Generic Interpreters and Microprocessor Verification

Phillip J. Windley

Department of Computer Science
University of Idaho

August, 1990

This work was sponsored under Boeing Contract NAS1-18586, Task Assignment No. 3, with NASA-Langley Research Center.

Outline

- Introduction
- Generic interpreters
- Microprocessor Verification
- Future Work

Microprocessor Verification

- VIPER, the first commercially available, "verified" microprocessor, has never been formally verified.

- The proof was not completed even though 2 years were spent on the verification.

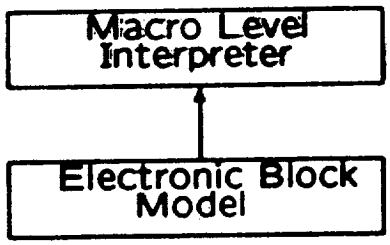
Microprocessor Verification
(continued)

- Our research is aimed at making the verification of large microprocessors tractable.

- *Our objective is to provide a framework in which a masters–level student can verify VIPER in 6 person–months.*

Determining Correctness

In VIPER (and most other microprocessors), the correctness theorem was shown by proving that the electronic block model implies the macro–level specification.



The Problem

(continued)

- Microprocessor verification is done through case analysis on the instructions in the macro level.

- The goal is to show that when the conditions for an instruction's selection are right, the electronic block model implies that it operates correctly.

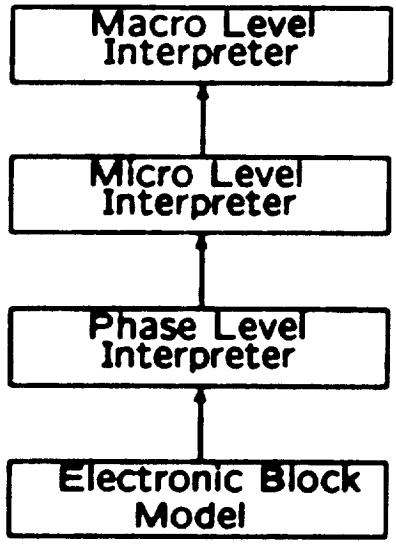
- A lemma that the EBM correctly implements each instruction can be used to prove the top–level correctness result.

The Problem

Unfortunately, the one–step method doesn't scale well because

- The number of cases gets large.
- The description of the electronic block model is very large.

Hierarchical Decomposition



- A microprocessor specification can be decomposed hierarchically.

- The abstract levels are represented explicitly.

Interpreters

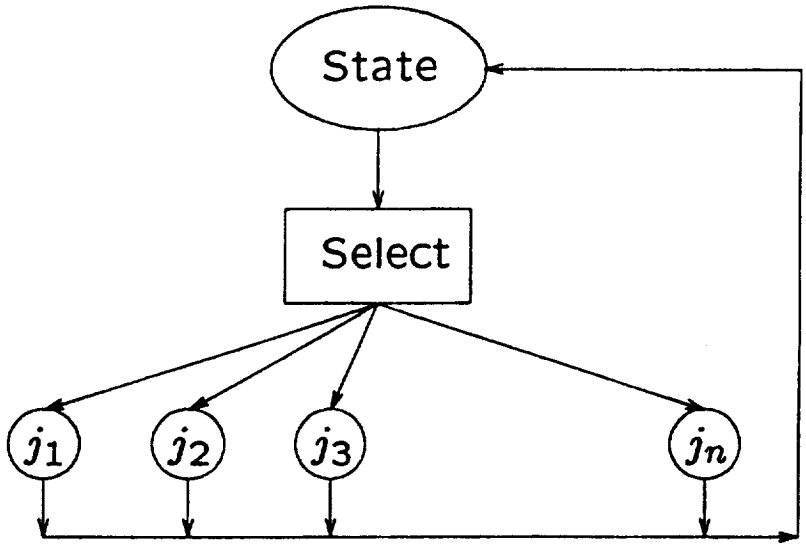
An abstract model of the different layers in the hierarchy provides a methodological approach to microprocessor verification.

- The model drives the specification.

- The model drives the verification.

Interpreters

(top level)



Specifying an Interpreter
(overview)

We specify an interpreter by:

- Choosing a n–tuple to represent the state, **S**.

- Defining a set of functions denoting individual interpreter instructions, **J**.

- Defining a next state function, **N**.

- Defining a predicate denoting the behavior of the interpreter, **I**.

Verifying an Interpreter
(overview)

We verify an interpreter, **I** with respect to its implementation **M** by showing

$$\mathbf{M} \Rightarrow \mathbf{I}.$$

To do this, we will show that every instruction in **J** can be correctly implemented by **M**:

$$\forall j \in \mathbf{J}.$$
$$\mathbf{M} \Rightarrow (\forall t: \text{time}.$$
$$\mathcal{C}(t) \Rightarrow s(t+n) = j(s(t)))$$

where $\mathcal{C}$ represents the conditions for instruction j's selection.

AVM-1

We have designed and are verifying a micro-computer with interrupts, supervisory modes and support for asynchronous memory.

- The datapath is loosely based on the AMD 2903 bit-sliced datapath.
- The instruction format is very simple.
- The control unit is microprogrammed.

AVM-1's Instruction Set
(subset)

Opcode	Mnemonic	Operation
000000	JMP	jump on 16 conditions
000001	CALL	call subroutine
000010	INT	user interrupt
000110	LD	load
000111	ST	store
010000	ADD	add (3-operands)
011011	SUBI	subtract immediate (2-operands)
011111	NOOP	no operation

- The architecture is load-store.
- The instruction set is RISC–like.
- There is a large register file.

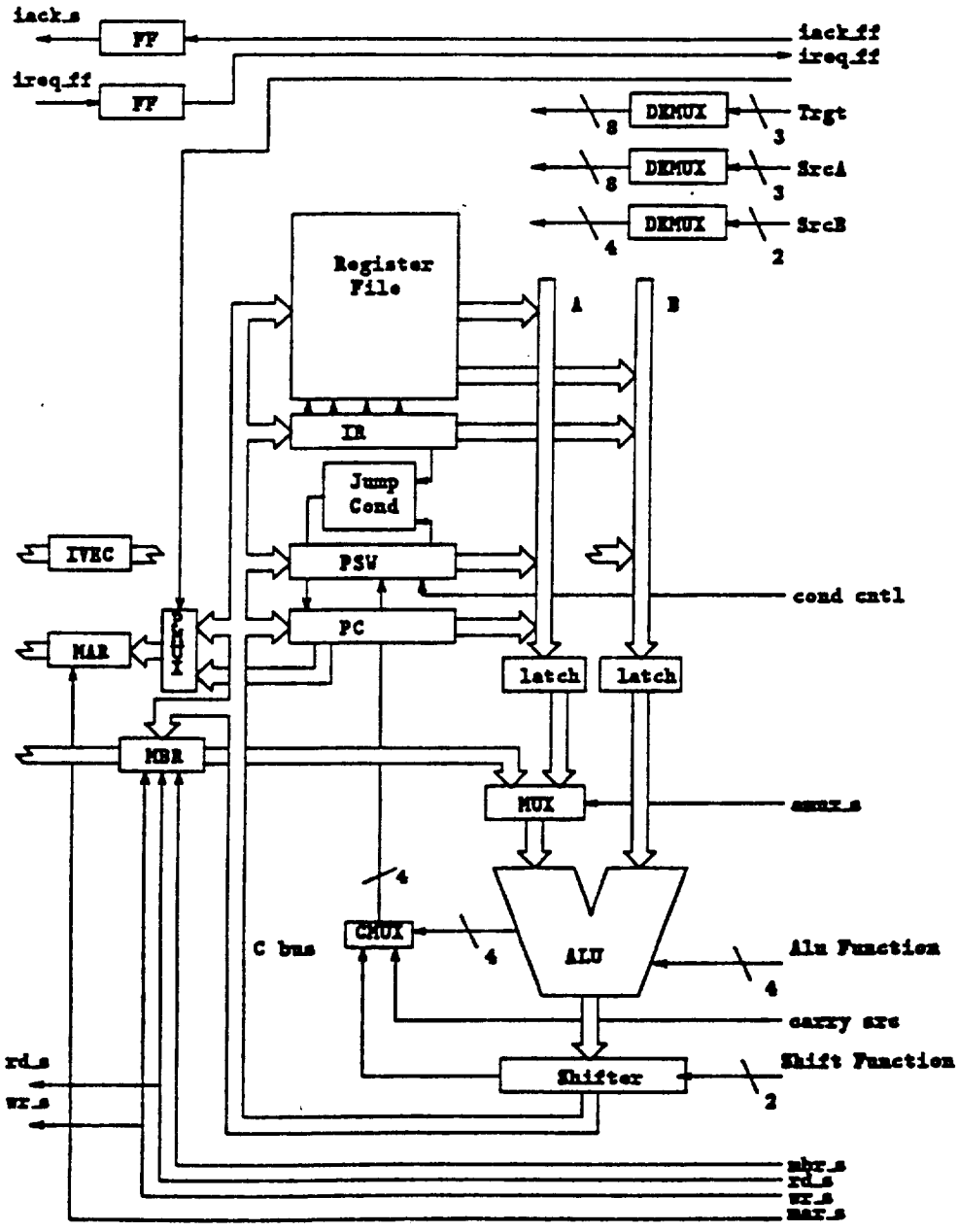


Figure 5.2: The *AVM-1* Datapath

The Phase–Level Specification

The n–tuple representing the state:

$$\begin{aligned}\mathbf{S}_{phase} = (&mir, mpc, reg,\\ &alatch, blatch, mar, mbr,\\ &clk, mem, urom, ireq, iack)\end{aligned}$$

The Phase–Level Specification

A typical function specifying an instruction's behavior from $\mathbf{J}_{phase}$:

```
⊢def phase_two rep (mir, mpc, reg, alatch, blatch,
                    mbr, mar, clk, mem, urom,
                    ireq, iack) =
    (mir, mpc, reg,
     EL (bt5_val (SrcA mir)) reg,
     EL (bt5_val (SrcB mir)) reg,
     mbr, mar, (T,F), mem, urom, ireq, Iack mir)
```

The Electronic Block Model

The electronic block model is not specified as an interpreter.

- EBM is a *structural* specification.

- The specification
  - is in terms of smaller blocks.
  - uses existential quantification to hide internal lines.

Objects

There are several abstract classes of objects that we will use to define and verify an abstract interpreter.

: $*state$	An object representing system state.
: $*key$	The identifying tokens for instructions.
: $time$	A stream of natural numbers.

We will prime class names to indicate that the objects are from the implementing level.

Operations

Operation	*Type*
inst_list	$: (*key \times (*state \rightarrow *state))list$
key	$: *key \rightarrow num$
select	$: *state \rightarrow *key$
cycles	$: *key \rightarrow num$
substate	$: *state' \rightarrow *state$
Impl	$: (time \rightarrow *state') \rightarrow bool$
clock	$: *state' \rightarrow *key'$
begin	$: *key'$

Interpreter Theory
(obligations)

The *instruction correctness lemma* is important in the generic interpreter verification.

Here is the generic version of that lemma for a *single* instruction:

$$\vdash_{def} \text{INST\_CORRECT } s'\ inst =$$
$$(\text{Impl } s') \Rightarrow$$
$$\forall t' : time'.$$
$$\text{let } s = (\lambda t.\ \text{substate}(s'\ t')) \text{ in}$$
$$\text{let } c = (\text{cycles}(\text{select}(s\ t'))) \text{ in}$$
$$(\text{select}(s\ t') = (\text{FST } inst)) \wedge$$
$$(\text{clock}(s'\ t') = \text{begin}) \Rightarrow$$
$$((\text{SND } inst)\ (s\ t') = (s(t' + c))) \wedge$$
$$(\text{clock}(s'(t' + c)) = \text{begin})$$

Interpreter Theory
(obligations)

Using the predicate INST_CORRECT, we can define the theory obligations:

1. The *instruction correctness lemma*:

   EVERY (INST_CORRECT s') inst_list

2. Every key selects an instruction:

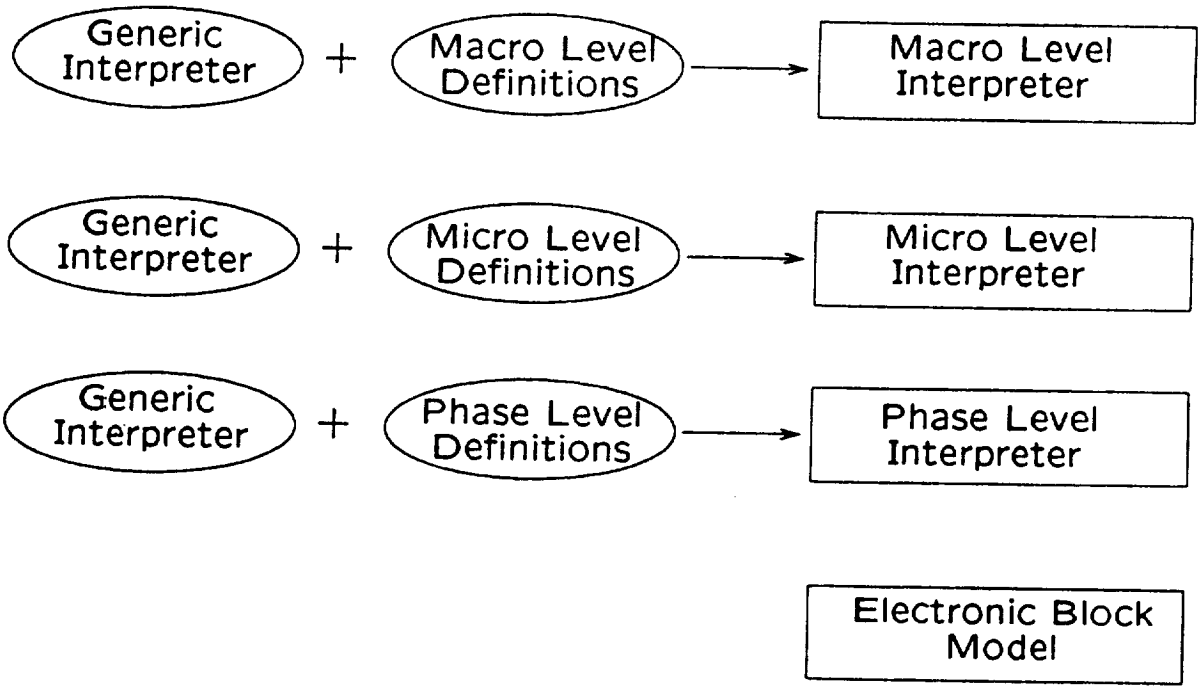
   $\forall k : *key.$ (key k) $<$ (LENGTH inst_list)

3. The instruction list is ordered correctly:

   $\forall k : *key.\ k =$ (FST (EL (key k) inst_list))

Generic Interpreters

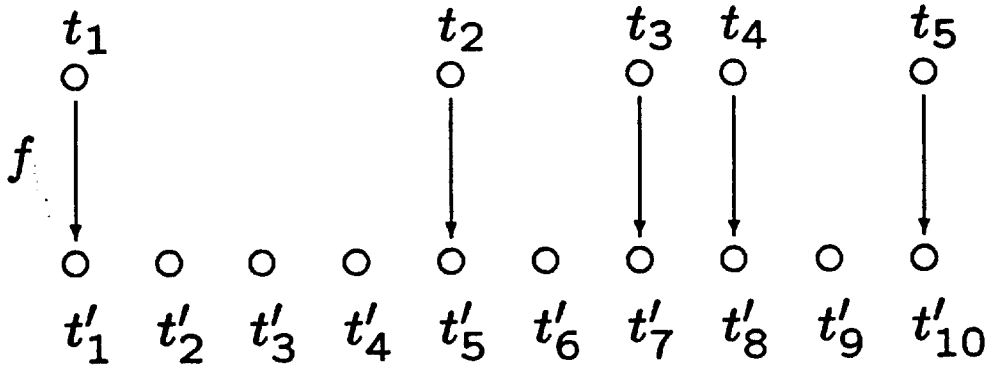
Instantiation



Interpreter Theory

(temporal abstraction)

We need to show a relationship between the state stream at the implementation level and the state stream at the top level.



The function f is a temporal abstraction function for streams.

Interpreter Theory
(definition)

An interpreter's behavior is specified as a predicate over a state stream.

$$\vdash_{def} \text{INTERP } s =$$
$$\forall t : time.$$
$$\text{let } n = (\text{key}(\text{select}(s\ t))) \text{ in}$$
$$s(t+1) = (\text{SND } (\text{EL } n \text{ inst\_list}))(s\ t)$$

Interpreter Theory
(correctness result)

Our goal is to verify an interpreter, **I** with respect to its implementation **M** by showing

$$\mathbf{M} \Rightarrow \mathbf{I}.$$

Here is the abstract result:

$$\vdash \text{Impl } s' \wedge (\text{clock}(s'\ 0) = \text{begin}) \Rightarrow \text{INTERP } (s \circ f)$$

where

s = $(\lambda t : time.\ \text{substate}(s'\ t))$ and

f = (time_abs (cycles $\circ$ select)s)

Instantiating a Theory

Instantiating the abstract interpreter theory requires:

- Defining the abstract constants.
- Proving the theory obligations.
- Running a tool in the formal theorem prover.

Definitions

We wish to instantiate the abstract interpreter theory for the phase-level. The electronic block model will be the implementing level.

Operation	*Instantiation*
inst_list	a list of instructions
key	bt2_val
select	GetPhaseClock
cycles	PhaseLevelCycles
substate	PhaseSubstate
Impl	EBM
clock	GetEBMClock
begin	EBM_Start

An Example

After proving the theory obligations, we can perform the instantiation.

```
let theorem_list =
    instantiate_abstract_theorems
        'gen_I'
        [Phase_I_EVERY_LEMMA;
         Phase_I_LENGTH_LEMMA;
         Phase_I_KEY_LEMMA]
        [
         "([(F,F),phase_one;
            (F,T),phase_two
            (T,F),phase_three
            (T,T),phase_four],
           bt2_val, GetPhaseClock,
           PhaseLevelCycles, PhaseSubstate,
           EBM, GetEBMClock, EBM_Start)";
         "(λ t:time. (mir t, mpc t, reg_list t,
                      alatch t, blatch t,
                      mbr_reg t, mar_reg t,
                      clk t, mem t, urom))"
        ]
       'PHASE';;
```


C6.

The Electronic Block Model

```
⊢ EBM rep (λ t.  (mir t, mpc t, reg t, alatch t, blatch t,
                  mbr t, mar t, clk t, mem t, urom,
                  ireq t, iack t)) =
        ∃ opc ie_s sm_s iack_s
          amux_s alu_s sh_s mbr_s mar_s rd_s wr_s
          cselect bselect aselect
          neg_f zero_f (float:time->bool).
        DATAPATH rep amux_s alu_s sh_s mbr_s mar_s rd_s wr_s
                 cselect bselect aselect neg_f zero_f float
                 float ireq iack_s iack opc ie_s sm_s
                 clk mem reg alatch blatch mar_reg
                 mbr_reg reset_e ireq_e ∧
       CONTROL_UNIT rep mpc mir clk amux_s alu_s sh_s mbr_s
                 mar_s rd_s wr_s cselect bselect aselect neg_f
                 zero_f ireq iack_s opc ie_s sm_s urom
                 reset_e ireq_e
```

Fully expanded, the electronic block model specification fills about six pages.

Future Work

- New architectural features.
- Composing verified blocks.
- Verifying operating systems.
- Gate–level verification.
- Byte-code interpreter verification.
- Other classes of computer systems.

An Example
(continued)

After some minor manipulation, the final result becomes:

```
⊢ EBM
    (λ t.
      (mir t,mpc t,reg_list t,alatch t,blatch t,
       mbr_reg t,mar_reg t, clk t,mem t,urom)) ==>
    Phase_I
    (λ t.
      (mir t,mpc t,reg_list t,alatch t,blatch t,
       mbr_reg t,mar_reg t, clk t,mem t,urom))
```

Conclusions

The generic proof

- Cleared away all the irrelevant detail.
- Formalized the notion of interpreter proofs which has been used in several microprocessor verifications.
- Provided a structure for future microprocessor verifications.