Architetture dei Sistemi di Elaborazione 02GOLOV [A-L]	Delivery date: November 12, 2020
Laboratory **4**	Expected delivery of lab_04.zip must include: - this document compiled in pdf format.

1) Getting started with gem5
gem5 is an event-driven simulator freely available at: `http://gem5.org/`
The laboratory version uses the ALPHA CPU model previously compiled.

From Portale della Didattica, download the gem5_env_2020.zip. Decompress it in your home directory.

NOTE: All the commands shown here must be executed from the terminal.

Preliminarily, set up the environment variables executing the following command:
`source start.sh` (**NOTE, if you are using the VBox VM, replace this command with** `source start_vbox.sh`**)**
These effects of these script <u>are visible only in the current shell</u>.

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ source start.sh
Setting up the environment...
```

a. Write a hello world C program (`hello.c`). Then compile the program, using the ALPHA compiler with the command `gem5_alpha_compiler`. The compiler is the gcc version for the ALPHA ISA, therefore it is used with the same options:

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_alpha_compiler -static -
o hello hello.c
```

b. Then simulate the program with the `gem5_sim` command as follows:

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY -c hello
```

In this simulation, gem5 uses *AtomicSimpleCPU* by default.

c. Check the results
your simulation output should be similar than the one provided in the following:

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY -c hello
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jan 17 2019 11:54:22
gem5 started Oct 30 2020 11:29:05
gem5 executing on ubuntu-desktop, pid 31977
command line: gem5.opt /opt/gem5/configs/example/se.py -c hello

/opt/gem5/configs/common/CacheConfig.py:50: SyntaxWarning: import * only allowed at
module level
  def config_cache(options, system):
Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned
(512 Mbytes)
warn: Breakpoints do not work in Alpha PAL mode.
      See PCEventQueue::doService() in cpu/pc_event.cc.
```

```
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
info: Increasing stack size by one page.
Hello There!!
Exiting @ tick 2411000 because exiting with last active thread context
```

- Check the output folder

in your working directory, gem5 creates an output folder (`m5out`), and saves there 3 files: `config.ini`, `config.json`, and `stats.txt`. In the following, **some examples** of the produced files are reported.

- Statistics (`stats.txt`)

```
---------- Begin Simulation Statistics ----------
sim_seconds             0.000003      # Number of seconds simulated
sim_ticks                2623000      # Number of ticks simulated
final_tick               2623000      # Number of ticks from beginning of simulation
sim_freq           1000000000000      # Frequency of simulated ticks
host_inst_rate           1128003      # Simulator instruction rate (inst/s)
host_op_rate             1124782      # Simulator op (including micro ops) rate(op/s)
host_tick_rate         564081291      # Simulator tick rate (ticks/s)
host_mem_usage            640392      # Number of bytes of host memory used
host_seconds                0.00      # Real time elapsed on the host
sim_insts                   5217      # Number of instructions simulated
sim_ops                     5217      # Number of ops (including micro ops) simulated
... ... ...
system.cpu_clk_domain.clock 500       # Clock period in ticks
... ... ...
```

- Configuration file (`config.ini`)

```
... ... ...
[system.cpu]
type=AtomicSimpleCPU
children=dtb interrupts isa itb tracer workload
branchPred=Null
checker=Null
clk_domain=system.cpu_clk_domain
cpu_id=0
default_p_state=UNDEFINED
do_checkpoint_insts=true
do_quiesce=true
do_statistics_insts=true
dtb=system.cpu.dtb
eventq_index=0
fastmem=false
function_trace=false
```

2) Simulate the same program using different CPU models.

Help command:

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY -h
```

List the CPU available models:

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY --list-cpu-types
```

a. *TimingSimpleCPU* simple CPU that includes an initial memory model interaction

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY --cpu-type=TimingSimpleCPU -c hello
```

b. *MinorCPU* the CPU is based on an in order pipeline including caches

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY --cpu-type=MinorCPU --caches -c hello
```

c. *DerivO3CPU* is a superscalar processor

```
labinf@ubuntu-desktop:~/Desktop/gem5_env_2020$ gem5_sim $GEM5_DEFAULT_PY --cpu-type=DerivO3CPU --caches -c hello
```

To practice with the generated statistics, create a table (TABLE1) gathering for each simulated CPU the following statistics (**when available!**):

- `sim_ticks` (Number of ticks simulated)
- `sim_insts` (Number of instructions simulated)
- `system.cpu.numCycles` (Number of CPU Clock Cycles)
- `system.cpu.cpi` (Clock Cycles per Instruction)
- `system.cpu.committedInsts` (Number of instructions committed)
- `host_seconds` (Host time in seconds)
- `system.cpu.fetch.Insts` (Number of instructions Fetch Unit has encountered)

TABLE1

Parameters	AtomicSimpleCPU	TimingSimpleCPU	MinorCPU	DeriveO3CPU
sim_ticks	2755000	356448000	31495500	18533000
sim_insts	5477	5477	5490	5278
system.cpu.numCycles	5511	712896	62991	37067
system.cpu.cpi	1.006	130.16	11. 473770	7. 022925
system.cpu.committedInsts	5477	5477	5490	5476
host_seconds	0.02	0.03	0.04	0.06
system.cpu.fetch.Insts			9245	11124

NOTE: When not available compute the CPI using the formula:

$$\frac{\text{system. cpu. numCycles}}{\text{system. cpu. committedInsts}}$$

3) Let's now switch to a slightly more complex benchmark: the computation of a Fast Fourier Transform. The program is in the *`benchmarks/fft`* subdirectory and can be compiled using the MakeFile with the commands `make clean` and then `make` that will produce as output the `fft` executable file.

   Simulate the program using the gem5 CPU models seen before and collect the following information (when available!) filling TABLE 2:

   - `sim_ticks` (Number of ticks simulated)
   - `sim_insts` (Number of instructions simulated)
   - `system.cpu.numCycles` (Number of CPU Clock Cycles)
   - `system.cpu.cpi` (Clock Cycles per Instruction)
   - `system.cpu.committedInsts` (Number of instructions committed)
   - `host_seconds` (Host time in seconds)

- `system.cpu.fetch.Insts` (Number of instructions Fetch Unit has encountered)
- Prediction ratio for Conditional Branches:
  `system.cpu.branchPred.condIncorrect` / `system.cpu.branchPred.condPredicted`
- `system.cpu.branchPred.BTBHits` (Number of BTB hits)

TABLE2:

Parameters	AtomicSimpleCPU	TimingSimpleCPU	MinorCPU	DeriveO3CPU
sim_ticks	10678466000	1266139855000	11824593000	5607559500
sim_insts	21356881	21356881	21356902	20972488
system.cpu.numCycles	21356933	2532279710	23649186	11215120
system.cpu.cpi	1	118.57	1.107332	0. 534754
system.cpu.committedInsts	21356881	21356881	21356902	20972488
host_seconds	9.47	53.59	39.67	50.86
system.cpu.fetch.Insts			21356902	24132664
Pred. ratio Cond. Branches			4.37%	4%
system.cpu.branchPred.BTBHits			1444140	1562500

4) Compare Table 1 and 2. Why the instructions encountered by the Fetch Unit differ from the instruction committed?

   Your Answer: I suppose that "instructions commited" means those instructions that reached the end of the operation. If this is right, the number of fetched instructions and committed ones differ because some instructions are fetched but incorrects predicted jump make some fetched instructions useless because they shall not be executed.

   HINTS: If you are thinking to use a bash script to automatically run and gather the statistics from the simulations, you might encounter some troubles since the commands listed above are actually aliases of more complex commands (see the `start.sh` for details). To have the different aliases visible from a bash script, put the following commands at the top of your script:

```
#!/bin/bash

shopt -s expand_aliases
source start.sh          # or start_vbox.sh

# Here starts your own script…
```

Instructions for importing the VBox VM

1. Import the virtual machine in VirtualBox (https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html)
2. The virtual machine can be downloaded using the following link:
   - https://baltea.polito.it/owncloud/index.php/s/SbJPJb6kQW7mcze

3. Log in using the following credentials:
    - Account: gem5
    - Password: gem5