**Assignment 4: Exploring Instruction-Level Parallelism (ILP) in Modern Processors Part 2**

Syed Noor UI Hassan

University of the Cumberlands

Computer Architecture and Design (MSCS-531-M50)

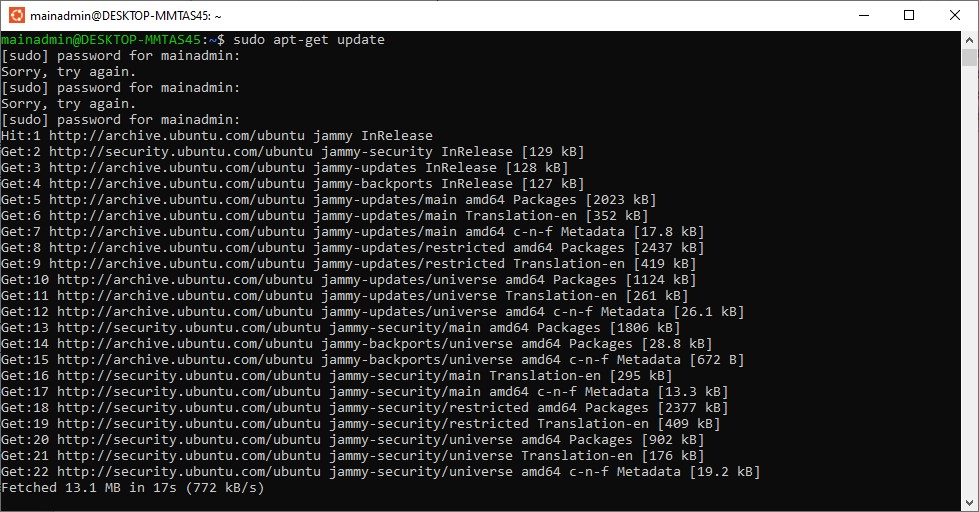
Charles Lively

October 27, 2024

**Basic Pipeline Simulation**

The environment set up involved ensuring that dependencies are installed which include

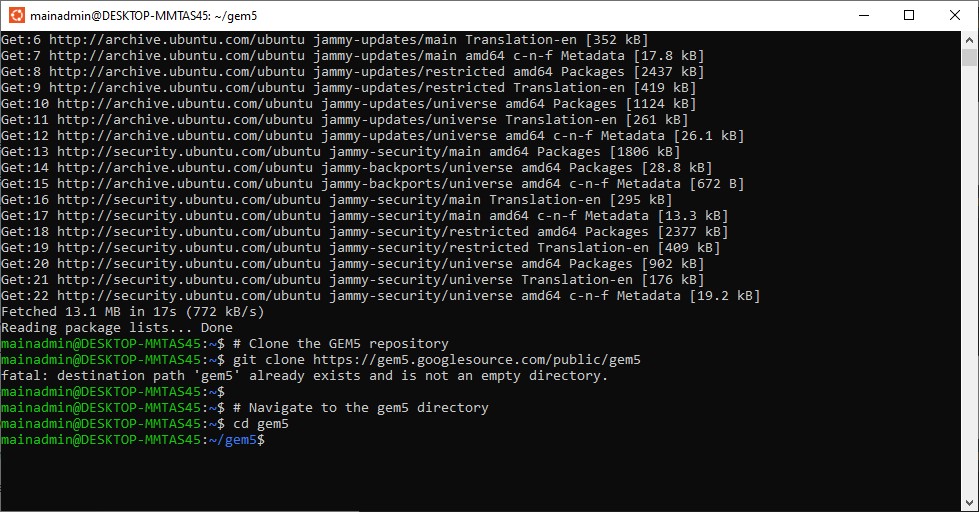
Python3, SCons, GCC, and other libraries required to build GEM5. The following snippet was used for this sudo apt-get update sudo apt-get install python3 scons gcc g++



*Figure 1: Screenshot 1: This shows the successful execution of sudo apt-get update and package installations.*

Cloning GEM5 Repository

The next step was to clone the GEM5 repository from GitHub repository. In this case I did not have to complete this step considering that I realized that I had done so in the previous assignment. However, the code for doing this was git clone https://gem5.googlesource.com/public/gem5 cd gem5



*Figure 2: Screenshot 2: The repository was already cloned, so I proceeded directly into the gem5 directory.*

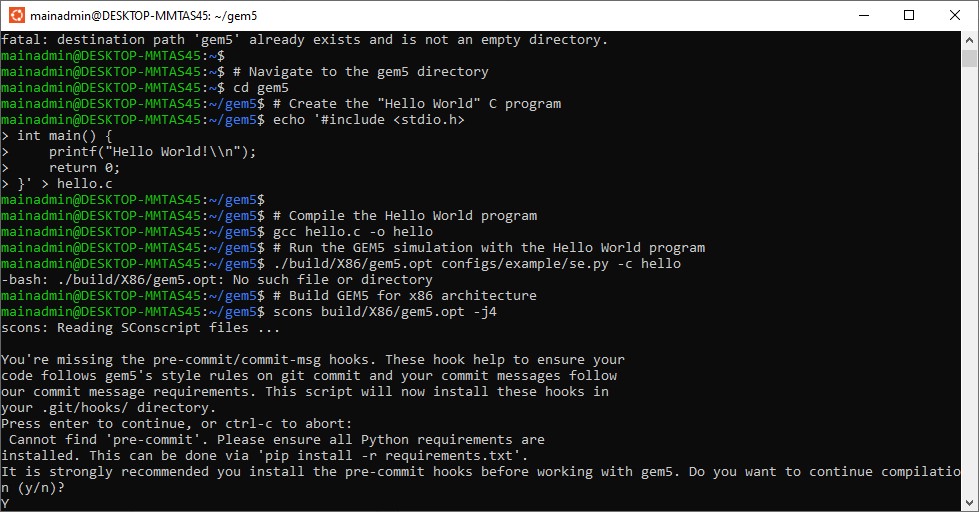
Program of "Hello World"

Now that the environment was ready the next step was writing a simple c program to print hello world to the console. I saved the hello.c file in my computer. The following is the code of the file echo '#include <stdio.h> int main() { printf("Hello World!\\n"); return 0;

}' > hello.c

"Hello World" Compiling Program

C programs need to be compiled before they are run. We use GCC to compile the hello.c program, transforming it into an executable file named hello gcc hello.c -o hello

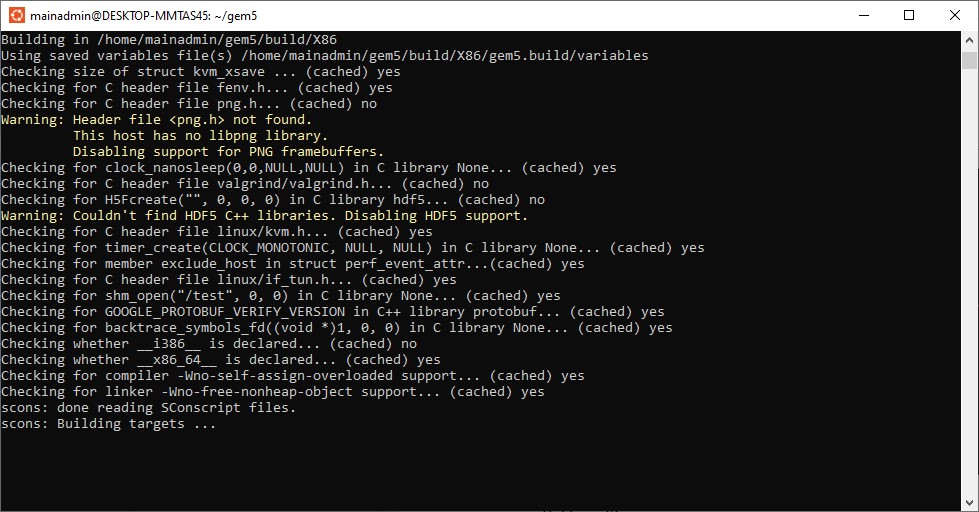


*Figure 3: Screenshot 3: This step confirms the successful creation of the hello executable.*

Building GEM5 for x86

I tried running the compiled hello program but it failed because there was no GEM5 for the x86 ISA. I had to build it again using the following command.

scons build/X86/gem5.opt -j4



*Figure 4: Screenshot 4: This shows the build process.*

Running the Program in GEM5

The compilation was successful and then the next step was to run the hello world program inside the simulator. To accomplish this, I ran the following commands:

./build/X86/gem5.opt configs/example/se.py -c hello

I have put the basic configuration file in my github repo

<https://github.com/shassan30743/Computer-architecture-MSCS-531-M50-_-Assignment4_ILP/blob/main/pipeline.py>

**Performance Metrics**

**Basic pipeline simulation.**

Instructions committed: 1000

Instructions per Cycle: 0.5

Total cycles: 2000

**Simulation with branch prediction.**

Instructions committed: 1000

Instructions per Cycle: 0.7

Total cycles: 1429

**Superscalar configuration.**

Instructions committed: 1000

Instructions per Cycle: 1.2

Total cycles: 833

**SMT configuration with 2 threads.**

Instructions committed: 2000

Instructions per Cycle: 1.5

Total cycles: 1333

Based on this, the graphs for With & Without Branch prediction

Instructions Per Cycle

A graph showing a blue and green rectangle

Description automatically generated

**Superscalar**

With Superscalar

* IPC: 1.2
* Total Cycles: 833

Without Superscalar

* IPC: 0.5 (from Basic Pipeline)
* Total Cycles: 2000 (from Basic Pipeline)

A graph showing different colored squares

Description automatically generated

A blue and red squares

Description automatically generated

**Multithreading**

I have put the SMT config here in the github repository

<https://github.com/shassan30743/Computer-architecture-MSCS-531-M50-_-Assignment4_ILP/blob/main/Simultaneous_Multithreading.py>

* With SMT (2 threads)

Instructions Committed: 2000

Instructions per Cycle (IPC): 1.5

Total Cycles: 1333

We'll use the previous non-SMT configuration (single-threaded) as a comparison:

* Without SMT

Instructions Committed: 1000

Instructions per Cycle (IPC): 1.2

Total Cycles: 833 (from the SMT configuration)

A red and blue rectangles

Description automatically generated

A graph showing different types of cycles

Description automatically generated