Auto Micro-program Generation for Restricted ISA and Hardware Support

Yingming Ma

12/17/2025

Background

- Micro-program
  - The low-level control sequence stored in control memory that implements each ISA instruction. (e.g. add)
  - They define how each high-level instruction (e.g., ADD, LOAD) executes using low-level operations such as register transfers, ALU enables, and memory accesses
- Limited Resources within the chip
  - Need to select what arithmetic logic needed to include

Motivation

- Manual microprogramming is architectural specific and complex

Architectural Specific

1. Tradeoffs that can be considered when selecting an arithmetic logic unit:

    Area, Power, Throughput, Number of each unit… etc.

2. Different combinations of arithmetic logics have a different micro program

Complex

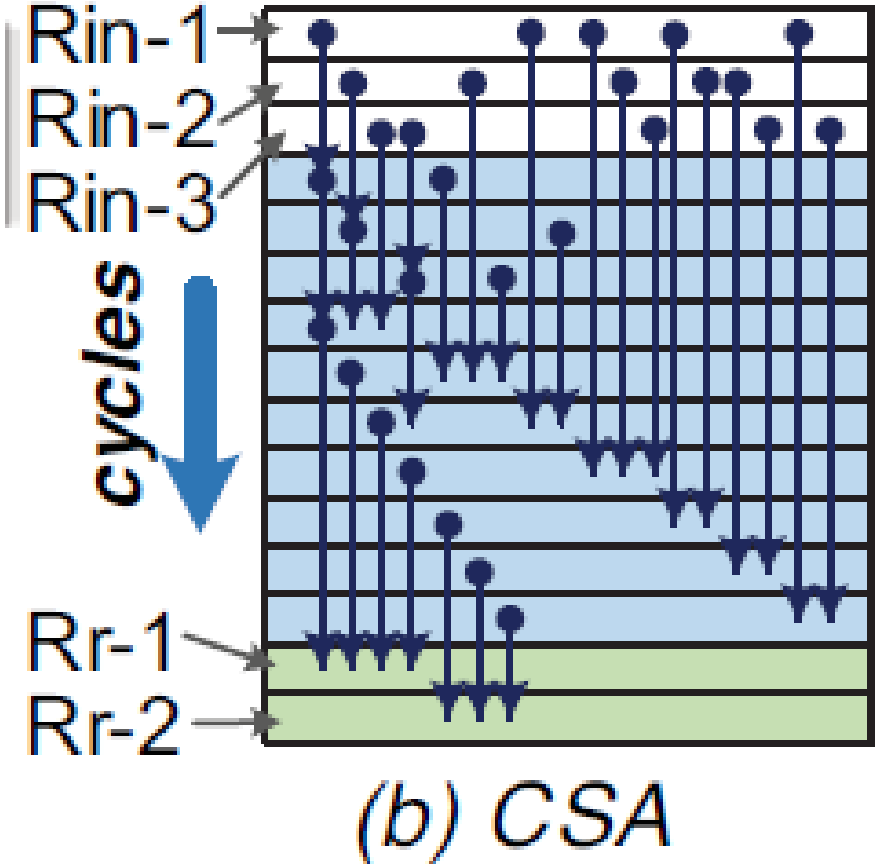
1.Resource Limited:

number of registers, movement of data… etc.

2.Simple Arithmetic Logic (e.g. only nor)

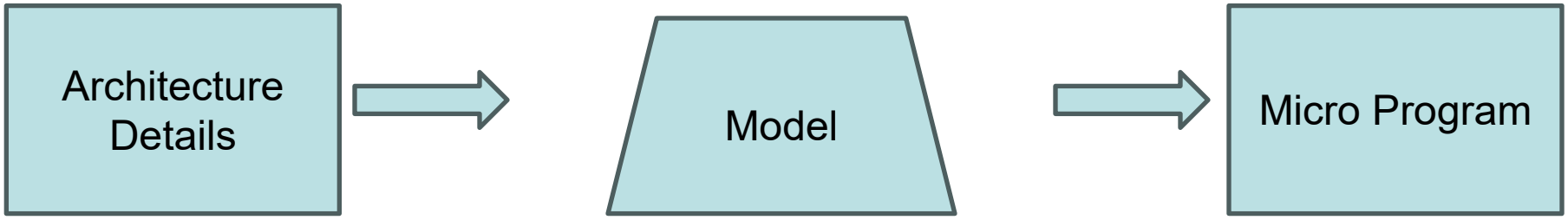
- Need to have complex algorithm
- More movements needed
- More registers needed

Addition Logic Using only Nor



Source: Li et al., “DRISA,” MICRO 2017 [1]

Target Task



Basic Input:

- Number of Register
- Register Mapping
- Logic Included
- Data Layout
- Target Micro Program

Advanced Input:

- Delay of Instruction
- Area Cost

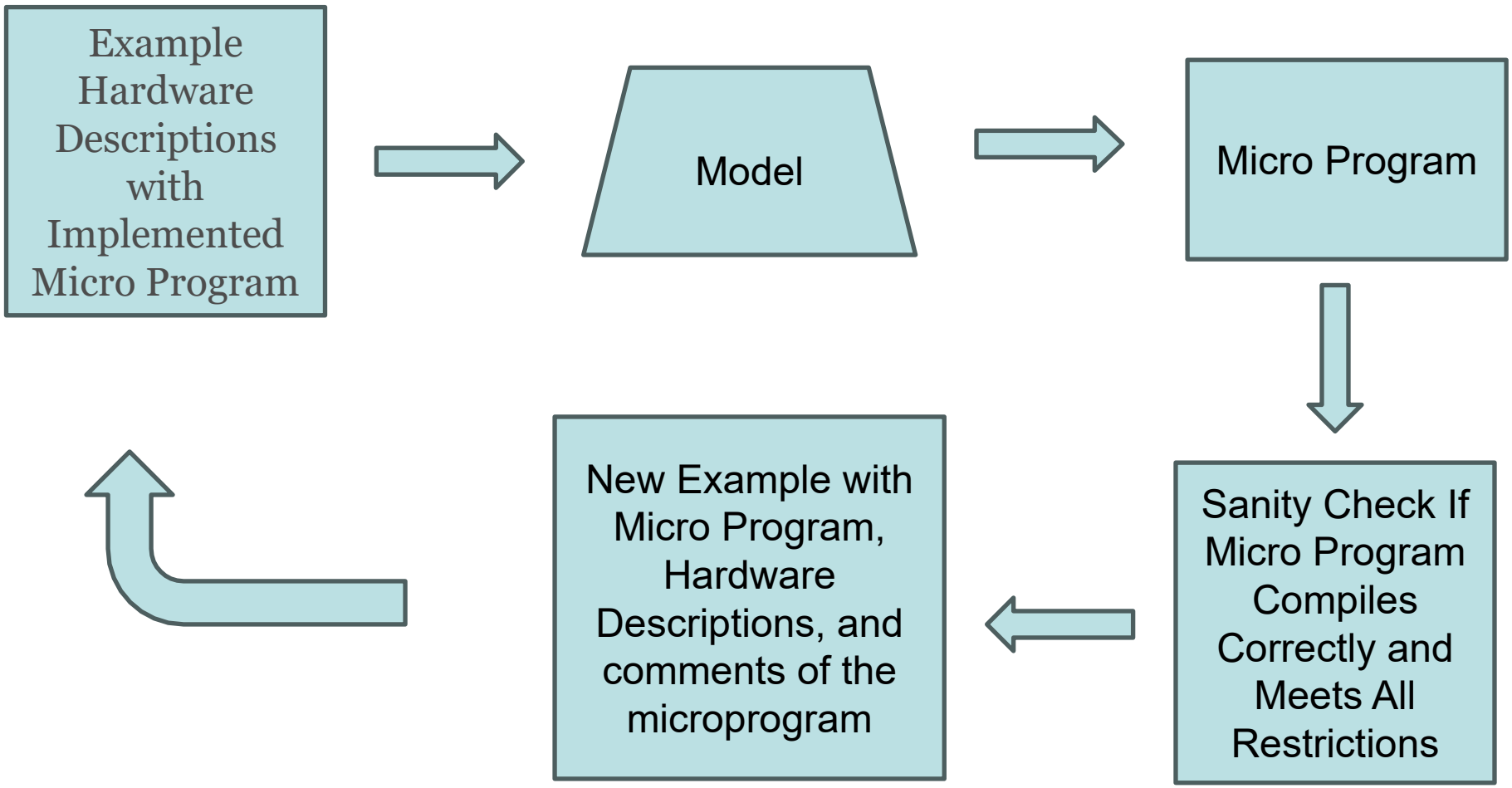
Basic Output:

- Working Micro Program Meets all hardware requirements

Advanced Output:

- Most Optimal Program within the current architecture
  - Smallest number of Register Used
  - Fastest

Proposed Solution



Input JSON File

- Examples
  - General Architecture Description
    - Architecture type
    - Register file numbers
    - Data movement restrictions
  - Instruction Set
  - Microprocessor Code step by step
    - Step number
    - Operation
    - arguments
  - Correctness and reasoning
- Query
  - Same as Examples
  - Correctness set to false
  - Only one query

Output

- Predefined in prompt outside of input JSON
- Part 1: Step by step fixed program
- Part 2: Reasoning Summary for researchers to debug


```
{
  "verifier_input": {
    "program": [
      {
        "step": 1,
        "instr": "ReadRowToSa(dram_row=ROW10)"
      },
      {
        "step": 2,
        "instr": "Swap(rr_index=1)"
      },
      {
        "step": 3,
        "instr": "ReadRowToSa(dram_row=ROW11)"
      },
      {
        "step": 4,
        "instr": "NOR()"
      },
      {
        "step": 5,
        "instr": "WriteSaToRow(dram_row=ROW12)"
      }
    ],
    "io": {
      "input_rows": [
        "ROW10",
        "ROW11"
      ],
      "output_row": "ROW12",
      "bitwidth": 32
    }
  },
  "reasoning_summary": [
    "Preserve A by swapping RR0 into RR1 before loading B.",
    "Load B into RR0, then NOR uses RR0 and RR1; result in RR0.",
    "Write the result from RR0 to OUT (ROW12)."
  ]
}
```

Verify

- Transform Output code JSON file to machine executable code
  - Currently a hand wrote conversion for limited amount of operation:
    - Read, Write, Swap, Nor
  - Potential to use existing simulator for more complex microprogram
- Calculate expected value based on the Query Description
  - Currently only Nor Implemented
- Compare the result and give Verification Result

Experimental Results & Analysis

Task Description: Given a reference implementation of a correct NOR-execution microprogram, the goal is to identify and correct an incomplete program by inferring and inserting the missing micro-operation required for correctness.

Model	Accuracy
Gpt 4o-mini	50%
Gpt5	100%
Gpt 4o	Format Issue

When algorithm gives wrong explanation, the code corrected will also be false.

Conclusion and Future Work

Modern large language models can debug simple microarchitectural designs without further training and demonstrate potential for generating simple programs under constrained instruction sets.

Future Work:

Complete the synthesis loop by incorporating verified output JSON files into the example database and automatically regenerating programs when verification fails.

Integrate the verification stage with an existing simulator to avoid re-implementing PIM execution rules and ensure consistency with established models.

Citation

1. Li, S., Niu, D., Malladi, K. T., Zheng, H., Brennan, B., & Xie, Y. (2017).

DRISA: A DRAM-based Reconfigurable In-Situ Accelerator.

In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (pp. 288-301). IEEE.

2. A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov,

B. Kozlovskii, F. J. R. Ruiz, A. Mehrabian, M. P. Kumar, A. See, S. Chaudhuri,

G. Holland, A. Davies, S. Nowozin, P. Kohli, and M. Balog,

“AlphaEvolve: A coding agent for scientific and algorithmic discovery,”

arXiv:2506.13131, 2025.