POLITECNICO DI TORINO

Master degree in Electronic Engineering - Embedded systems

Speed-up of RISC-V core using Logic-in-Memory operations

Supervisors:
Marco Vacca - Politecnico di Torino
Marco Ottavi - Università degli Studi di Roma Tor Vergata

Candidate:
Antonia Ieva - s253237

Agenda

Key points

- Possible solution to memory-wall problem in modern computing systems.
- RISC-V architecture as case study:
  - RISC-V standard is becoming mainstream because of its open source Instruction Set Architecture (ISA).
  - Extendable ISA that allows to customise its architecture and integrate tailored features.
- Logic-in-Memory integrated in RISC-V core to speed-up the execution of programs.

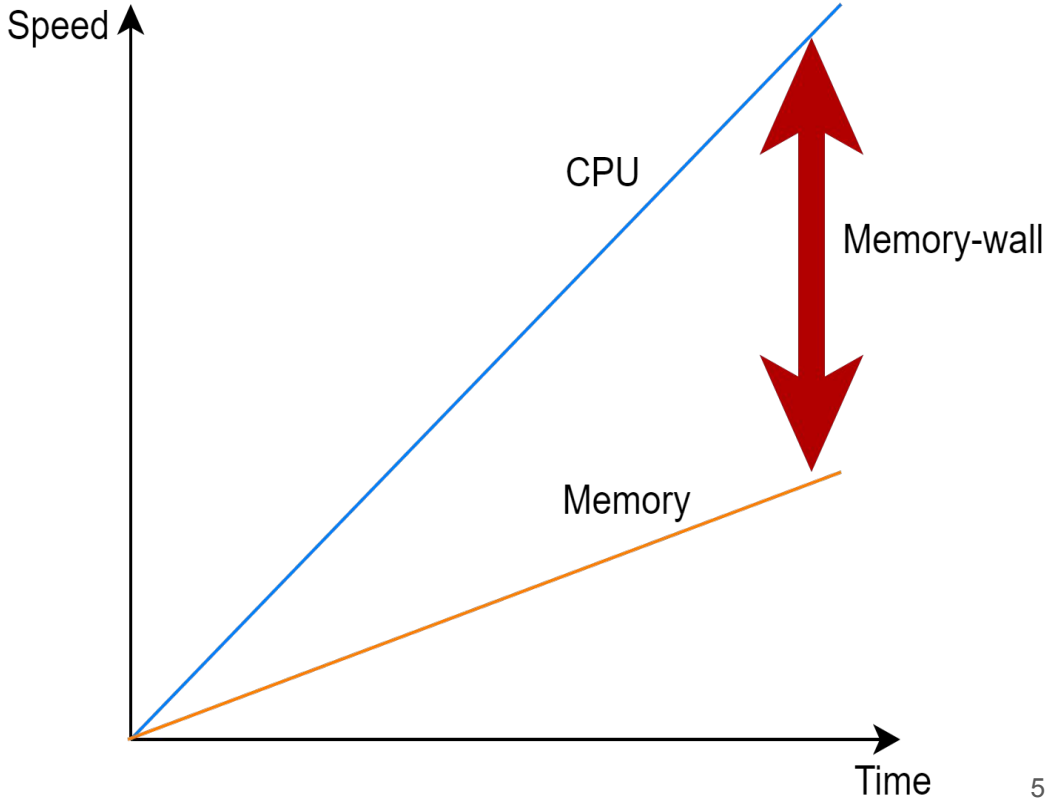
1. FRAMEWORK

1.1 Problem Statement

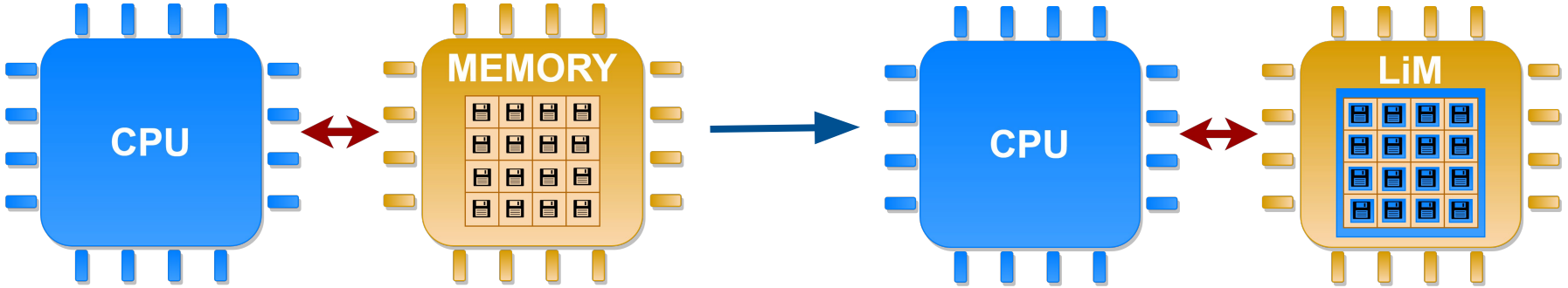
Modern computing systems present:

- *processor* for arithmetic operations.
- *memory* for instructions and data storage.

The ***memory-wall*** is a known issue that states the speed difference between memory and processor.



1.2 Logic-in-Memory concept



- **Logic-in-Memory** (LiM) is a specific memory architecture that combines storage and computational capabilities.
- The logic is added within the memory cell and around the memory array.
- Solution to memory-wall problem because of the reduced number of accesses in memory.

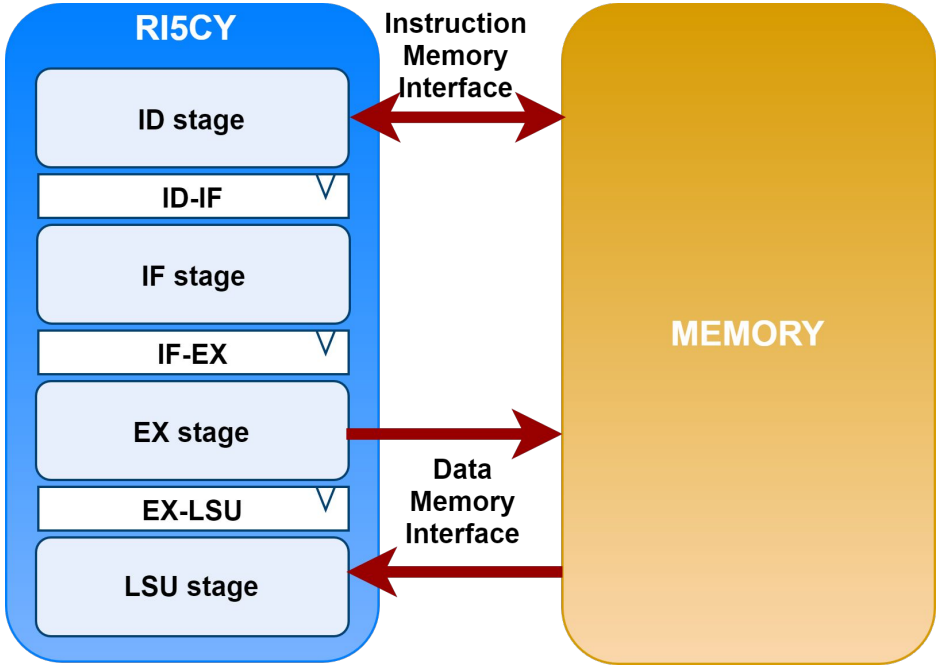
1.3 RI5CY core architecture

RISC-V standard:

- Open source Instruction Set Architecture (ISA) organised in extensions (standard and non-standard).

RI5CY core:

- Specific implementation of RISC-V ISA maintained by PULP platform.
- 4 stage pipelined architecture.
- Supporting standard extensions and non-standard extensions.



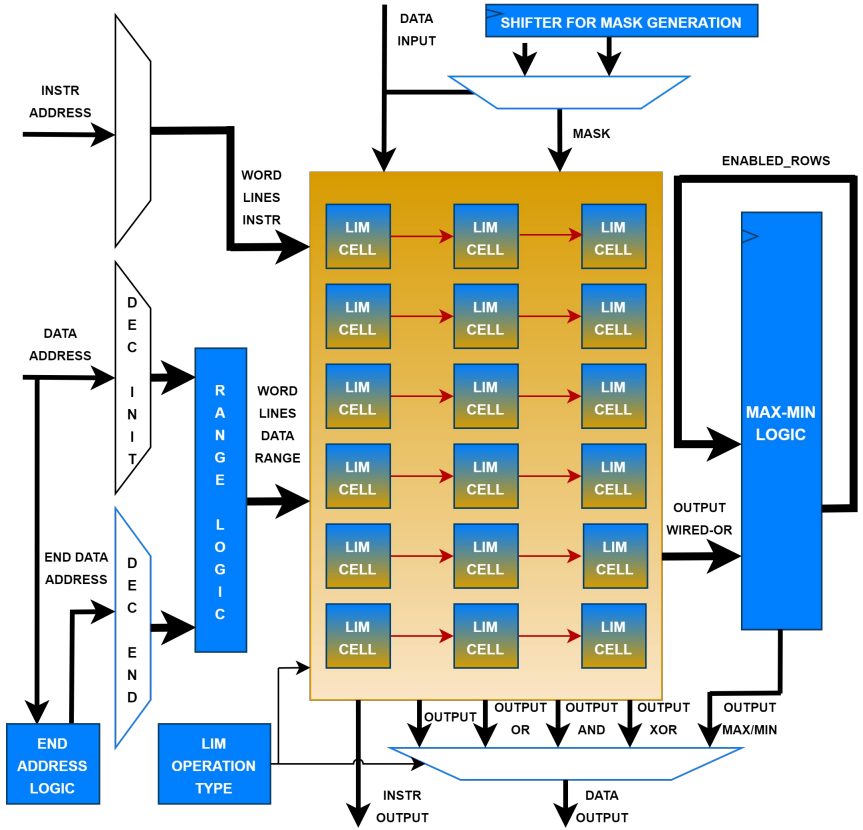
2. IMPLEMENTATION

2.1 Logic-in-Memory specifications

Memory capabilities:

- **Standard load and store** operations in 1 clock cycle.
- **Bitwise operations:** load and store AND, OR, XOR in 1 clock cycle.
  - Load operations only possible on 1 memory location.
  - Store operations (with the same input mask) possible on single or multiple memory locations.
- **Load maximum or minimum** value in 33 clock cycles.
  - MAX/MIN computed on a certain range of memory locations.
  - No need to reset the memory.

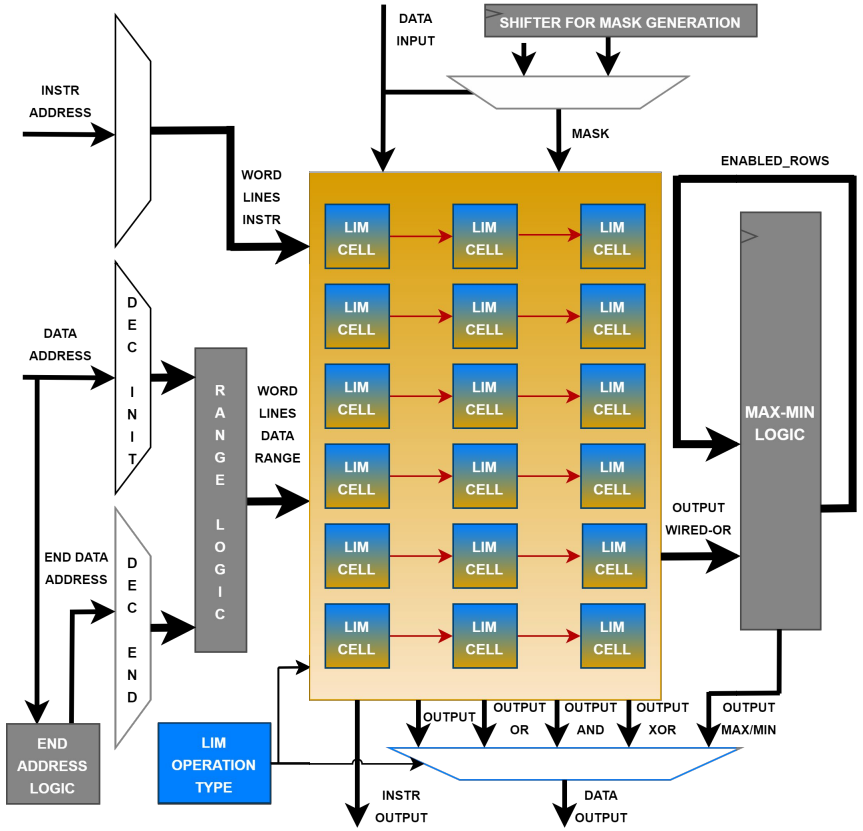
2.2 Logic-in-Memory implementation



To support the new operations logic has been added:

- Inside the memory cells.
- Around the memory array.

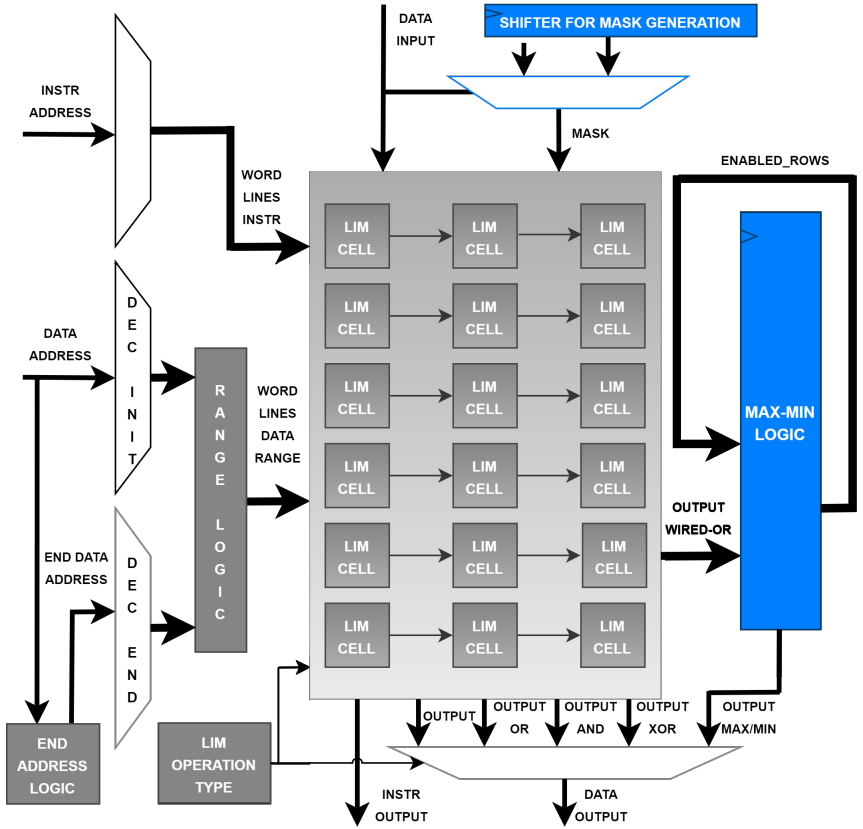
2.2 Logic-in-Memory implementation



Each memory cell is enlarged to host computational logic:

- LiM cell is the core of the bitwise operations.
- LiM cell performs a partial computation for the maximum and minimum algorithm.

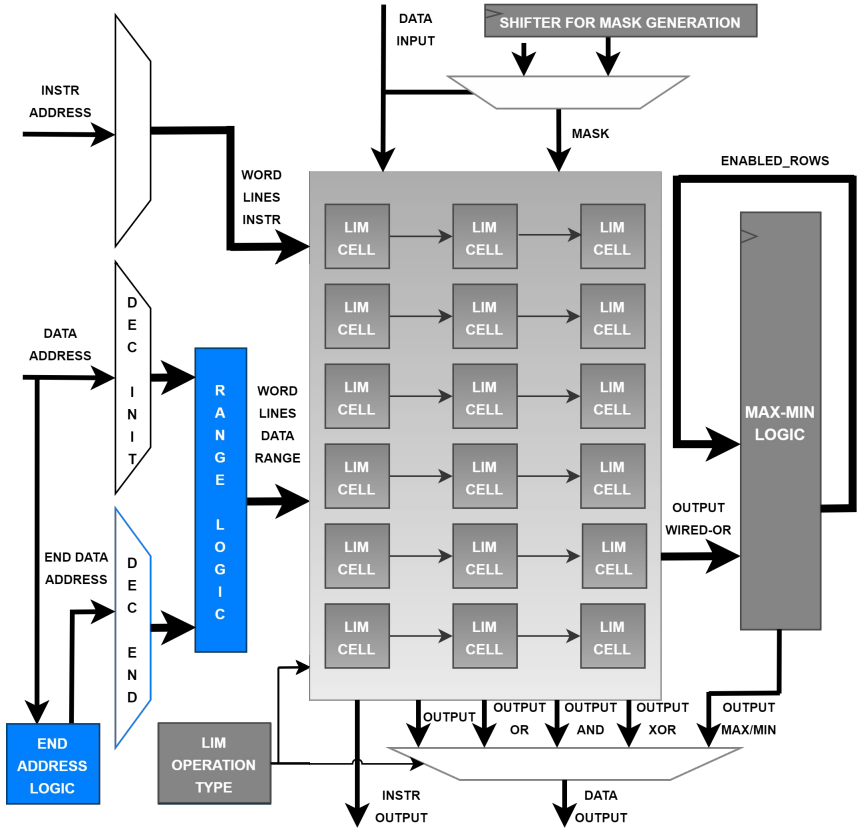
2.2 Logic-in-Memory implementation



Maximum and Minimum algorithm completed outside the memory array:

- Mask generator: different mask in each clock cycle, obtained through shifts.
- Logic that stores the information about the excluded words. This information is updated at each cycle.

2.2 Logic-in-Memory implementation

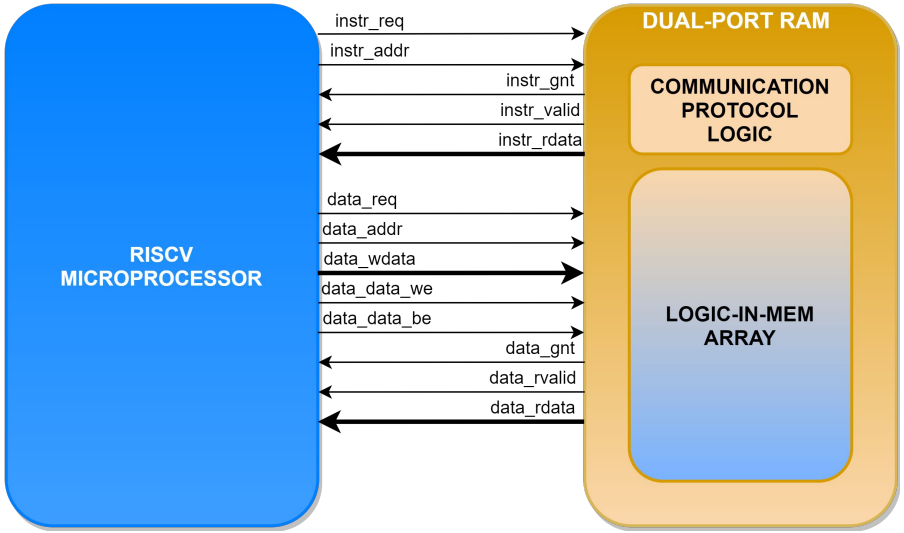


Range operations supported using:

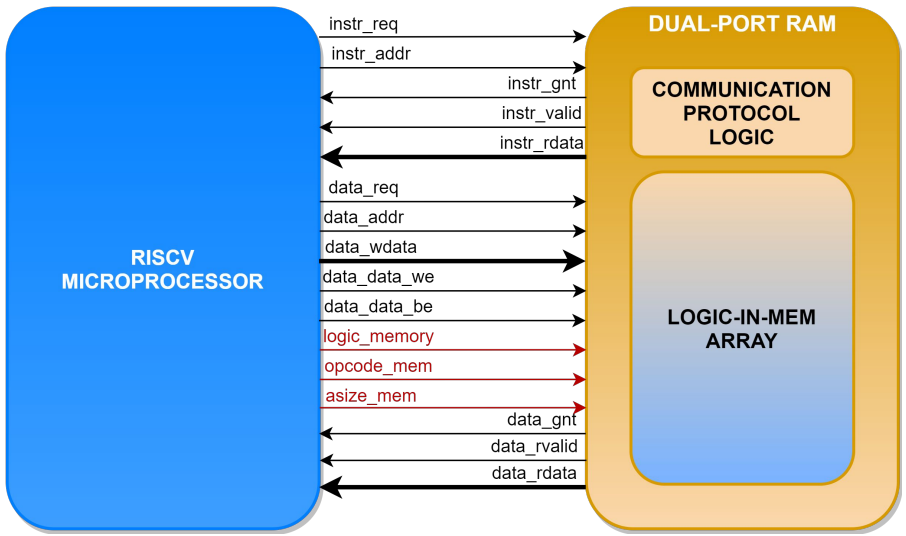
- Two decoders: for initial and end addresses.
- Combinational logic to enable all the all the word lines in between.

2.3 LiM integration in RI5CY

Same interface

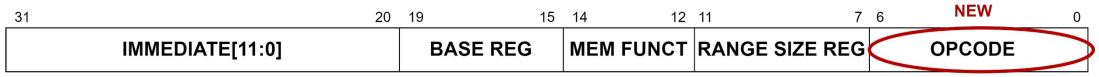


New interface

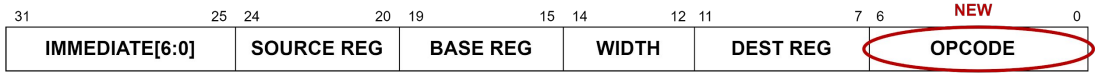


2.3.1 Same Interface approach

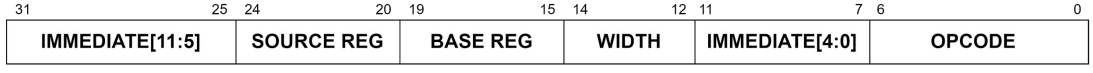
New ISA extension with new instructions:



STORE_ACTIVATE_LOGIC



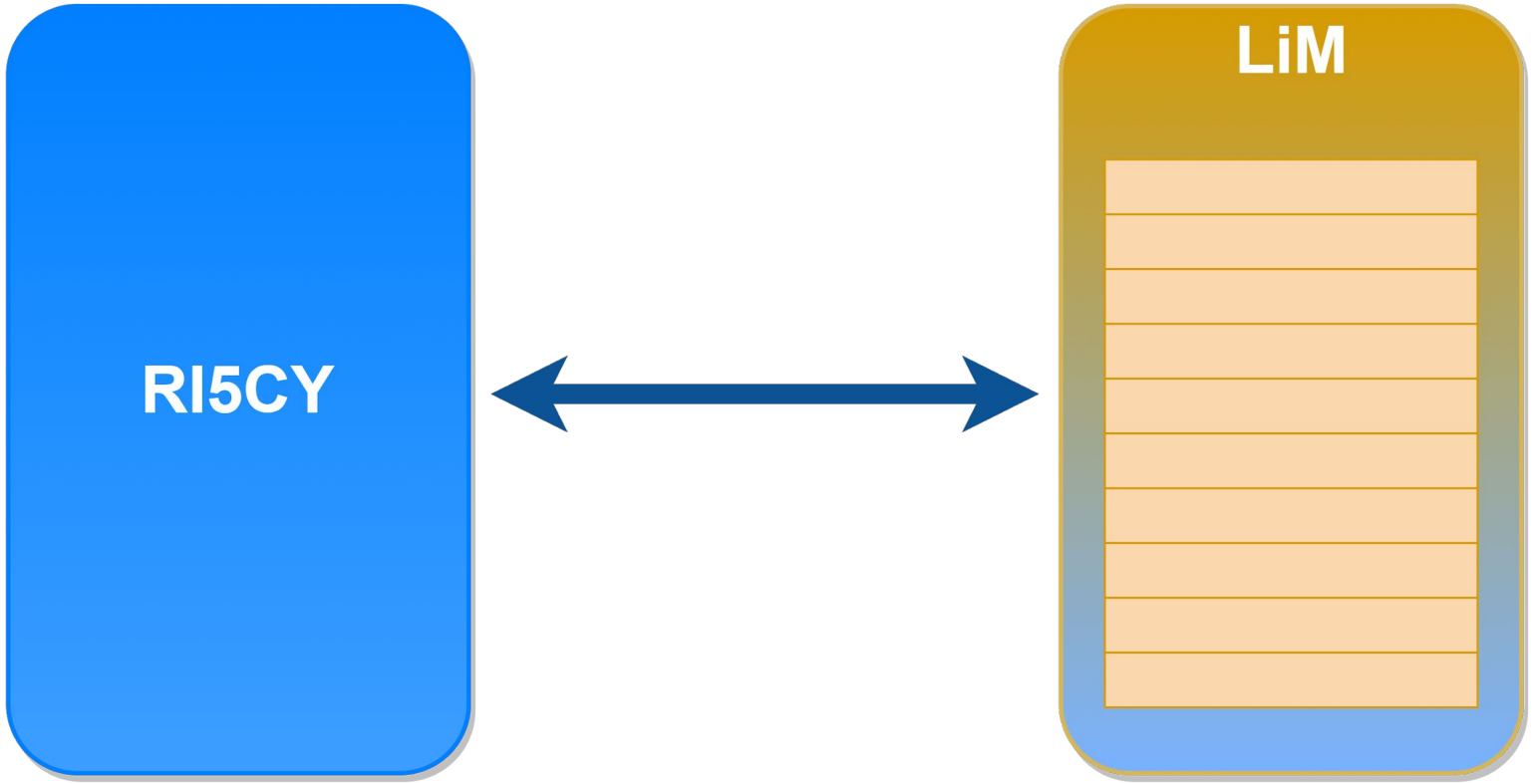
LOAD_MASK



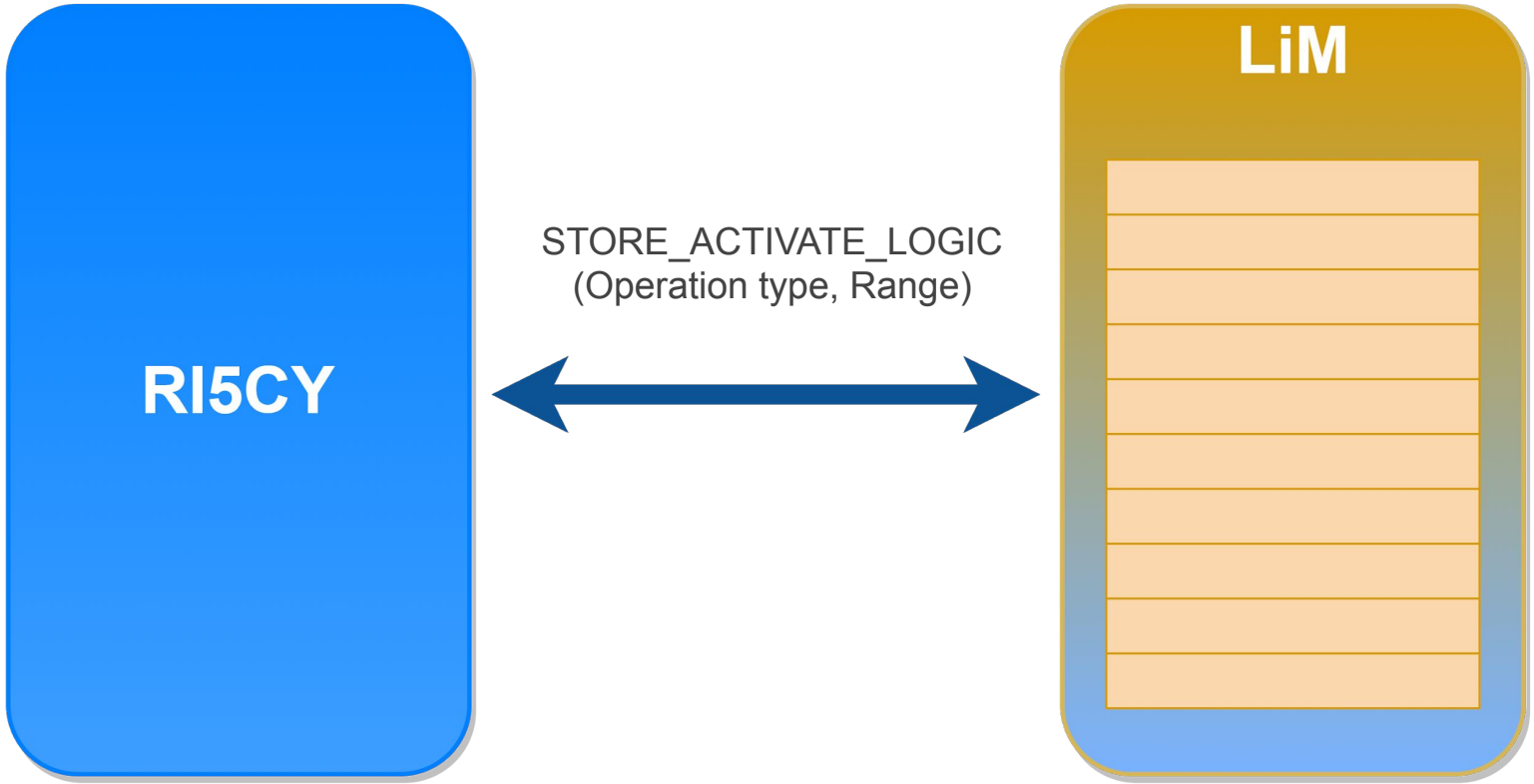
STORE

1. **STORE_ACTIVATE_LOGIC**: instruction to configure the memory operation. Information about:
   I. size of memory locations for the operation.
   II. operation type.
2. **LOAD_MASK:** normal load operation with input mask.
3. **STORE:** unchanged instruction.

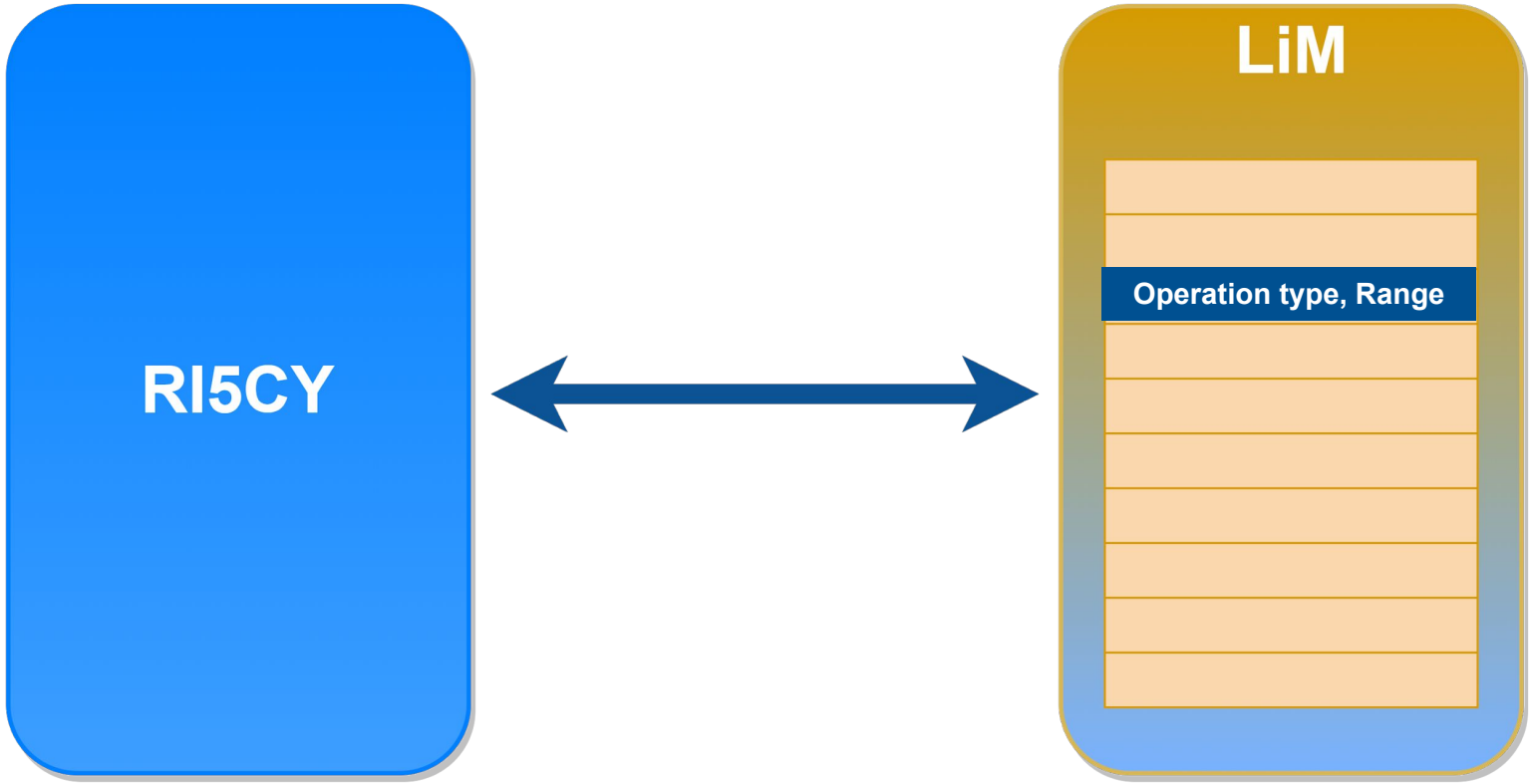
2.3.1 Same Interface approach



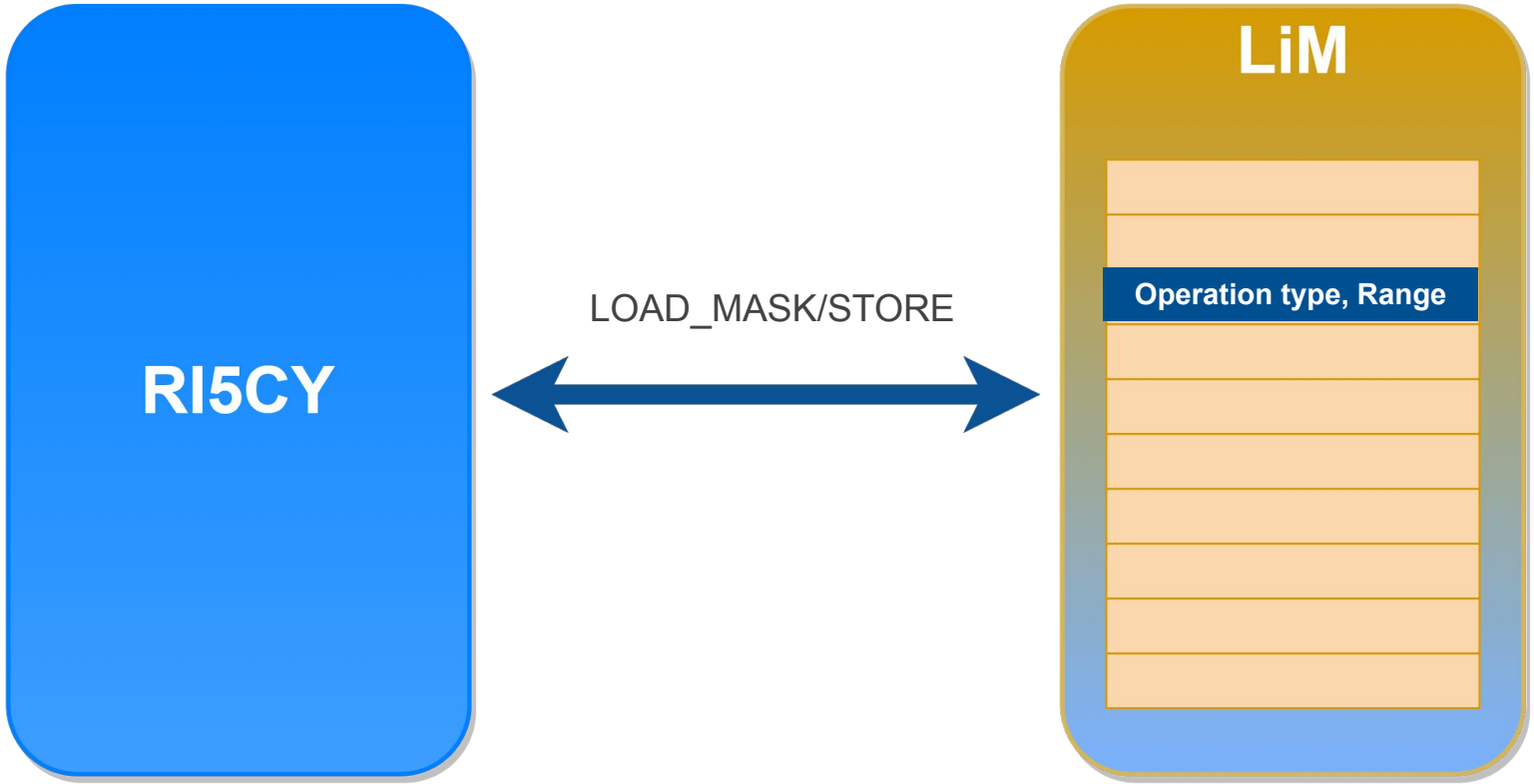
2.3.1 Same Interface approach



2.3.1 Same Interface approach

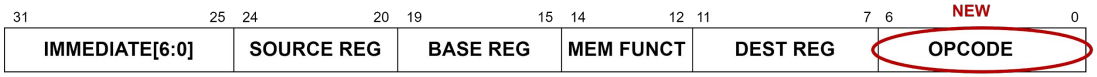


2.3.1 Same Interface approach

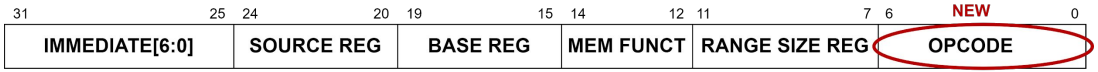


2.3.2 New Interface approach

New ISA extension with new instructions:



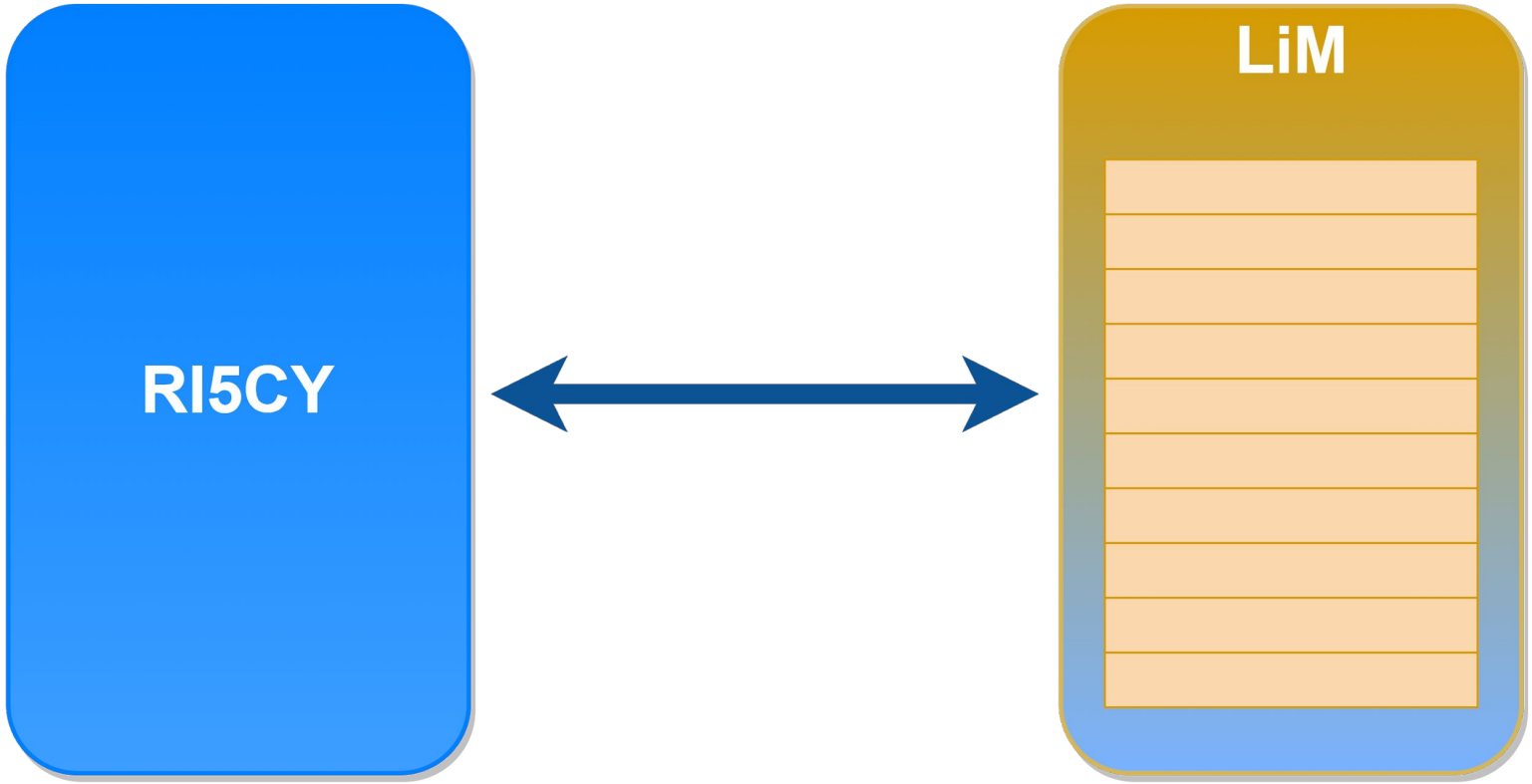
LOAD_LOGIC



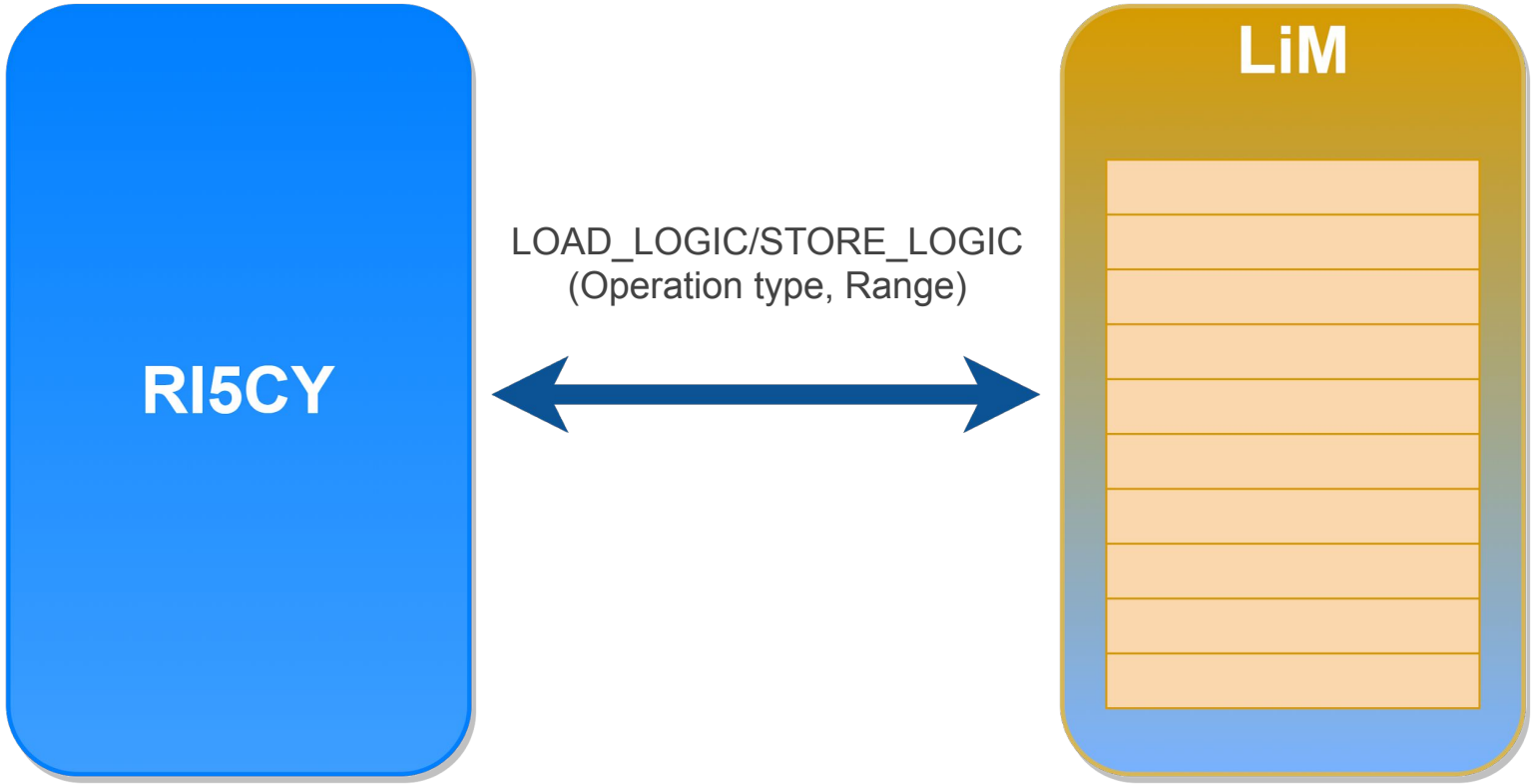
STORE_LOGIC

1. **STORE_LOGIC:** logic operations for single location or multiple locations.
2. **LOAD_LOGIC:** logic operation for single location in all the cases except for MAX/MIN, that is computed on a fixed size of locations.

2.3.2 New Interface approach



2.3.2 New Interface approach



3. SIMULATION & RESULTS

3.1 Simulation methodology

1 **C-program compilation**

Compilation using only RISC-V standard extensions.

3.1 Simulation methodology

1	**C-program compilation**	Compilation using only RISC-V standard extensions.
2	**Simulation (w/o LiM)**	Simulation using LiM as a normal memory.

3.1 Simulation methodology

1	**C-program compilation**	Compilation using only RISC-V standard extensions.
2	**Simulation (w/o LiM)**	Simulation using LiM as a normal memory.
3	**LIM ISA extension**	Replacement of some instructions with LiM specific instructions.

3.1 Simulation methodology

1	**C-program compilation**	Compilation using only RISC-V standard extensions.
2	**Simulation (w/o LiM)**	Simulation using LiM as a normal memory.
3	**LIM ISA extension**	Replacement of some instructions with LiM specific instructions.
4	**Simulation (with LiM)**	Simulation exploiting the LiM capabilities.

3.1 Simulation methodology

1	**C-program compilation**	Compilation using only RISC-V standard extensions.
2	**Simulation (w/o LiM)**	Simulation using LiM as a normal memory.
3	**LIM ISA extension**	Replacement of some instructions with LiM specific instructions.
4	**Simulation (with LiM)**	Simulation exploiting the LiM capabilities.
5	**Comparison**	Comparison between the execution times.

3.2 Simulation results

Tested algorithms have been compared on the basis of their execution times (clock cycles).

Algorithms	W/O LiM	NEW IF LiM	SAME IF LiM
Custom programs			
Bitwise	129	24 (-82%)	33 (-74%)
Maximum-Minimum	177	71 (-59%)	74 (-58%)
Standard programs			
Bitmap search	119	113 (-5%)	116 (-2%)
Aes addround key	208	176 (-15%)	179 (-14%)
Transport problem (min cost)	1609	1345 (-16%)	1391 (-14%)

Conclusions & Future work

LiM integration results

- Execution time improved in tested programs.
- *New Interface project* guarantees always better results.
- *Same Interface project* still gives good results and flexibility.

Future researches

- More complex algorithms can be exploited in future works
- RISC-V compiler to support the new ISA extensions.

Thanks for your attention!