloExampleFlag` should be listed whenever we print debug help from gem5. Let's run the following command in the base gem5 directory to verify that our `DebugFlag` is added.

```
./build/NULL/gem5.opt --debug-help
```

Below shows the expected output.

```
// asciinema
```

Enabling DebugFlags: Using Configuration Script

To enable a `DebugFlag` you can import `flags` from `m5.debug` and access the flag by indexing `flags`. You can enable and disable flags by calling `enable` and `disable` methods. Below is an example of what your `second-hello-example.py` would look like if you wanted to enable `HelloExampleFlag`.
CAUTION: Do **not** make this change in your configuration script for now.

```
import m5
from m5.debug import flags
from m5.objects.Root import Root
from m5.objects.HelloSimObject import HelloSimObject



root = Root(full_system=False)
root.hello = HelloSimObject(num_hellos=5)

m5.instantiate()

flags["HelloExampleFlag"].enable()
```

Enabling DebugFlags: Using Command Line

Alternatively you can pass `--debug-flags=[comma-separated list of DebugFlags]` to your gem5 binary when running your configuration script. As an example, below is a shell command that you can use to enable `HelloExampleFlag` (like always, run it in the base gem5 directory).

```
./build/NULL/gem5.opt --debug-flags=HelloExampleFlag configs/bootcamp/hello-sim-object/second-hello-example.py
```

Simulate: Without HelloExampleFlag

Now let's simulate `second-hello-example.py` with and without `DebugFlags` and compare the output.

Run the following command to simulate `second-hello-example.py` without `DebugFlags`.

```
./build/NULL/gem5.opt configs/bootcamp/hello-sim-object/second-hello-example.py
```

Below is a recording of my terminal when doing this.

```
# Mysore/Saili put asciinema here.
```

Simulate: With HelloExampleFlag

Run the following command to simulate `second-hello-example.py` with `HelloExampleFlag`.

```
./build/NULL/gem5.opt --debug-flags=HelloExampleFlag configs/bootcamp/hello-sim-object/second-hello-example.py
```

Below is a recording of my terminal when doing this.

```
# Mysore/Saili put asciinema here.
```

Assertions in gem5

I strongly recommend using `assert` and `static_assert` when developing for gem5. They will help you find untrue assumptions you've made, and they will help you find any development mistakes early. `assert` and `static_assert` are standard C++ functions that you can (and are strongly encouraged to) use while developing in gem5.

`fatal`, `fatal_if`, `panic`, and `panic_if` are gem5's specific assert-like functions that allow you to print error messages. gem5 convention is to use `fatal` and `fatal_if` to assert assumptions on user inputs (similar to `ValueError`). As an example, if a user tries to configure your SimObject with negative capacity you can use `fatal` or `fatal_if` in your `SimObject` to let the user (most probably yourself) know their mistake. Below shows an example of doing this with `fatal` and `fatal_if`.

```
if (capacity < 0) { fatal("capacity can not be negative.\n"); }
\\ OR
fatal_if(capacity < 0, "capacity can not be negative.\n");
```

You should use `panic`, and `panic_if` to catch developer mistakes. We will see some examples in Ports.

Other Debugging Facilities in gem5

```
// Saili write up here.
// Look at other DPRINT/DDUMP functions.
// http://learning.gem5.org/book/part2/debugging.html
// src/base/trace.hh
// https://www.gem5.org/documentation/learning_gem5/part2/debugging/
```