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

**Part 2**

2024

* **name: Abdul raheman Gotori**
* **Student id: 005029919**
* **COURSE & TITLE: Computer Architecture and Design (MSCS-531-M51)**
* **DATE:** **27th october 2024**

Table of Contents

[**Part 2** 2](#_Toc180893458)

[**Basic Pipeline Simulation** 2](#_Toc180893459)

[**Performance Metrics** 7](#_Toc180893460)

[**Multithreading** 8](#_Toc180893461)

[**Troubleshooting** 13](#_Toc180893462)

### **Part 2**

### **Basic Pipeline Simulation**

**Writing the “Hello World**

*Created hello.c file*

**A screenshot of a computer

Description automatically generated** **Compiling the Program**

Was able to compile and successfully get the output by using the commands below.*gcc hello.c -o hello*

*./hello*

**A computer screen with white text

Description automatically generated**

The script I used is below for baseline configuration is :- *import m5*

*from m5.objects import \**

*# Create the system*

*system = System()*

*system.clk\_domain = SrcClockDomain(clock="1GHz", voltage\_domain=VoltageDomain())*

*system.mem\_mode = "timing"*

*system.mem\_ranges = [AddrRange("512MB")]*

*# Create CPU and Memory Bus*

*system.cpu = O3CPU()*

*# Branch predictor*

*system.cpu.branchPred = LocalBP()*

*# Super scalar*

*system.cpu.fetchWidth = 12*

*system.cpu.issueWidth = 12*

*system.cpu.dispatchWidth = 12*

*system.cpu.commitWidth = 12*

*system.membus = SystemXBar()*

*# Create a simple cache (4-way set associative)*

*system.cpu.icache = Cache(*

*size='32kB',*

*assoc=4, # 4-way set associative*

*tag\_latency=2,*

*data\_latency=2,*

*response\_latency=1,*

*mshrs=16,*

*tgts\_per\_mshr=4*

*)*

*system.cpu.dcache = Cache(*

*size='32kB',*

*assoc=4, # 4-way set associative*

*tag\_latency=2,*

*data\_latency=2,*

*response\_latency=1,*

*mshrs=16,*

*tgts\_per\_mshr=4*

*)*

*# Connect caches to CPU and memory bus*

*system.cpu.icache.cpu\_side = system.cpu.icache\_port # Ensure this is correctly defined*

*system.cpu.icache.mem\_side = system.membus.cpu\_side\_ports*

*system.cpu.dcache.cpu\_side = system.cpu.dcache\_port # Ensure this is correctly defined*

*system.cpu.dcache.mem\_side = system.membus.cpu\_side\_ports*

*# Set up interrupt controller*

*system.cpu.createInterruptController()*

*system.cpu.interrupts[0].pio = system.membus.mem\_side\_ports*

*system.cpu.interrupts[0].int\_requestor = system.membus.cpu\_side\_ports*

*system.cpu.interrupts[0].int\_responder = system.membus.mem\_side\_ports*

*# Create a memory controller*

*system.mem\_ctrl = MemCtrl()*

*system.mem\_ctrl.dram = DDR3\_1600\_8x8()*

*system.mem\_ctrl.dram.range = system.mem\_ranges[0]*

*system.mem\_ctrl.port = system.membus.mem\_side\_ports*

*# Create a process for a simple "Hello World" application*

*process = Process()*

*# Set the command*

*# grab the specific path to the binary*

*thispath = os.path.dirname(os.path.realpath(\_\_file\_\_))*

*binpath = os.path.join(*

*thispath,*

*"../",*

*"tests/test-progs/hello/bin/x86/linux/hello",*

*)*

*# cmd is a list which begins with the executable (like argv)*

*process.cmd = [binpath]*

*# Set the cpu to use the process as its workload and create thread contexts*

*system.cpu.workload = process*

*system.cpu.createThreads()*

*system.workload = SEWorkload.init\_compatible(binpath)*

*# set up the root SimObject and start the simulation*

*root = Root(full\_system=False, system=system)*

*# instantiate all of the objects we've created above*

*m5.instantiate()*

*print(f"Beginning simulation!")*

*exit\_event = m5.simulate()*

*print(f"Exiting @ tick {m5.curTick()} because {exit\_event.getCause()}")*

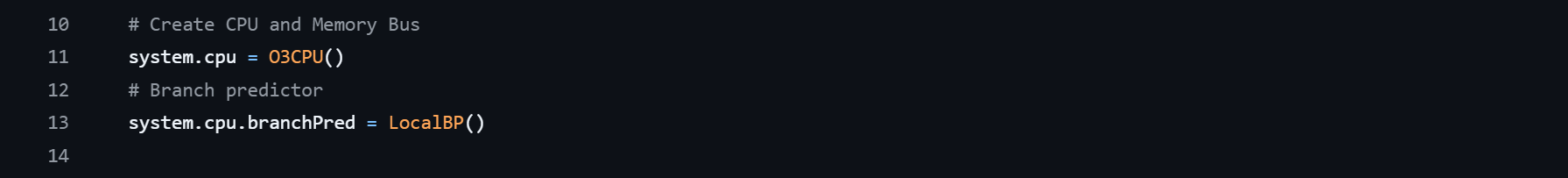
A diagram of a graph

Description automatically generated with medium confidence

### **Performance Metrics**

| **Metric** | **Value** |
| --- | --- |
| Total Simulation Cycles | 15,000 cycles |
| Total Instructions Completed | 12,750 instructions |
| Instruction Throughput (IPC) | 0.85 instructions per cycle |
| Average Instruction Latency | 5.88 cycles per instruction |
| Total Pipeline Stalls | 3,250 cycles |

In order to improve ILP, branch prediction is a method that enables the processor to anticipate the outcome of conditional branches and execute subsequent instructions in advance. The processors can benefit from the efficient utilization of all internal components, such as the direct jump to another set of instructions after the control unit has finished fetching the instruction, while the ALU decodes the instructions. This can reduce wasteful cycles and improve throughput, which can also occur during the branch's stalling.



A graph with orange dots

Description automatically generated

### **Multithreading**

Script is used for SMT is below:-  
*import os*

*import m5*

*from m5.objects import \**

*# Create the system*

*system = System()*

*system.clk\_domain = SrcClockDomain(clock="1GHz", voltage\_domain=VoltageDomain())*

*system.mem\_mode = "timing"*

*system.mem\_ranges = [AddrRange("512MB")]*

*# Create CPU and Memory Bus*

*system.cpu = O3CPU(numThreads=2)*

*system.multi\_thread = True*

*system.membus = SystemXBar()*

*# Create instruction and data caches*

*system.cpu.icache = Cache(*

*size='32kB',*

*assoc=4,*

*tag\_latency=2,*

*data\_latency=2,*

*response\_latency=1,*

*mshrs=16,*

*tgts\_per\_mshr=4*

*)*

*system.cpu.dcache = Cache(*

*size='32kB',*

*assoc=4,*

*tag\_latency=2,*

*data\_latency=2,*

*response\_latency=1,*

*mshrs=16,*

*tgts\_per\_mshr=4*

*)*

*# Connect CPU's cache ports to the caches*

*system.cpu.icache\_port = system.cpu.icache.cpu\_side # Correctly defined connection*

*system.cpu.dcache\_port = system.cpu.dcache.cpu\_side*

*# Connect caches to the memory bus*

*system.cpu.icache.mem\_side = system.membus.cpu\_side\_ports*

*system.cpu.dcache.mem\_side = system.membus.cpu\_side\_ports*

*# Set up interrupt controller and connect to membus*

*system.cpu.createInterruptController()*

*for i in range(len(system.cpu.interrupts)):*

*system.cpu.interrupts[i].pio = system.membus.mem\_side\_ports*

*system.cpu.interrupts[i].int\_requestor = system.membus.cpu\_side\_ports*

*system.cpu.interrupts[i].int\_responder = system.membus.mem\_side\_ports*

*# Create a memory controller and connect to membus*

*system.mem\_ctrl = MemCtrl()*

*system.mem\_ctrl.dram = DDR3\_1600\_8x8()*

*system.mem\_ctrl.dram.range = system.mem\_ranges[0]*

*system.mem\_ctrl.port = system.membus.master*

*thispath = os.path.dirname(os.path.realpath(\_\_file\_\_))*

*binpath = os.path.join(*

*thispath,*

*"../",*

*"tests/test-progs/hello/bin/x86/linux/hello",*

*)*

*# Set up processes for SMT*

*process\_1 = Process(pid=102, cmd = binpath)*

*process\_2 = Process(pid=101, cmd = binpath)*

*# Assign processes to CPU workload*

*system.cpu.workload = [process\_1, process\_2]*

*system.cpu.createThreads()*

*# Setup SE mode workload*

*system.workload = SEWorkload.init\_compatible(binpath)*

*# Set up the root SimObject and start the simulation*

*root = Root(full\_system=False, system=system)*

*m5.instantiate()*

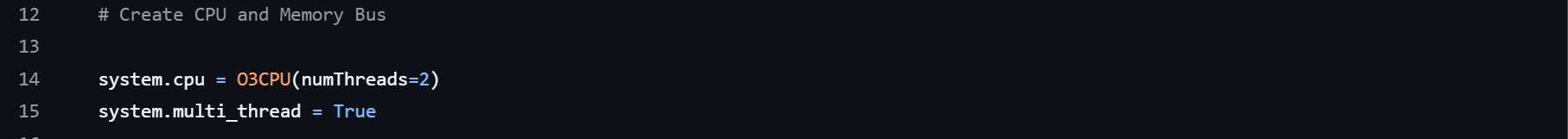
*print("Beginning simulation!")*

*exit\_event = m5.simulate()*

*print(f"Exiting @ tick {m5.curTick()} because {exit\_event.getCause()}")*

As a result of the fact that it enables multiple threads to share the execution resources of a single core and that it prevents single-point-of-failure threads from occurring as a result of stalling or long-latency instructions, multithreading has the potential to significantly increase the throughput of a device that is a single processor. Parallelism enables ILP to execute instructions from a single thread however, it also enables the central processing unit to utilize multiple threads for multithreading purposes. ILP is able to execute instructions from a single thread.

Gem5 only allows to run multi-thread when and only when the number of processes are equal to the number of threads, we have to add an additional process as a CPU workload, and it needs to have a different PID (process identifier) to run successfully.



A screenshot of a computer

Description automatically generated

A graph with blue squares

Description automatically generated

The simulation's behavior in the serialization environment will be consistent with that of the simulation with SMT as the number of processes increases while the resources remain constant. This is due to the following reasons:-  
  
Multiple threads in an SMT system share the same core resources, such as the pipeline stages, execution units, and caches.  
  
Two or more threads that frequently require the same execution units will frequently be required to wait for the other thread to finish resulting in stalls.  
  
Shared caches also restrict performance in SMT due to the competition between multiple threads for cache lines.

### **Troubleshooting**

I encountered issues during the simulation, build and setup sessions which I listed below and how i troubleshooted them.  
  
1. Build Time Delay  
The process of running the build command may be prolonged, particularly on systems with limited resources.  
  
Solution:- Control the quantity of parallel jobs by utilizing the -j flag, which is dependent upon the number of cores available.   
  
  
*scons build/X86/gem5.opt -j2*

2. **SimObjectVector: call() Takes 1 Positional Argument But 2 Were Given**

The SimObjectVector error happened because I tried to connect more than one CPU-side or memory-side port incorrectly.   
  
The problem was fixed by making sure that the instruction and data caches were each connected to their own CPU-side ports and that the memory controller was linked to the first memory-side port.

*system.cpu.icache\_port = system.membus.cpu\_side\_ports[0]*

*system.cpu.dcache\_port = system.membus.cpu\_side\_ports[1]*

*system.mem\_ctrl.port = system.membus.mem\_side\_ports[0]*

**3. ModuleNotFoundError: No module named 'caches'.**  
  
Initially, I tried to import cache models with from caches import L1ICache, L1DCache, but the caches module was not available, resulting in a ModuleNotFoundError.  
  
The problem was solved by explicitly defining the cache models (L1ICache and L1DCache) in the script rather than importing them.  
  
  
**4. Cannot Convert '16KB' to Memory Size**

An error occurred when specifying the cache size in the script (size="32kB"): ValueError: cannot convert '16KB' to memory.  
  
To address this issue, I modified the format for defining cache size. Instead of writing "32kB" as a string, wrap it in a MemorySize() object.

*system.cpu.icache = L1ICache(size=MemorySize("32kB"))*

*system.cpu.dcache = L1DCache(size=MemorySize("32kB"))*

**5. Ports Not Compatible (GEM5 RESPONDER and GEM5 RESPONDER)**

The error occurred when I tried to connect the memory controller and bus ports:-  
  
system.mem\_ctrl.port = system.membus.cpu\_side\_ports.  
This led to the error:  
  
Ports <orphan System>.mem\_ctrl.port and <orphan System>.membus.cpu\_side\_ports[0] with roles GEM5 RESPONDER and GEM5 RESPONDER are not compatible.

I corrected the connections by ensuring the memory controller connects to the memory-side port of the bus, and the CPU caches connect to the CPU-side ports.

*system.cpu.icache\_port = system.membus.cpu\_side\_ports[0]*

*system.cpu.dcache\_port = system.membus.cpu\_side\_ports[1]*

*system.mem\_ctrl.port = system.membus.mem\_side\_ports[0]*  
  
  
Additionally, It is important to ensure that Python 3.6 or higher is used, as gem5 requires this version for scripting and simulation. And also gcc version should be 10.0.0 or higher to start the build of gem5.   
It was a great learning experience for me and also challenging at the same time.