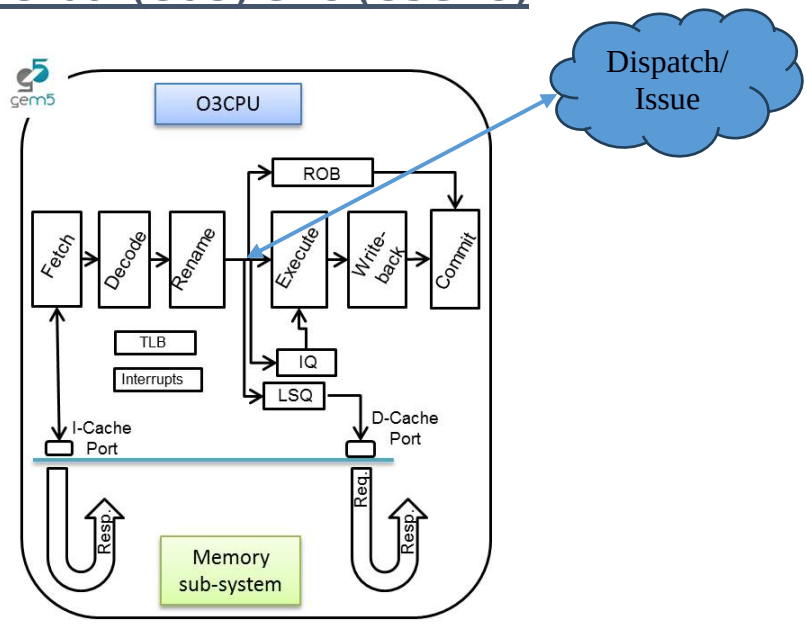
| | |
|---|---|
| **Architetture dei Sistemi di Elaborazione**<br>**O2GOLOV**<br><br>**Laboratory 4** | Delivery date:<br>**Until 2 AM of October 30th 2024**<br><br>Expected delivery of **lab_4.zip** must include:<br>• each configuration of the custom architecture (riscv_o3_custom.py) that you modify.<br>• This document with all the field compiled and in PDF form. |

# Introduction and Background

## Simulating an Out-of-Order (OoO) CPU (O3CPU)




In this laboratory, you will be able to configure an OoO CPU by using a script called riscv_o3_custom.py. In a few words, the script configures an Out-of-Order (O3) processor based on the *DerivO3CPU*, a superscalar processor with a reduced number of features.

**Pipeline**

The processor pipeline stages can be summarized as:

- **Fetch stage:** instructions are fetched from the instruction cache. The `fetchWidth` parameter sets the number of fetched instructions. This stage does branch prediction and branch target prediction.
- **Decode stage:** This stage decodes instructions and handles the execution of unconditional branches. The `decodeWidth` parameter sets the maximum number of instructions processed per clock cycle.
- **Rename stage:** As suggested by the name, registers are renamed, and the instruction is pushed to the IEW (Issue/Execute/Write Back) stage. It checks that the *Instruction Queue* (**IQ**)/*Load and Store Queue* (**LSQ**) can hold the new instruction. The maximum number of instructions processed per clock cycle is set by the `renameWidth` parameter.

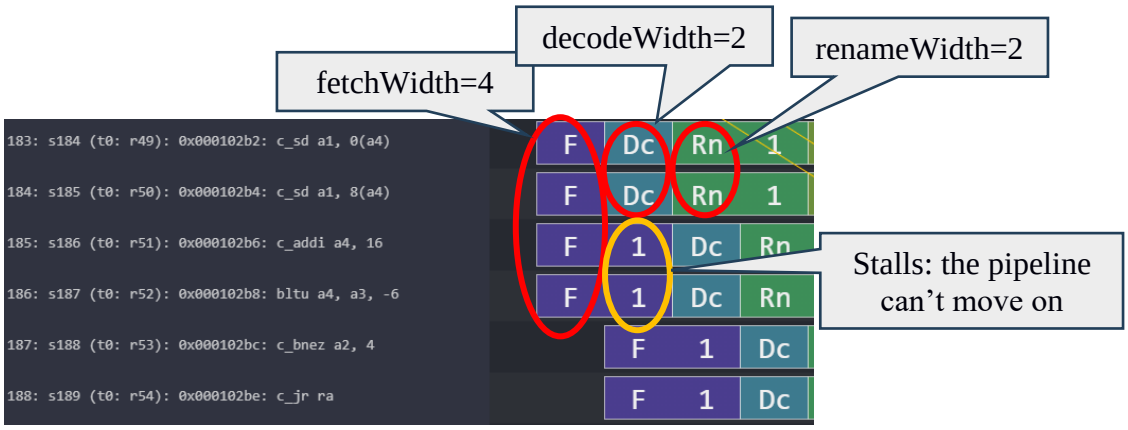



*Figure 1: Understanding configurable OoO CPU parameters.*

- **Dispatch stage**: instructions whose renamed operands are available are dispatched to functional units (**FU**). For loads and stores, they are dispatched to the Load/Store Queue (**LSQ**). The maximum number of instructions processed per clock cycle is set by the `dispatchWidth` parameter.
- **Issue stage**: The simulated processor has a single instruction queue from which all instructions are issued. Ordinarily, instructions are taken in-order from this queue. An instruction is issued if it does not have any dependency.
- **Execute stage:** the functional unit (**FU**) processes their instruction. Each functional unit can be configured with a different latency. Conditional branch mispredictions are identified here. The maximum number of instructions processed per clock cycle depends on the different functional units configured and their latencies.
- **Writeback stage**: it sends the result of the instruction to the reorder buffer (**ROB**). The maximum number of instructions processed per clock cycle is set by the `wbWidth` parameter.
- **Commit stage**: it processes the reorder buffer, freeing up reorder buffer entries. The maximum number of instructions processed per clock cycle is set by the `commitWidth` parameter. Commit is done in order.

In the event of a **branch misprediction**, trap, or other speculative execution event, "squashing" can occur at all stages of this pipeline. When a pending instruction is squashed, it is removed from the instruction queues, reorder buffers, requests to the instruction cache, etc.


*Figure 2: Example of a branch **misprediction** (transparent rows)*

**Pipeline Resources**

Additionally, it has the following structures:

- Branch predictor (BP)
  - Allows for selection between several branch predictors, including a local predictor, a global predictor, and a tournament predictor. Also has a branch target buffer (BTB) and a return address stack (RAS).
- Reorder buffer (ROB)
  - Holds instructions that have reached the back end. Handles squashing instructions and keep instructions in program order.
- Instruction queue (IQ)
  - Handles dependencies between instructions and scheduling ready instructions. Uses the **memory dependence predictor** to tell when memory operations are ready.
- Load-store queue (LSQ)
  - Holds loads and stores that have reached the back end. It hooks up to the d-cache and initiates accesses to the memory system once memory operations have been issued and executed. Also handles forwarding from stores to loads, replaying memory operations if the memory system is blocked, and detecting memory ordering violations.
- Functional units (FU)
  - Provides timing for instruction execution. Used to determine the latency of an instruction executing, as well as what instructions can issue each cycle.
  - **Floating point units, floating point registers,** and respective instructions are supported.


*Figure 3: Pipeline example of FP instructions and FP registers*

# Laboratory: hands-on

All the needed resources are at a GitHub repository:
https://github.com/cad-polito-it/ase_riscv_gem5_sim

To create your simulation environment:
For HTTPS clone:

```
~/my_gem5Dir$ git clone https://github.com/cad-polito-it/ase_riscv_gem5_sim.git
```

For SSH:

```
~/my_gem5Dir$ git clone git@github.com:cad-polito-it/ase_riscv_gem5_sim.git
```

The environment is configured to be executed on the LABINF MACHINES.

Follow the HOWTO instructions available on the GitHub Repository for simulating a program.

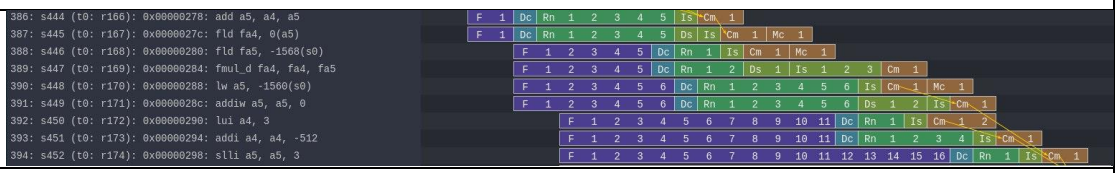
## Exercise 1:

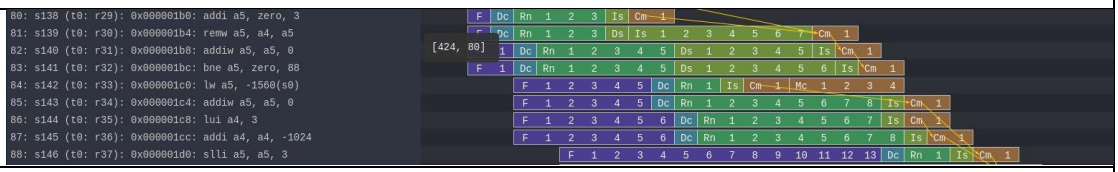
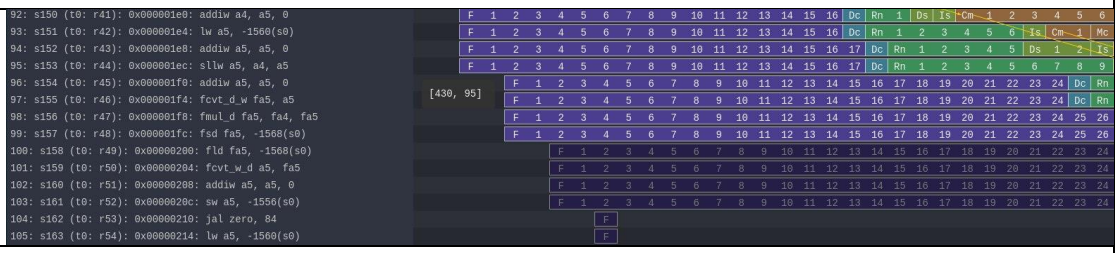
Simulate the benchmark *my_c_benchmark_2* (*main.c*) by using the gem5 simulator to obtain the *trace.out* file. Then, you can visualize the pipeline (i.e., load the *trace.out* file on Konata).

Based on the CPU architecture described in *riscv_o3_custom.py*, visualize the Konata's pipeline to find out the conditions:

1. Out-of-order execution (issue), in-order commit (commit)
2. Two commits in the same clock cycle
3. Flush of the pipeline.

For every condition, fill the following tables.

| **Condition** | Out-of-order execution, in-order commit |
|---|---|
| **Screenshot from Konata** |  |
| **Explain the reason behind the condition** | A riga 391 si può osservare una OoO. Infatti l'istruzione successiva viene eseguita mantra quella precedente è in stallo. |

| | |
|---|---|
| **Briefly explain the advantages of the OoO execution in a CPU** | Il vantaggio principale di eseguire istruzione “out of order operations” è che viene ridotto il numero complessivo di cicli di clock causati per esempio da hazard strutturali grazie all’esecuzione in parallelo. |
| **Condition** | Two or more commits in the same clock cycle |
| **Screenshot from Konata** |  |
| **Explain the reason behind the condition** | A riga 85 e 86 si vedono due istruzioni che eseguono nello stesso ciclo di clock l’istruzione di commit grazie al fatto che il ‘commit width’ è Maggiore di 1. |
| **Briefly explain the Commit functioning** | In questi tipi di processor è possibile avere due commit nello stesso clock cycle, ma sempre in ordine |
| **Condition** | Flush of the pipeline |
| **Screenshot from Konata** |  |
| **Explain the reason behind the condition** | In questo caso il branch è stato preso e quindi la pipeline viene flushata. |

## Exercise 2:

Given your benchmark (*main.c* in *my_c_benchmark_2*), optimize the CPU architecture (i.e., modify the *riscv_o3_custom.py* file) and write down the improvements in terms of CPI and speedup.

- To optimize the CPU architecture, open the configuration file of the CPU (i.e., the *riscv_o3_custom.py)*, and tune specific hardware-related parameters.

  You have to change specific values in **one or more** stages of the pipeline:

  - # - FETCH STAGE
    - Tune parameters such as the *fetchWidht*, *fetchBuffersize* and so on, and see the effects on your system.
  - # - DECODE STAGE
  - # - RENAME STAGE
    - Try changing some values, but don't touch the "Phys" ones.
  - # - DISPATCH/ISSUE STAGE
  - # - EXECUTE STAGE
    - Here you can optimize the Functional units of your CPU like the INT ALU, the FP ALU, the FP Multiplier/Divider and so on.
    - Tune the number of units (*count*) that you have in the system, as well as their latency (*opLat*) to see how this affects the execution of your program.

  - You can create a different branch predictor. They are defined in *create_predictor.py)*

  - You can also try to change the parameters of the L1 Cache. Look for the "class L1Cache" in the *riscv_o3_custom.py* file. The L1 cache, also referred to as the primary cache, is the smallest and fastest level of memory. It is located directly on the processor, and it is used to store frequently accessed data by the CPU. In this way, the CPU saves time with respect to the normal access to the main memory.

**HINT:** To implement the best hardware optimization, and understand how to change the parameters, the best option consists in analysing the *stats.txt* file (in **ase_riscv_gem5_sim/results/my_c_benchmark_2**).
Find information regarding the workload profiling. In other words, look for lines such as "system.cpu.commitStats0.committedInstType::**IntAlu**", and the following ones to understand which kind of instructions are executed the most. In this way, you can target a specific functional unit and modify its specifications.

Fill the following Tables with the CPI that you obtain with the old and the new architectures. Compute also the equivalent speedup that you obtain.
HINT: You can get the CPI and other useful information from the *stats.txt* file.

| **Parameters** | **Configuration 1** | **Configuration 2** | **Configuration 4** | **Configuration 5** |
|---|---|---|---|---|
| **First changed paramenter** | the_cpu.fetchWidth = 9 | the_cpu.issueWidth = 2 | renameWidth=8 | |
| **Second changed paramenter** | the_cpu.dispatchWidth =1 | None | decodeWidth=8 | numberOfIntAlu=7 |
| **…** | | None | | |
| | | | | |

Original CPI (no hardware optimization): 2.190180

| | **Configuration 1** | **Configuration 2** | **Configuration 4** | **Configuration 5** |
|---|---|---|---|---|
| **CPI** | 2.290411 | 2.190180 | 2.177433 | 2.190180 |
| **Speedup (wrt Original CPI)** | 0.956 | 1 | 1.006 | 1 |

Which is the best optimization in terms of CPI and speedup, why?

Your answer:

La migliore è la configurazione 4. Analizzando il codice si è visto però che ci sono molte istruzioni di calcolo intere quindi è ragionabile pensare che aumentando il numberOfIntAlu si otenga un CPI minore, ma non è così. Infatti ci sono dipendenze di dato che non permettono di ottenere questo grande speed up.