Processor Verification with Precise Exceptions and Speculative Execution

Jun Sawada
Department of Computer Sciences,
University of Texas at Austin
Austin, TX 78712, USA
E-mail: sawada@cs.utexas.edu

Warren A. Hunt, Jr.
IBM Austin Research Laboratory
11400 Burnet Road MS/9460,
Austin, TX 78758, USA
E-mail: whunt@austin.ibm.com

Overview of the Talk

- We discuss the framework of processor verification with exceptions and speculative executions.
  - Our target micro-processor design.
  - Our correctness criterion for a pipelined processor with exceptions and speculative execution.
  - Modeling of instruction behavior with the MAETT.
  - Verification of the correctness criterion from invariant conditions.

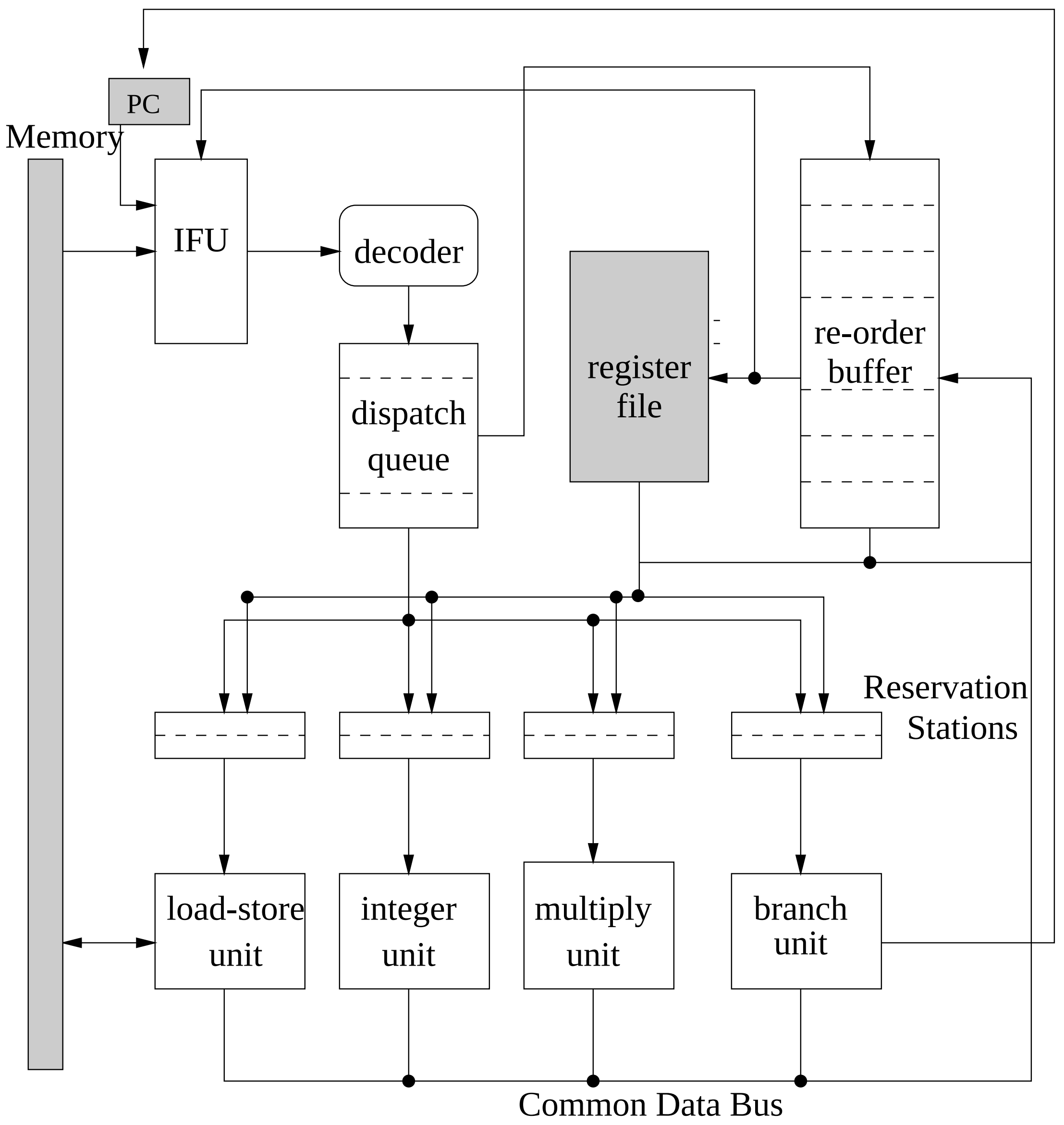
Features of Our Target Micro-architecture

The complexity of today's micro-processor design makes the verification problems more difficult. Examples of features included in our target design are:

- Out-of-order Execution of Instructions with Re-order Buffer.
- Multi-issue Capability.
- Renaming Registers.
- Speculative Execution.
- Precise Handling of Internal Exceptions and External Interrupts.
- Up to 15 instructions can be in the pipeline.
- Branch predication mechanism is modeled. When predication is wrong, up to 12 instruction can be speculatively executed.
- Memory delay is modeled with oracles.

c.f. In CAV '97, we have presented verification of a simpler pipelined processor without these features.

Our Target Pipelined Machine Design



Machine Design Specification

- We have written behavioral specification functions at the
  - Micro-architectural Level: MA-step(MA,MA-Inputs)
  - Instruction-Set Architecture Level: ISA-step(ISA,ISA-Inputs)

- This example processor has a small number of instructions. Each instruction is in a different instruction family.

Instruction	Description
ADD	Addition instruction, a typical integer unit instruction.
MULT	Multiplication instruction.
BR	Conditional branch.
LD	Load from memory.
LDIM	Load from memory using immediate address.
ST	Store to memory.
STIM	Store to memory using immediate address.
SYNC	Synchronize pipeline.
RFEH	Return from Exception. (privileged)
MFSR	Move from a special register. (privileged)
MTSR	Move to a special register. (privileged)

Our Correctness Criterion

– Consider an arbitrary execution sequence of the micro-architecture, starting from and ending up with flushed pipeline states.

– Suppose the initial flushed state is MA_0, and the execution takes n clock cycles and reaches a final flushed state MA_n.

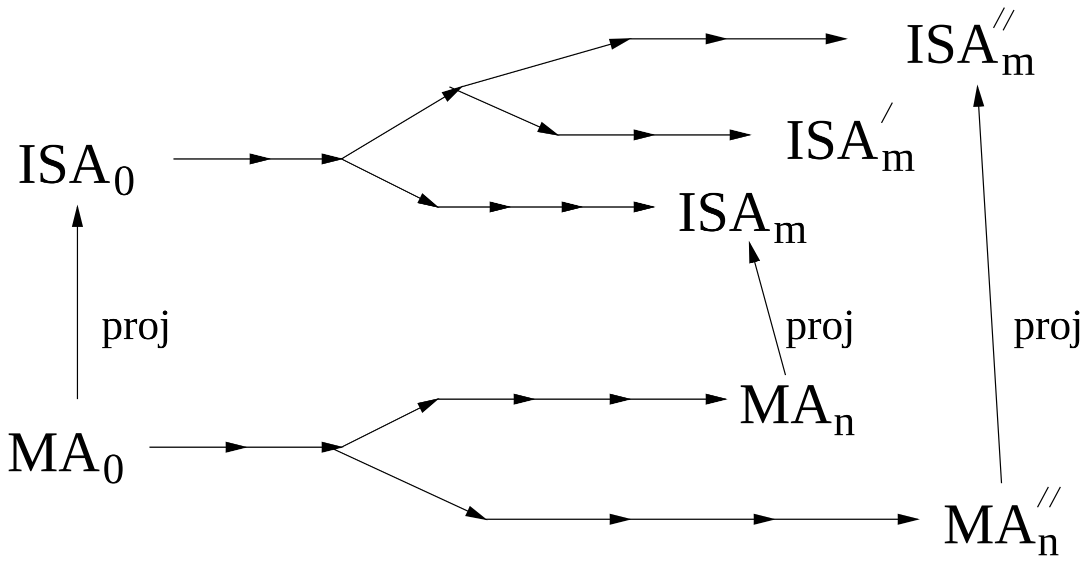
– If MA executes m instructions during the n clock cycles, then m ISA transitions from the initial state $\mathsf{ISA}_0 = \mathsf{proj}(\mathsf{MA}_0)$ result in the final ISA state $\mathsf{ISA}_m = \mathsf{proj}(\mathsf{MA}_n)$.

$$\begin{tikzcd} \mathrm{ISA}_0 \arrow[rrr, "\text{m ISA transitions}"] & & & \mathrm{ISA}_\mathrm{m} \\ \mathrm{MA}_0 \arrow[u, "\text{proj}"'] \arrow[rrr, "\text{n cycle MA transitions}"] & & & \mathrm{MA}_\mathrm{n} \arrow[u, "\text{proj}"'] \end{tikzcd}$$

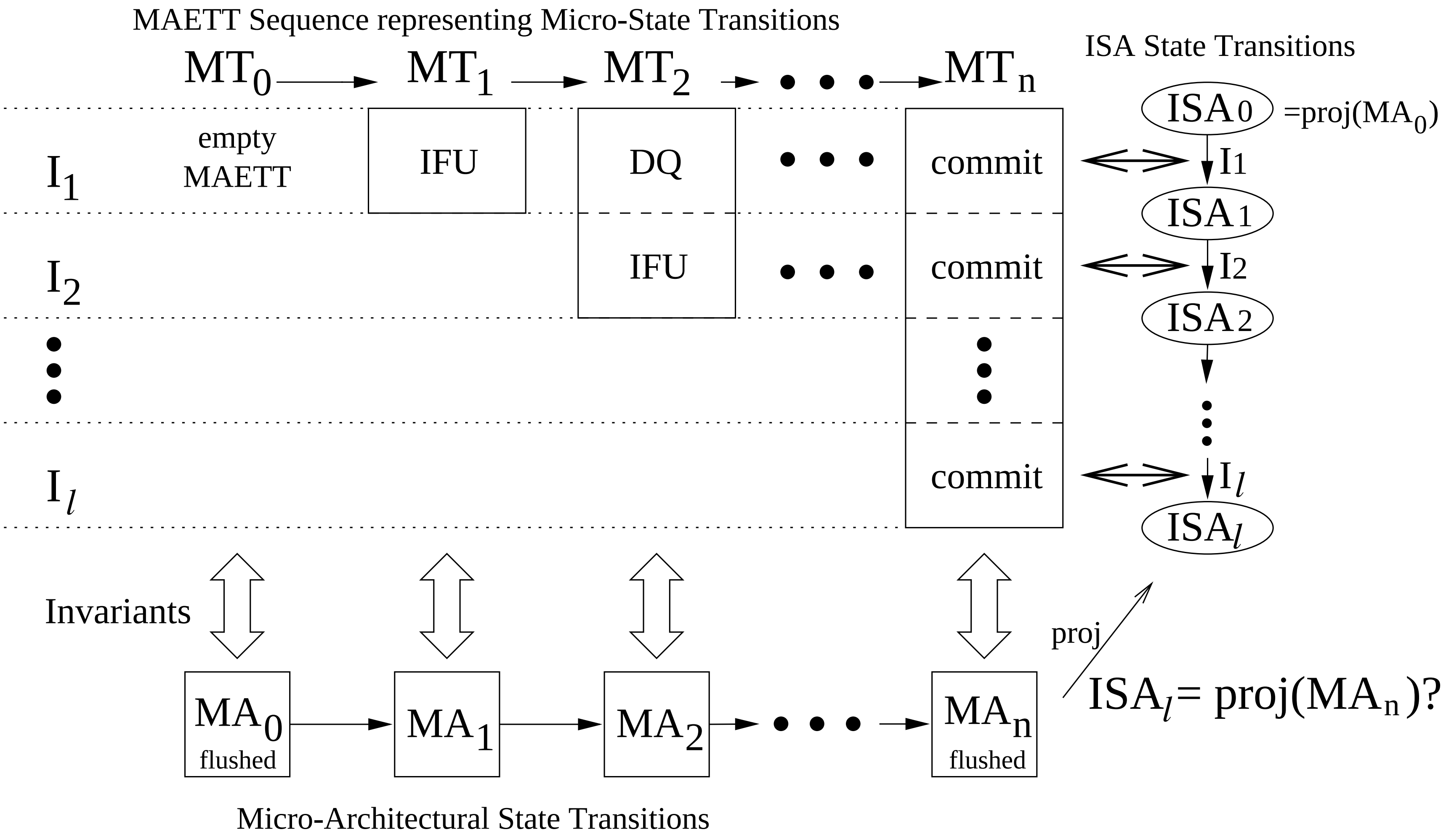
- This simple picture is valid even if the execution involves speculation and internal exceptions.

Correctness Criterion for External Interrupt

- Non-determinism is introduced by external interrupts.
- An external signal can interrupt any instruction in the pipeline.
- Correctness Criterion
  For an arbitrary transition sequence from a flushed MA state to another flushed state, there exists a corresponding ISA transition sequence from the initial state $\mathsf{ISA}_0 = proj(\mathsf{MA}_0)$ to the final state $\mathsf{ISA}_m = proj(\mathsf{MA}_n)$.



Where MAETT Lies in the Picture of Verification



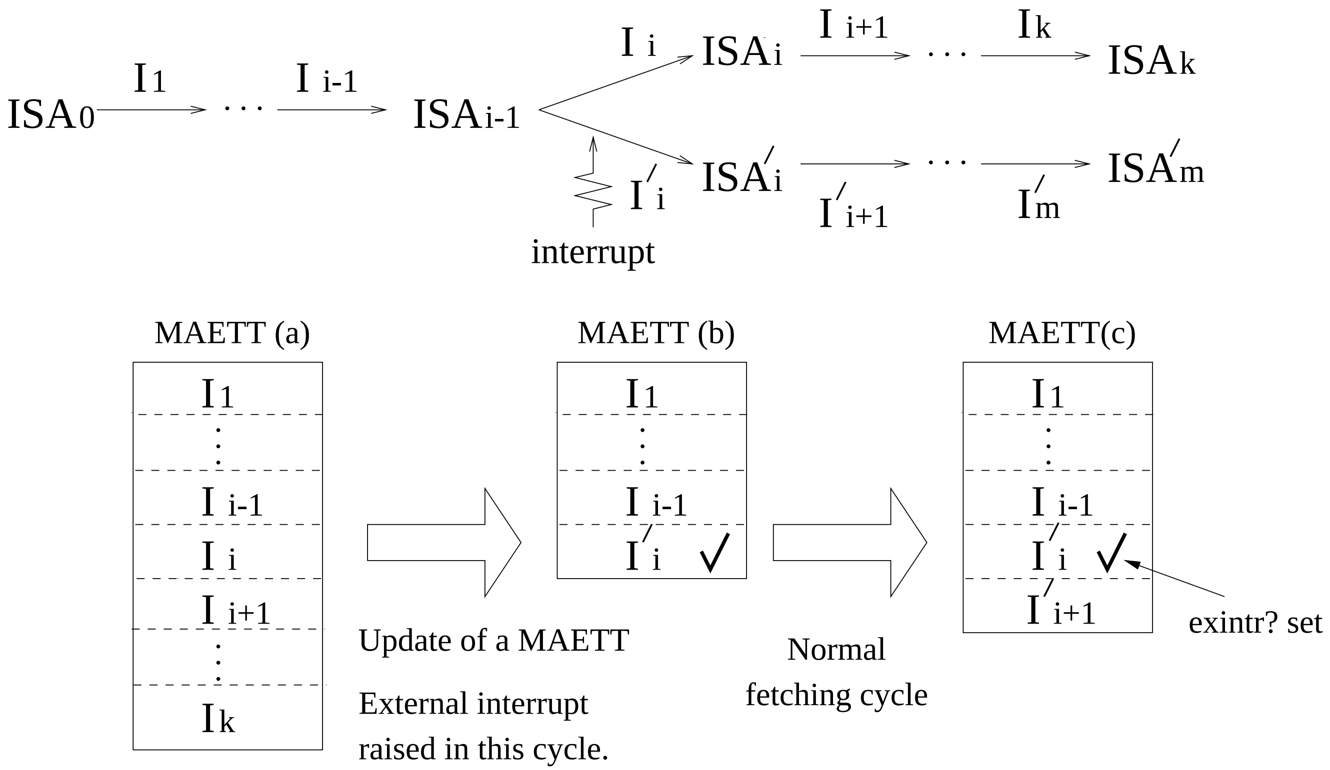
Micro-architecture Execution Trace Table

- The Micro-architectural Execution Trace Table(MAETT) is a table of instructions which are completed or partially executed.
- A MAETT is an abstraction of a micro-architectural state.
- Each entry of the table records the state of an instruction using a data-structure shown below.

Field name	Brief description
`ID`	Identity of I_i.
`word`	Instruction word.
`stg`	Current pipeline stage of I_i.
`robe`	Reorder buffer entry where I_i is stored.
`modify?`	Flag to show whether I_i is a modified instruction.
`speculative?`	Flag to show whether I_i is speculatively executed.
`br-predict?`	Outcome of branch prediction if I_i is a conditional branch.
`exintr?`	Flag to show whether I_i is interrupted.
`pre-ISA`	ISA_{i-1}, i.e., ISA state before executing I_i.
`post-ISA`	ISA_i, i.e., ISA state after executing I_i.

MAETT Update for Exceptions and Speculative Execution

- We defined the MAETT update function, MAETT-step(MT,MA,MA-Inputs), which abstracts the pipelined execution of instructions.
  - As each new instruction is fetched, a new entry is added.
  - As an instruction advances, the corresponding entry is updated.
  - If instructions are abandoned because of speculative execution or exceptions, the corresponding entries in MAETT are removed.



Invariant Conditions

- We have identified and defined various invariant conditions about our pipelined design and instruction streams.

`in-order-dispatch-commit-p` : Instructions dispatch and commit in order.

`in-order-ROB-p` : Re-order buffer works as a FIFO queue.

`no-stage-conflict` : Each pipeline stage contains at most one instruction.

`no-speculative-commit-p` : Speculatively executed instructions (after mis-prediction of a branch or exception-causing instruction) never commit.

⋮

- We define the conjunction of the invariant conditions as predicate $Inv(MT,MA)$ of a MA state MA and its MAETT representation MT.

Requirements on Inv

- We proved our correctness criterion provided that $Inv(MT,MA)$ satisfies several requirements.

Requirement 1 If $Inv(MT_k, MA_k)$ holds and MA_k is a flushed pipeline state, then every instruction I_i in MT_k is committed.

Requirement 2 If $Inv(MT_k, MA_k)$ holds and $MT_k = (I_1, \ldots, I_l)$, then

$$\neg \mathit{MAETT\text{-}speculative?}(MT_k) \Rightarrow \mathit{MA\text{-}pc}(MA_k) = \mathit{ISA\text{-}pc}(ISA_l).$$

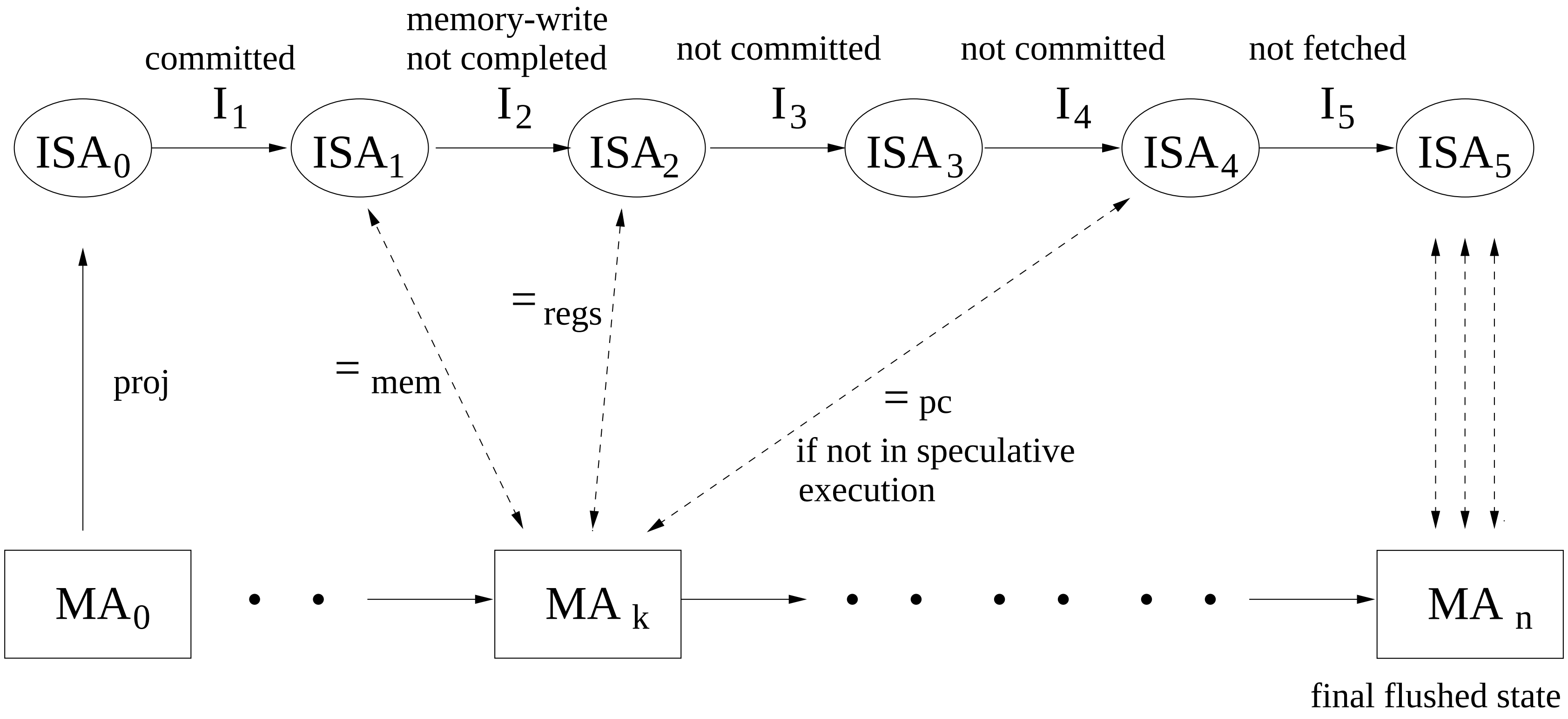
Requirement 3 If $Inv(MT_k, MT_k)$ is true and I_{i+1}, $i < l$, is the first uncommitted instruction in $MT_k = (I_1, \ldots, I_l)$, then

$$\mathit{MA\text{-}regs}(MA_k) = \mathit{ISA\text{-}regs}(ISA_i).$$

Requirement 4 If $Inv(MT_k, MA_k)$ is true and I_{i+1}, $i < l$, is the first memory store instruction whose memory write is not completed, then

$$\mathit{MA\text{-}mem}(MA_k) = \mathit{ISA\text{-}mem}(ISA_i).$$

Pictorial Description of Required Conditions



Requirements on Inv (contd.)

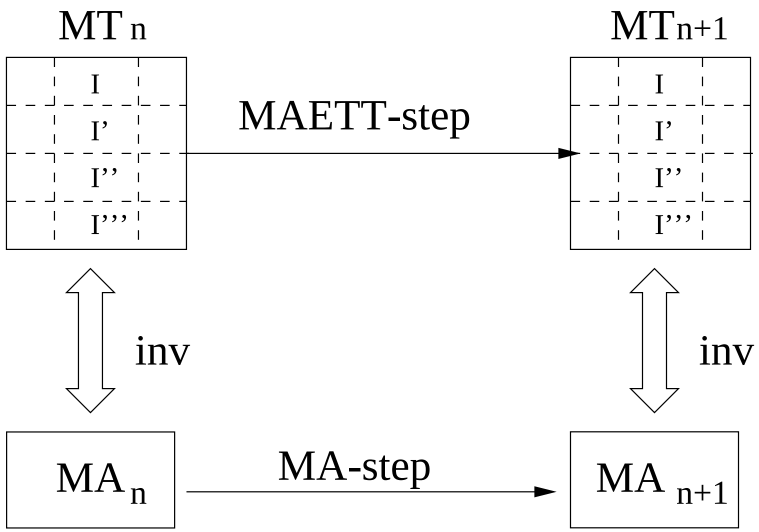
Requirement 5 For an arbitrary flushed initial state MA_0 and its MAETT MT_0, $Inv(MT_0, MA_0)$ holds.

Requirement 6 Suppose MA_{k+1} and MT_{k+1} are the next MA state and next MAETT, that is:

$$MA_{k+1} = MA\text{-}step(MA_k, Inputs_k),$$
$$MT_{k+1} = MAETT\text{-}step(MT_k, MA_k, Inputs_k).$$

Then

$$Inv(MT_k, MA_k) \Rightarrow Inv(MT_{k+1}, MA_{k+1}) \vee commit\text{-}self\text{-}modified\text{-}inst\text{-}p(MT_{k+1})\ .$$



Mechanical Proof of Correctness Criterion

- We have proved the correctness criterion under the assumption that our invariant condition Inv satisfy the six Requirements.
- In order to complete the verification, we need to verify the invariant condition Inv.
  - Requirements 1-5 have been established.
  - We are trying to verify that Inv satisfies Requirement 6; i.e. Inv is actually an invariant.

Conclusion and Future Work

- We have modeled the execution of instructions with an intermediate model called MAETT.
- We have verified our correctness criterion under the assumption that there exist invariant conditions that satisfy our Requirements.
- The verification of invariant conditions is ongoing. We have found several bugs during the verification of invariants.
- The liveness property can be proved separately.

Appendix

- Some people asked why Burch and Dill's diagram does not work with speculative execution and external interrupts.
  - Speculatively fetched instruction may not be always completed .
  - Any instruction in the pipeline can be externally interrupted.