RISC-V based Programming Model of a Computational SRAM Vector Processing Unit

Maha Kooli, Henri-Pierre Charles, Thaddée Bricout, Jean-Philippe Noel, María Ramírez Corrales, Bastien Giraud
Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
maha.kooli@cea.fr

Von Neumann Bottleneck

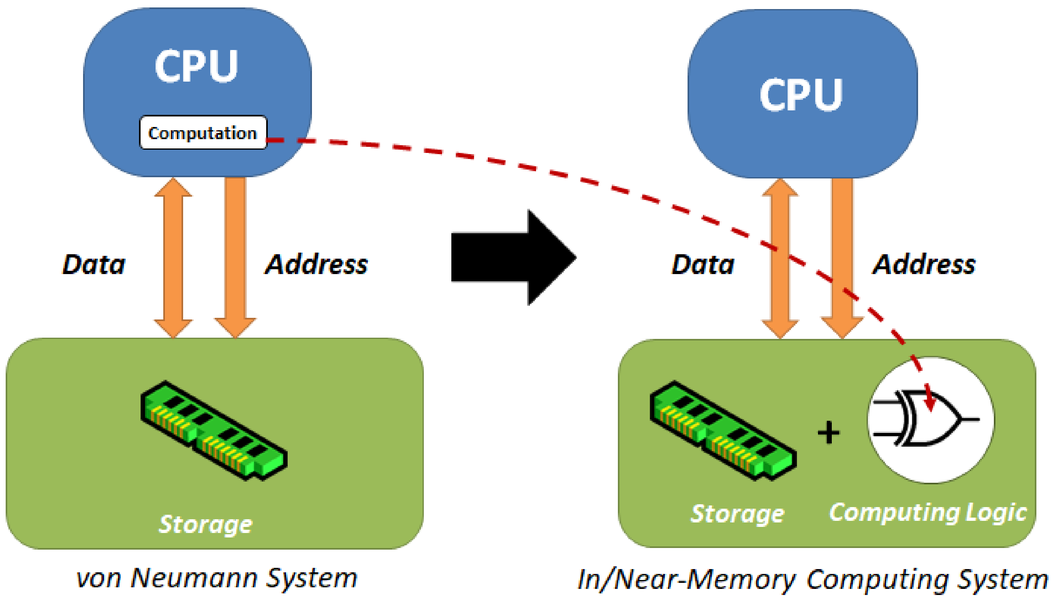


Fig 1. In/Near Memory Computing Architecture

- Integrating computation inside or near memories
- Reduce data transfer between processor and memory
- Increase the system performance and reduce energy consumption

Computational SRAM (C-SRAM)

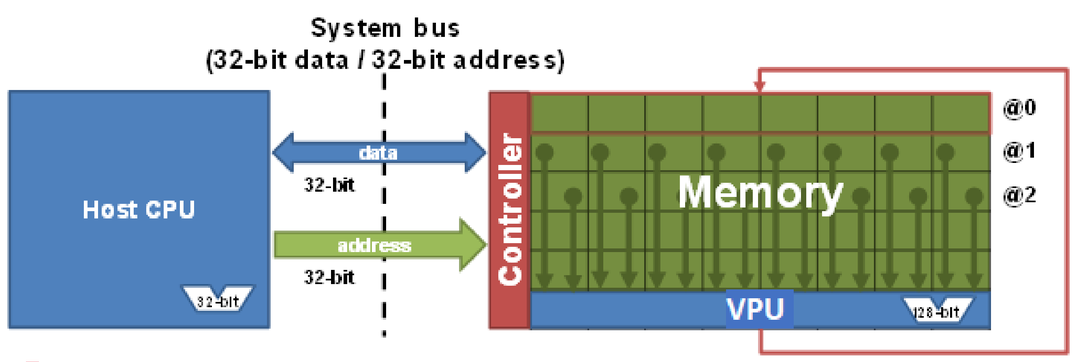


Fig 3. C-SRAM Architecture

- **Compute where data is located**
  - ➢ Vector computing
  - ➢ Low-latency memory
- **Rethink micro-architecture and compilation**
  - ➢ Specific instructions
  - ➢ Compile-time dynamic allocation
- **Two design methodologies**
  - ➢ Full custom solution
    - Change the memory bitcell
    - Specific in- and near-memory functions
  - ➢ Automated solution
    - Based on SRAM cut generated from SRAM compilers
    - Near memory functions only
    - Better Time-to-market

Category	Width (bits)	Mnemonic	Description
Memory	CSRAM Line	copy	Copy a line inti another
	8,16,32	copyeq, copygeq, copygt, copyleq, copylt, copyneq	Conditional Copy
	8,16,32	bcast	Broadcast 8,16 or 32-bit value to whole line
	32,64	hswap	Horizontal 32 or 64-bit word swap
Logical	8,16,32	slli, srli	Shift Left or Right Logical Immediate
	CSRAM Line	and, nand	Logical AND
	CSRAM Line	or, nor	Logical OR
	CSRAM Line	xor, xnor	Logical XOR
Arithmetic	8,16,32	add	Arithmetic Addition
	8,16,32	sub	Arithmetic Subtraction
	8,16,32	cmp	Comparison
	8	mullo, mulhi	Arithmetic 8-bit integer multiply
	8	maclo	Arithmetic 8-bit integer multiply-accumulate

Table 1. C-SRAM Instruction Set Architecture

C-SRAM Programming Model

A C-SRAM instruction is composed of 3 elements that must be encoded on the bus transaction:

- The opcode defines the C-SRAM operation
- The source addresses define the memory vector-lines that store the operands
- The destination address defines the memory vector-line that stores the result

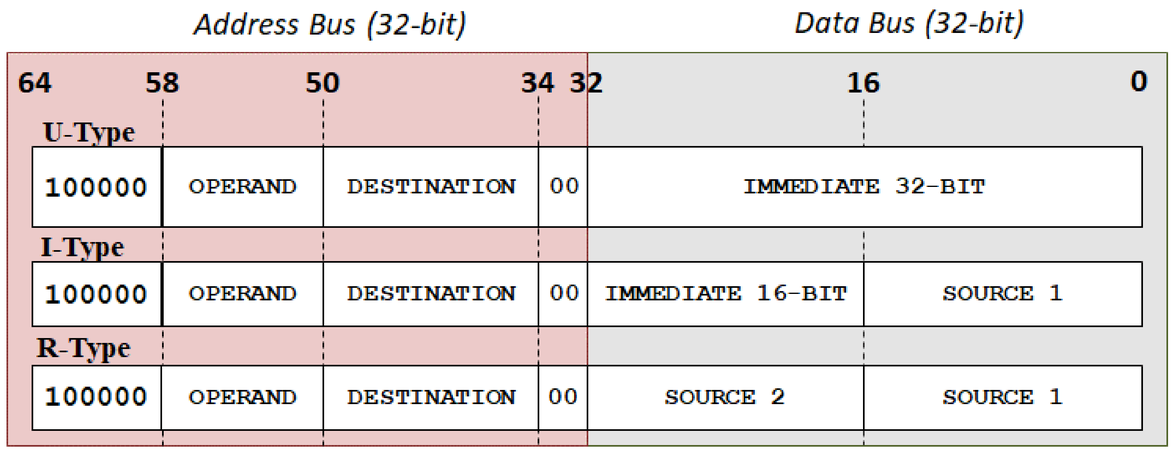
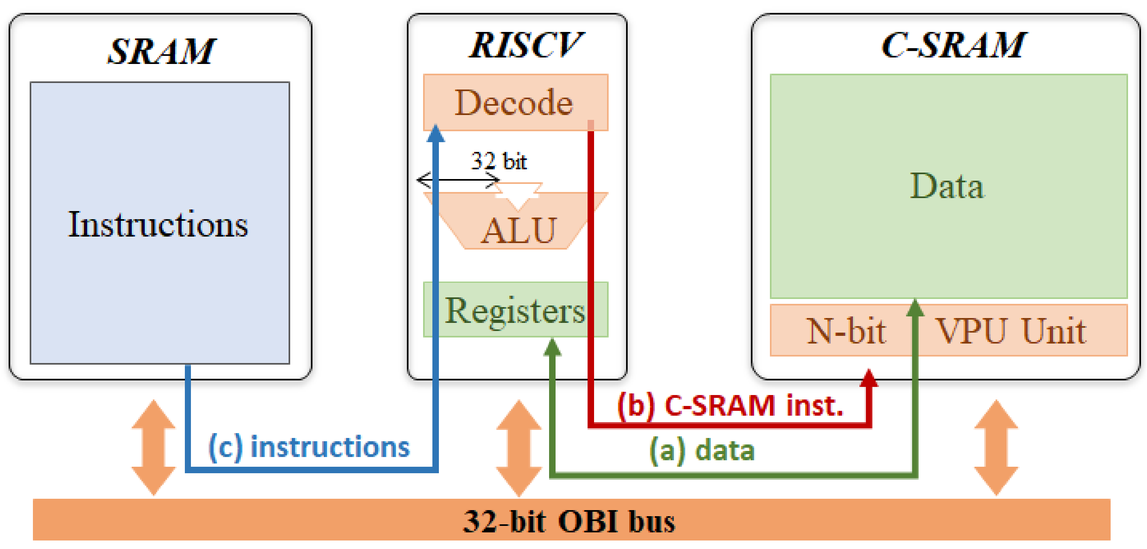


Fig 2. C-SRAM 64-bit Instruction Formats



(a) **Data Transfer:** *32-bit load/store conventional memory access*
(b) **C-SRAM instructions:** *64-bit **STORE** instruction encoding C-SRAM operation & operands*
(c) **Program Code:** *load/store, CPU scalar compute, C-SRAM vector compute, branch, etc*

Fig 4. C-SRAM System Communication Protocol

- The communication protocol that defines the interaction between the C-SRAM and the host processor, and manages the transfer of C-SRAM instructions, helps to integrate the C-SRAM circuit into an existing system without changing all its architecture
- The host processor runs the control part of the program, while the C-SRAM runs the main workload.
- We propose to use the '***STORE***' instruction to represent a C-SRAM instruction, where the C-SRAM operation and operands are encoded inside the registers of the '***STORE***' instruction,

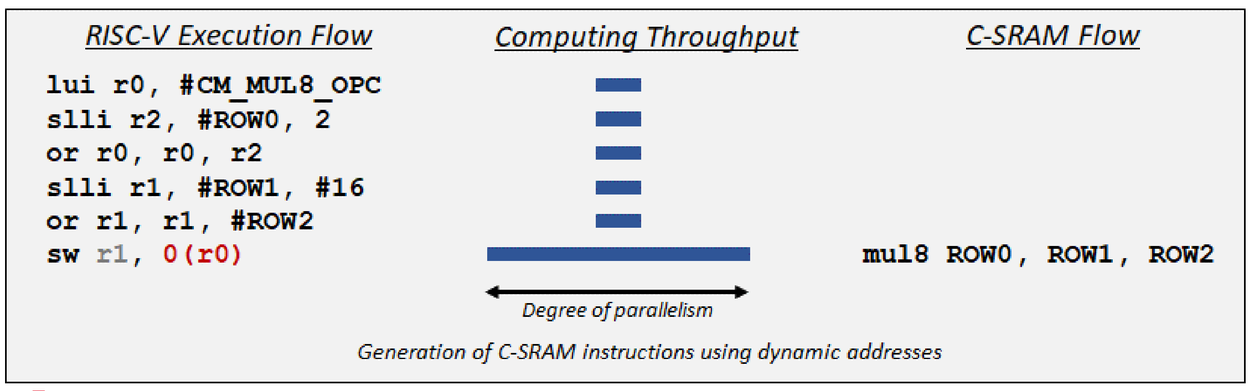


Fig 5. C-SRAM generated instruction

RISC-V based Software Tool Chain

To support the proposed programming model, we have developed a software stack:

- **A QEMU-based plugin emulator**
- **A RISC-V C cross-compiler and debugger**
- **A source-to-source compiler used to generate C-SRAM instructions**

1 K. Mambu et al. 2023 *Dedicated Instruction Set for Pattern-based Data Transfers: an Experimental Validation on Systems Containing In-Memory Computing Units*. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*
2 Mambu et al. 2022 *Towards Integration of a Dedicated Memory Controller and Its Instruction Set to Improve Performance of Systems Containing Computational SRAM*. *J. Low Power Electron. Appl.*, *12*, 18.
3 M. Kooli, et al. 2022. *Towards a Truly Integrated Vector Processing Unit for Memory-bound Applications Based on a Cost-competitive Computational SRAM Design Solution*. J. Emerg. Technol. Comput. Syst. 18, 2, Article 40, 26 pages.
4 K. Mambu et al. 2021. *Instruction Set Design Methodology for In-Memory Computing through QEMU-based System Emulator*. In IEEE ESWEEK-RSP.
5 J-P. Noel, et al. 2020. *A 35.6 TOPS/W/mm2 3-stage pipelined computational SRAM with adjustable form factor for highly data-centric applications*. IEEE ISSCL Letters 3, 286–289.
6 J-P. Noel et al. 2020. *Computational SRAM Design Automation using Pushed-Rule Bitcells for Energy-Efficient Vector Processing*. In IEEE DATE Conference. 1187–1192.
7 M. Kooli et al. 2018. *Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces*. In IEEE DATE Conference. 1634–1639.