A Compiler-Assisted On-Chip Assigned-Signature Control Flow Checking

Xiaobin Li Jean-Luc Gaudiot

Department of Electrical Engineering and Computer Science
University of California, Irvine

The Ninth Asia-Pacific Computer System Architecture Conference
(ACSAC'04)
Beijing, China
September 9, 2004

PASCAL
University of California, Irvine

Reliable Processor Design Is a Challenge

1. CMOS built-in reliability
   - 1.1 Time-dependent dielectrical breakdown
   - 1.2 Hot-carrier-induced degradation
2. Radiation-induced transient faults
   - 2.1 Alpha particles
   - 2.2 Cosmic rays
3. Design faults

PASCAL
University of California, Irvine

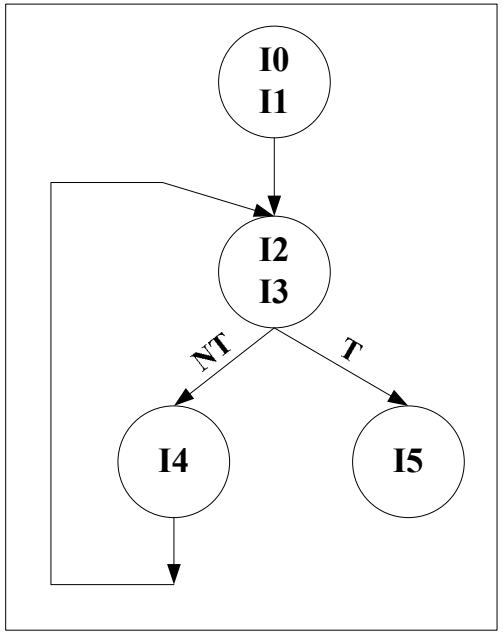
Outline

- Control flow errors
- Assigned-signature control flow checking algorithm
- Hardware enhancement
- Conclusion

PASCAL
University of California, Irvine

Control Flow Graph (CFG)

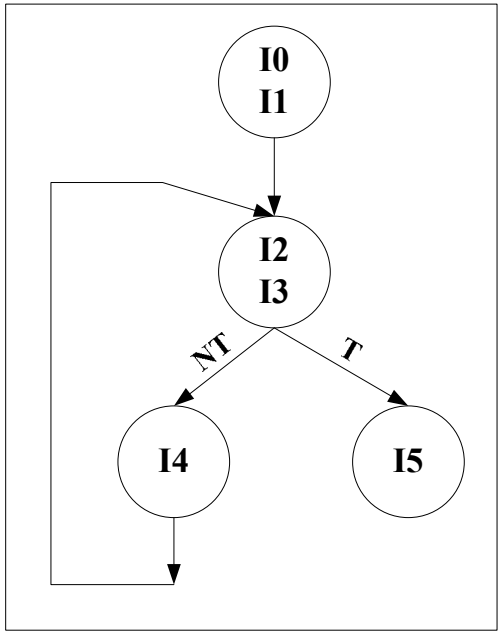
```
(I0)  load R1 = 5
(I1)  load R2 = -1
(I2)  R1 = R1 + R2
(I3)  if (R1 = = 0) goto I5
(I4)  goto I2
(I5)  halt
```



1. Node: *aka* basic block, a sequence of instructions with no branch-in except for the entry point and no branch-out except for the exit point.
2. Directed edge: representing jumps in the program control flow.

Control Flow Graph (CFG)

```
(I0)  load R1 = 5
(I1)  load R2 = -1
(I2)  R1 = R1 + R2
(I3)  if (R1 = = 0) goto I5
(I4)  goto I2
(I5)  halt
```



1. Built by compilers for code optimization;
2. Express the sequencing information among nodes:
   - 2.1 Node (I2+I3) should be after node (I0+I1);
   - 2.2 Only if R1 is zero (branch taken case), node (I5) can be executed;

Control Flow Errors

- Those causes a processor to violate the correct sequencing of instructions;
- Failure of instruction caches, program counters, branch units, etc.;
- Account for between 33% and 77% of all run-time errors.

PASCAL
University of California, Irvine

Assigned-Signature Control Flow Checking Algorithm

(without considering MBI and ITE nodes)

1. **Compile time:**
   - 1.1 Take a CFG from the compiler;
   - 1.2 Assign unique state code $D(i)$: e.g. numbering nodes in sequence;
   - 1.3 Compute reference signature:

     $$S(i) = D(i) \oplus D(pred(V_i));$$

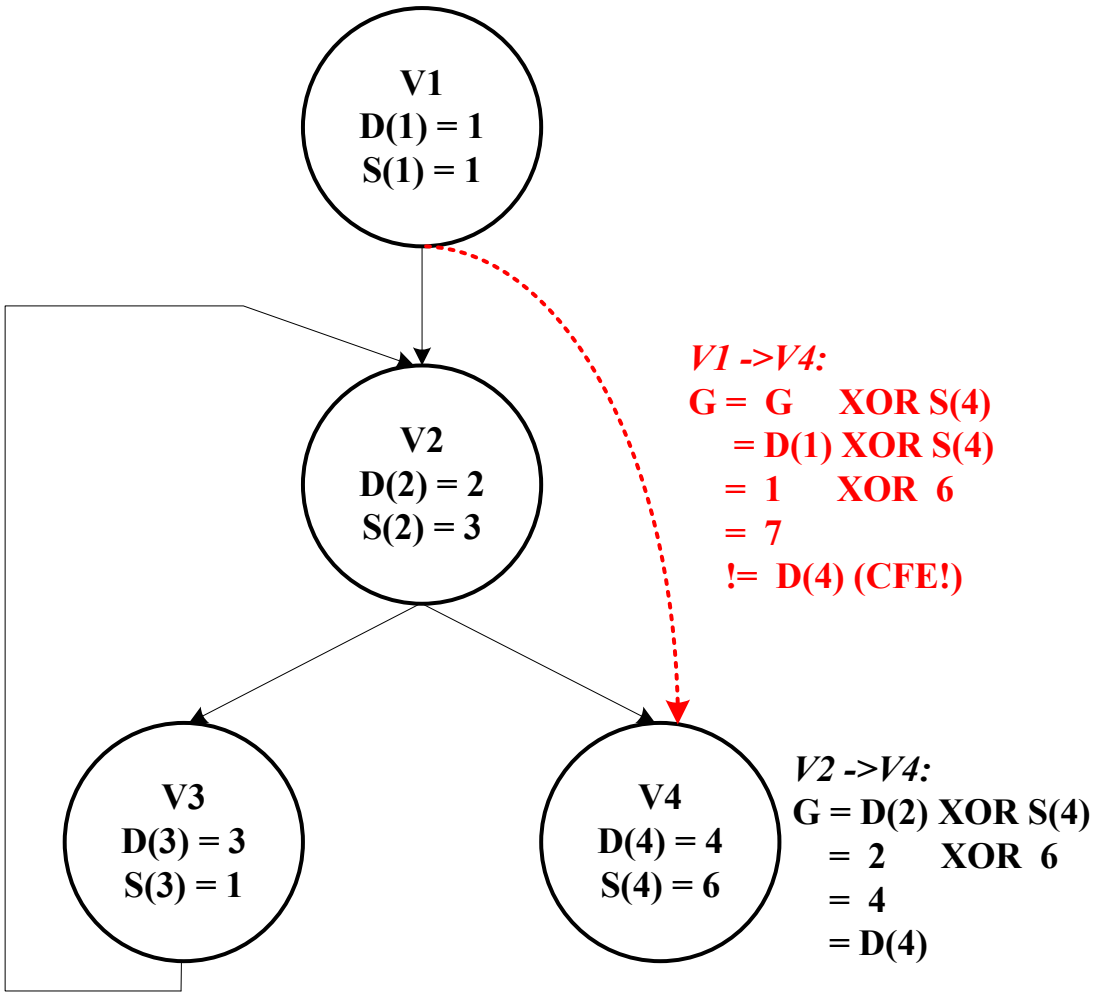
     $pred(V_i)$ — immediate predecessor of node V_i.
2. **Run time:**
   - 2.1 Update run time signature:

     $$G = G \oplus S(i);$$
   - 2.2 Check control flow errors (fault-free condition):

     $$G \oplus D(i) = 0; \text{ i.e., } G = D(i).$$

PASCAL
University of California, Irvine

Example of the Control Flow Checking Algorithm



Multiple-Branch-In (MBI) Nodes

- A node with multiple legal branch-in paths;
- Example: V_2 with "$V_1 \rightarrow V_2$" and "$V_3 \rightarrow V_2$";
- Multiple immediate predecessors:

$$\mathbb{S} = \{V_k | V_k \text{ is an immediate predecessor of MBI}\}$$

PASCAL
University of California, Irvine

Multiple-Branch-In (MBI) Nodes

- A node with multiple legal branch-in paths;
- Example: V_2 with "$V_1 \rightarrow V_2$" and "$V_3 \rightarrow V_2$";
- Multiple immediate predecessors:

$$\mathbb{S} = \{V_k | V_k \text{ is an immediate predecessor of MBI}\}$$

☞ Which one is used for calculating the reference signature of the MBI node: $S(i) = D(i) \oplus D(pred(V_i))$?
☞ How to check those "multiple legal branch-in paths?"

PASCAL
University of California, Irvine

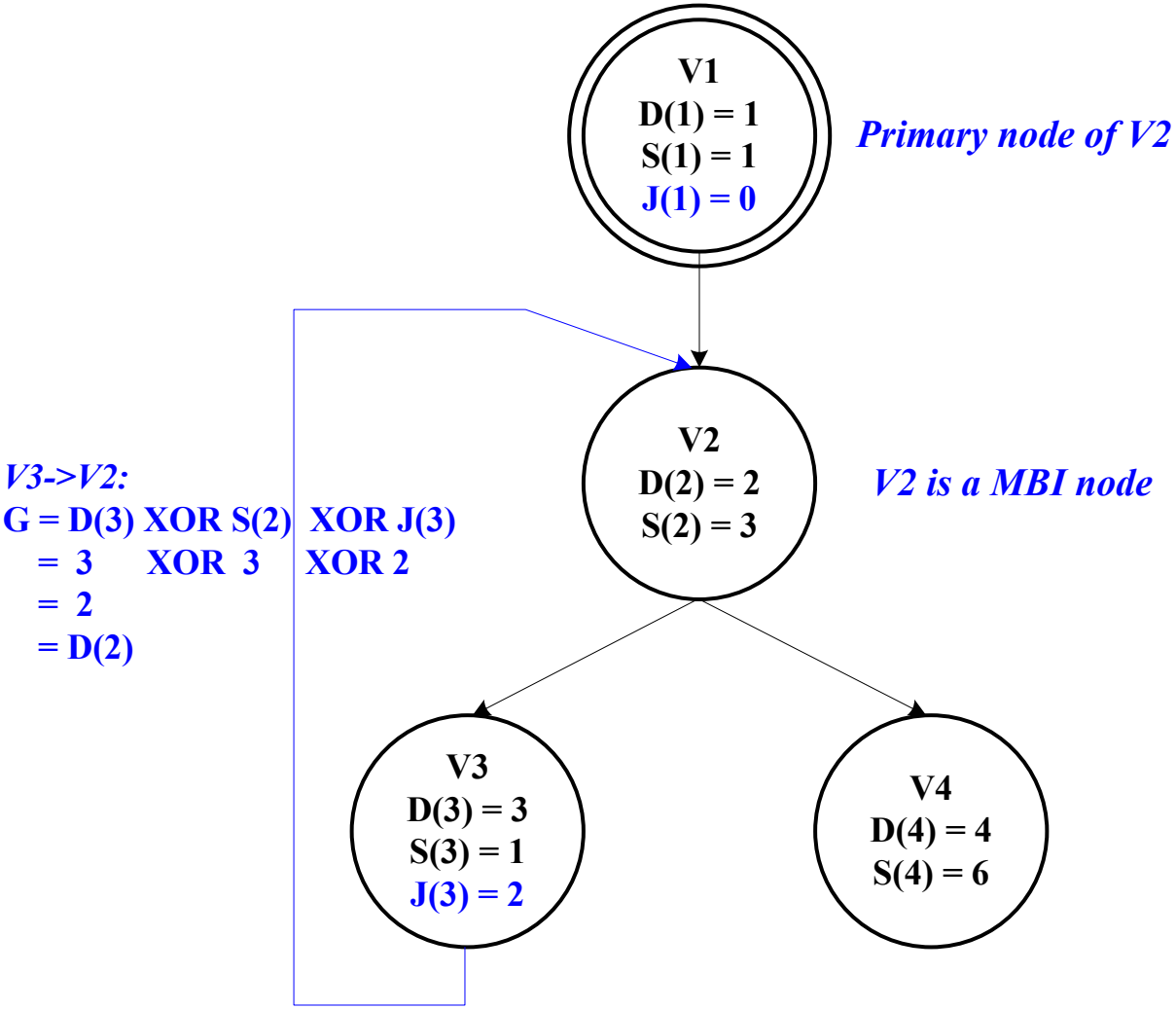
Primary Node and Justifying Signature

Handling MBI Nodes

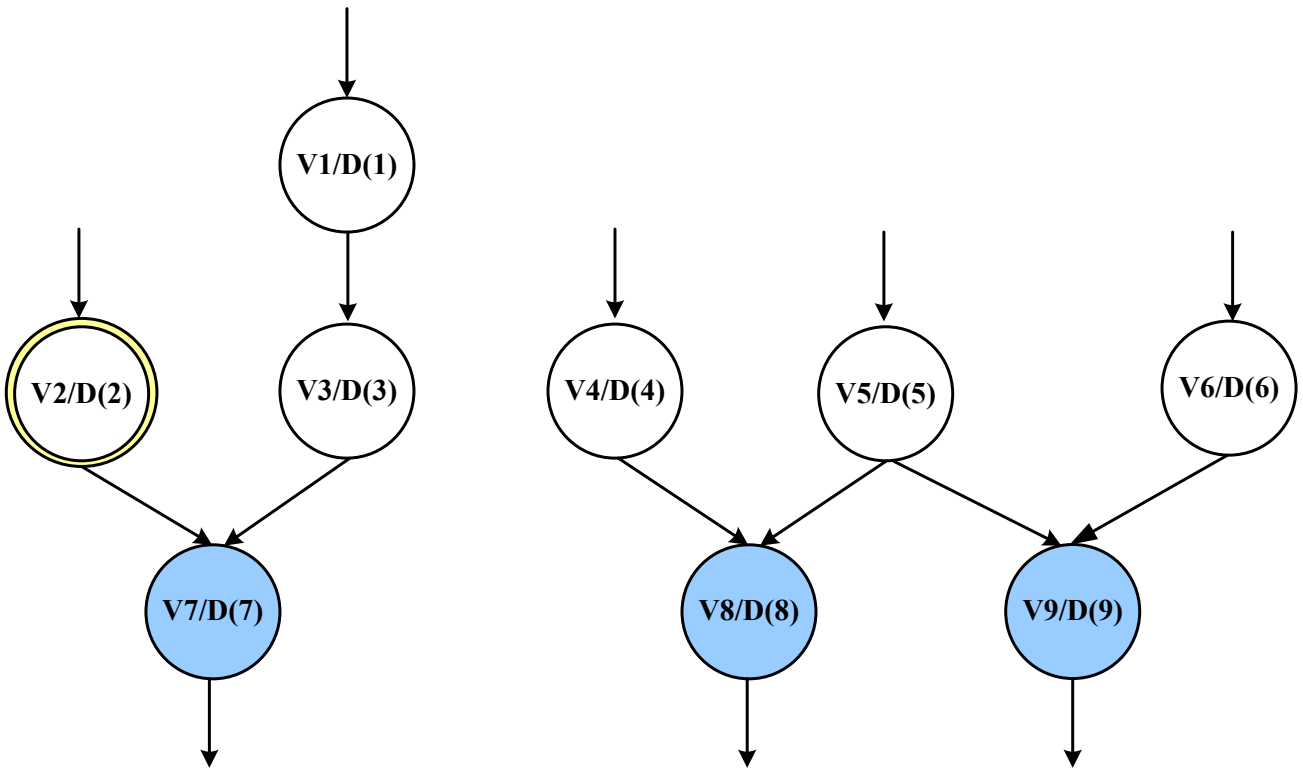
1. **Compile time:**
   - 1.1 Select a node from $\mathbb{S}$ as the MBI node primary node: V_j;
   - 1.2 Compute reference signature of the MBI node:
   $$S(MBI) = D(MBI) \oplus D(j);$$
   - 1.3 Associate justifying signature to every node $V_k \in \mathbb{S}$:
   $$J(k) = D(k) \oplus D(j);$$
2. **Run time:**
   - 2.1 Update run time signature when $V_k \rightarrow MBI$:
   $$G = G \oplus S(MBI) \oplus J(k);$$
   - 2.2 Check fault-free condition:
   $$G = D(MBI).$$

PASCAL
University of California, Irvine

Example of the MBI Node Handling



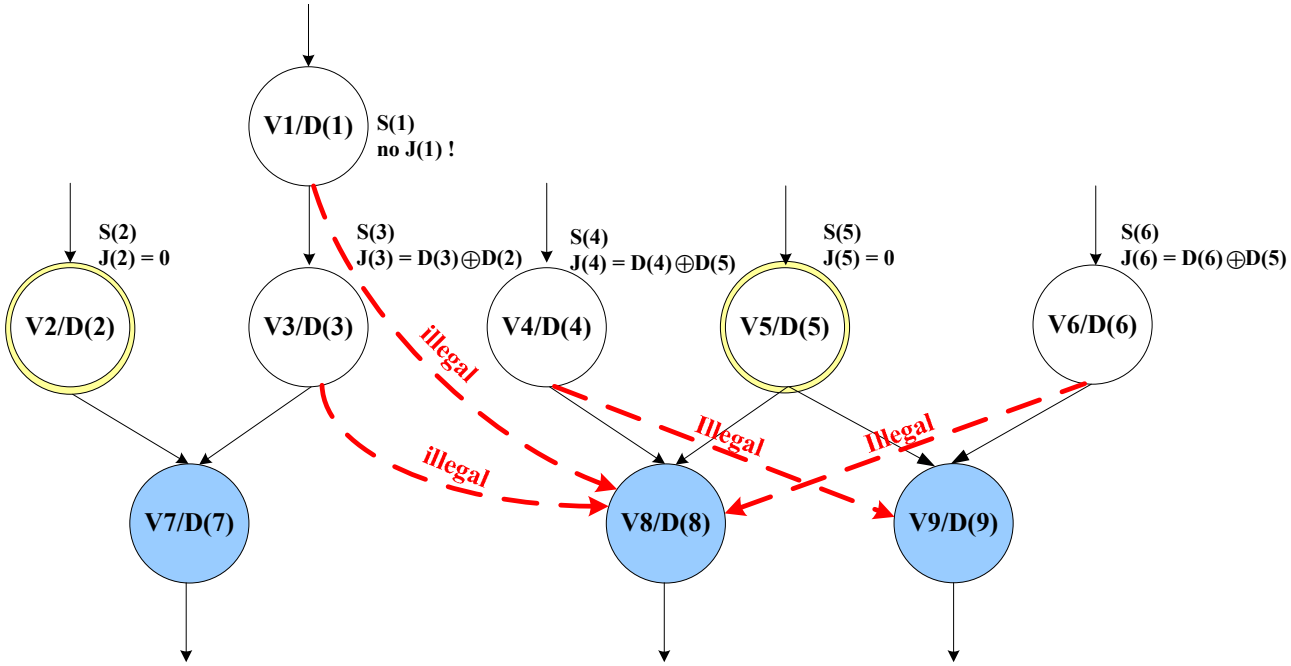
Primary Node Selection vs. Fault Detection Coverage



☞ Could V_5 be the primary node of both V_8 and V_9?

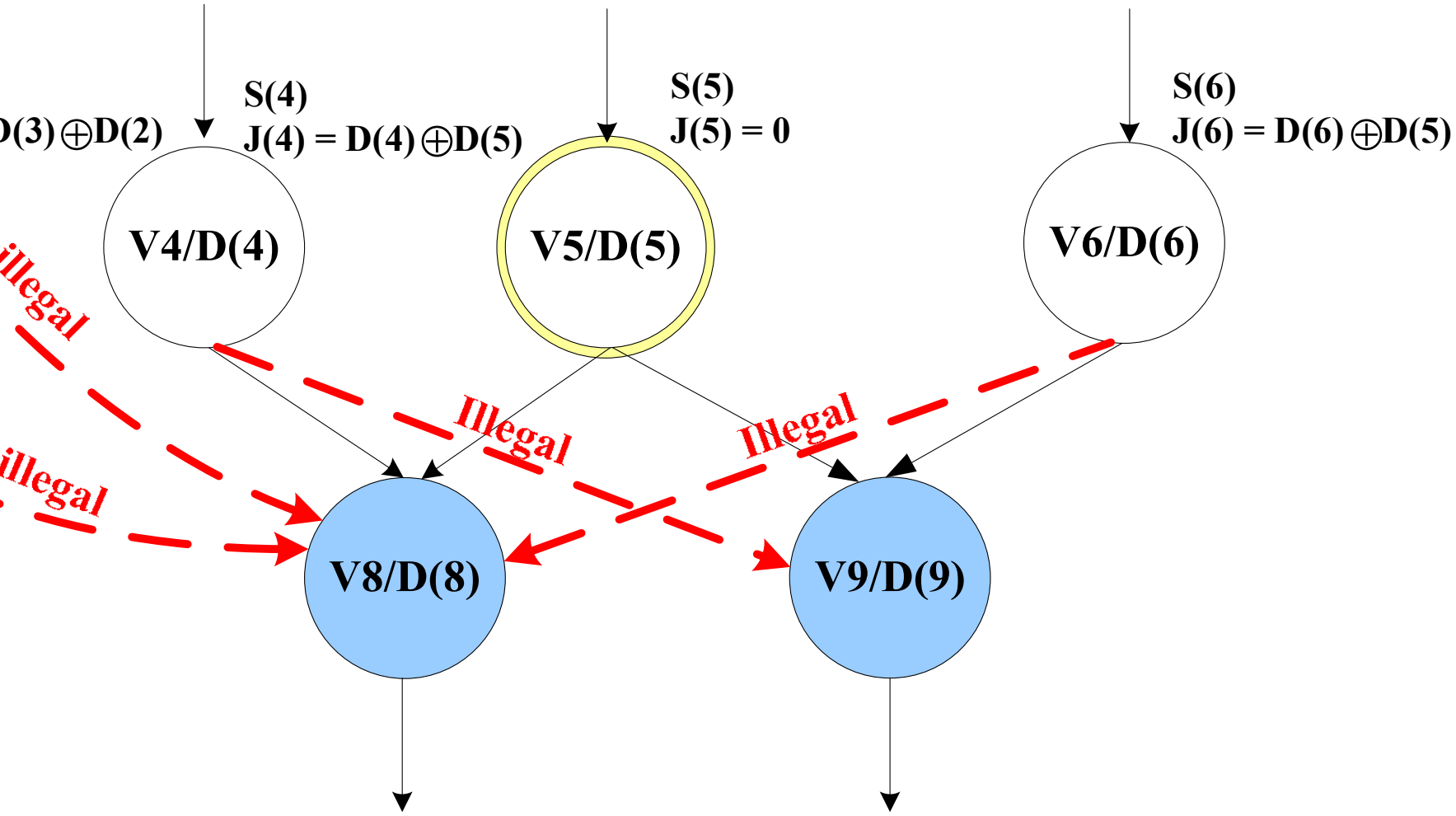
Sharing Primary Node

→ Decrease Fault Detection Coverage

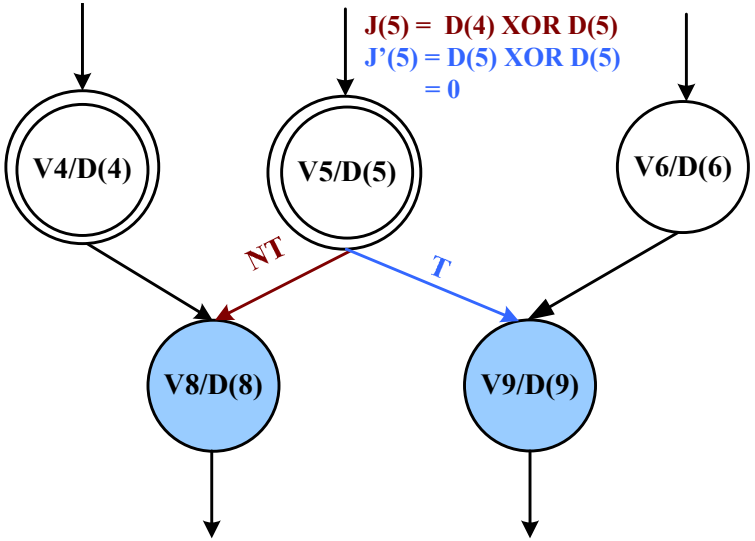


☞ If V_8 and V_9 share V_5 as their primary node, illegal control flow changes as "$V_6 \rightarrow V_8$" and "$V_4 \rightarrow V_9$" cannot be detected!

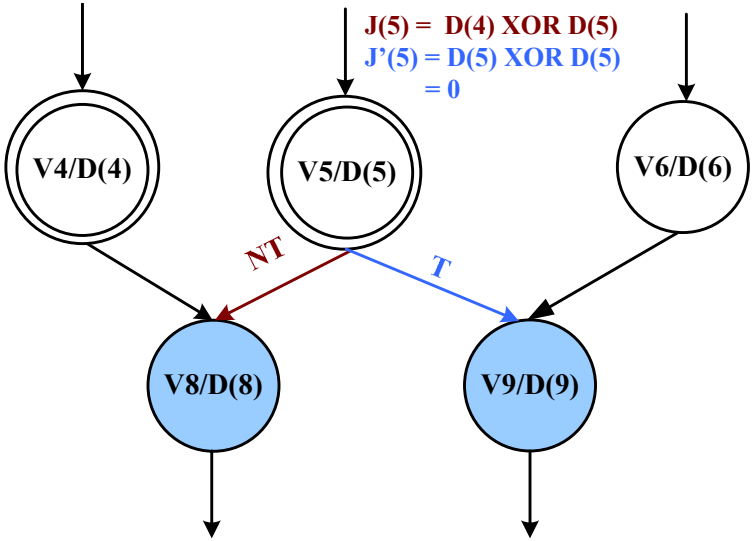
A Close Look


D(3) ⊕D(2)
S(4)
J(4) = D(4) ⊕D(5)
S(5)
J(5) = 0
S(6)
J(6) = D(6) ⊕D(5)
V4/D(4)
V5/D(5)
V6/D(6)
illegal
illegal
Illegal
Illegal
V8/D(8)
V9/D(9)

Two Justifying Signatures with a Node

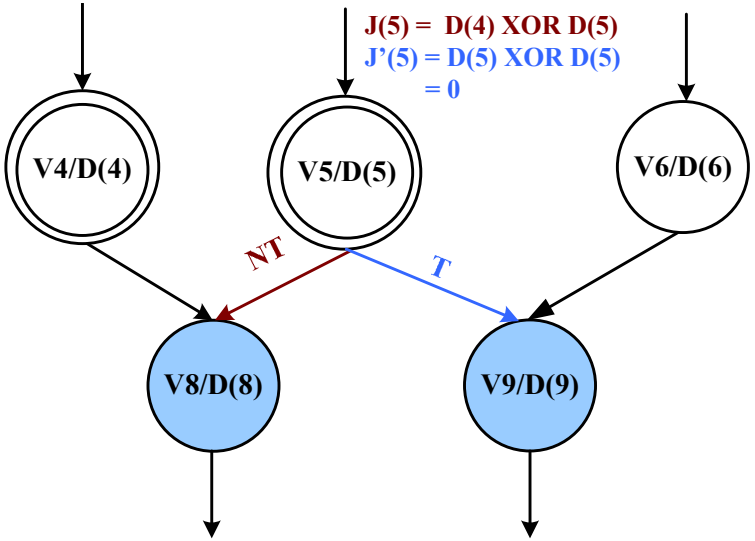


Two Justifying Signatures with a Node



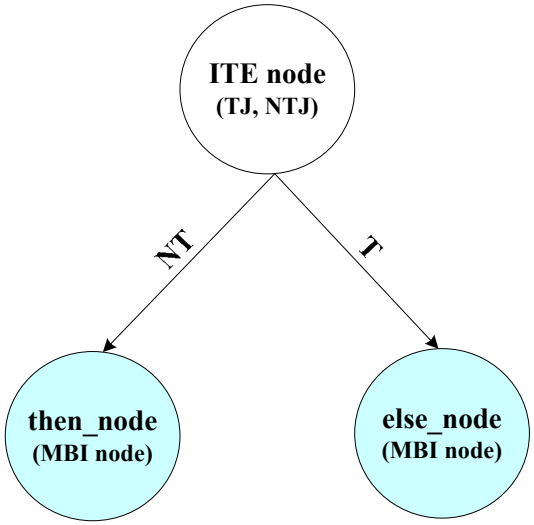
1. Exit of the node is a conditional branch, i.e., an if-then-else node;
2. Both branch destinations are MBI nodes.

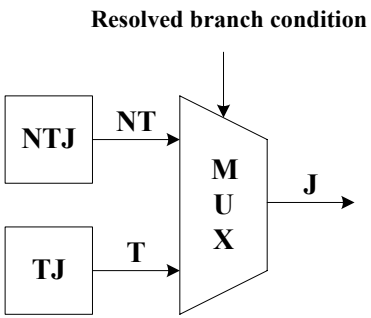
Two Justifying Signatures with a Node



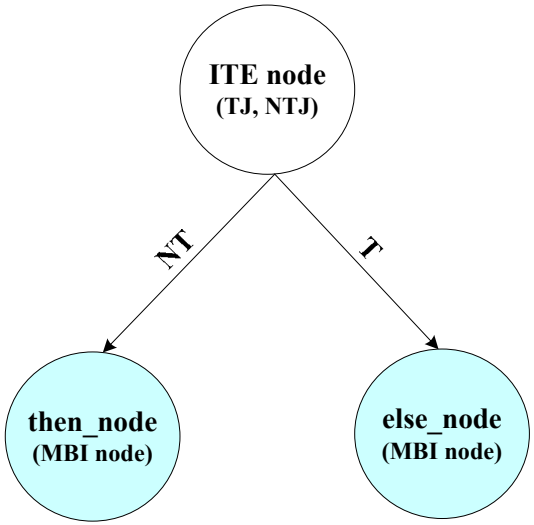
☞ If J(5) used for taken path checking, end up a false-alarm!

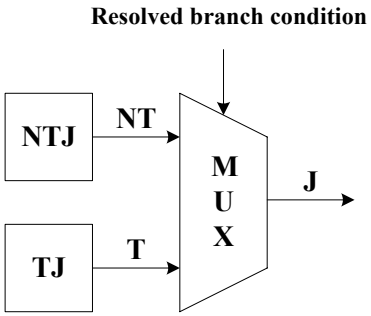
Hardware Approach for Two Justifying Signatures





Hardware Approach for Two Justifying Signatures





1. **Compile time**:
   1.1 Check directed edge of the CFG;
   1.2 If taken edge, place justifying signature into TJ register;
   1.3 If not-taken edge, place into NTJ register.
2. **Run time**: Resolved branch condition chooses the appropriate registers.

Hardware Enhancement for Control Flow Checking

1. Three additional instructions;
2. On-chip control flow checker;
3. Algorithm for embedding three additional instructions into programs.

PASCAL
University of California, Irvine

Three Additional Instructions

1. `SIC imm1, imm2`(SIgnature Checking)
   Compile time: set imm1 = S(i) and imm2 = D(i)
   Run time: update $G = G \oplus imm1$ and then check if (G = imm2);

PASCAL
University of California, Irvine

Three Additional Instructions

1. `SIC imm1, imm2`(SIgnature Checking)
   Compile time: set imm1 = S(i) and imm2 = D(i)
   Run time: update G = G ⊕ imm1 and then check if (G = imm2);
2. `SIJ imm1, imm2`(SIgnature Justifying)
   Compile time: If (ITE node) imm1 = D(i) ⊕ D(j) for NTJ and imm2 = D(i) ⊕ D'(j) for TJ
   else imm1/imm2 = D(i) ⊕ D(j)
   Run time: resolved branch condition select justifying signature;

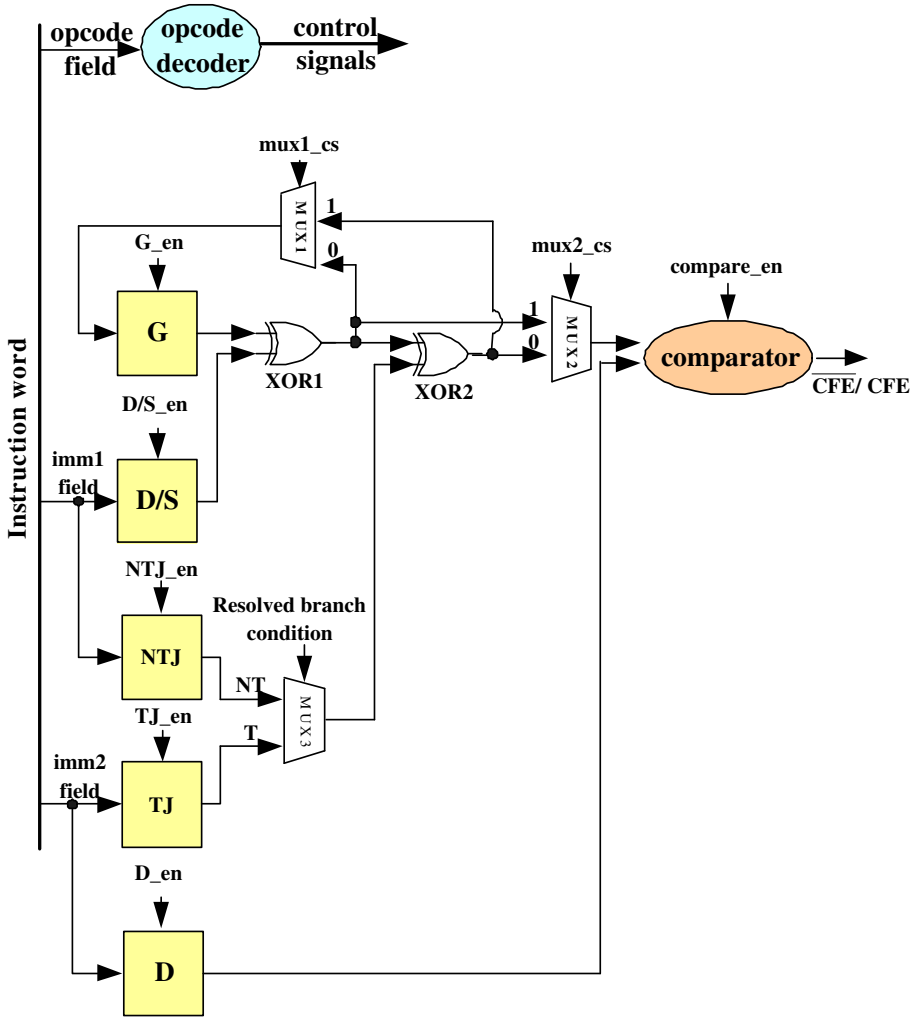
PASCAL
University of California, Irvine

Three Additional Instructions

1. `SIC imm1, imm2`(SIgnature Checking)
   Compile time: set imm1 = S(i) and imm2 = D(i)
   Run time: update G = G ⊕ imm1 and then check if (G = imm2);
2. `SIJ imm1, imm2`(SIgnature Justifying)
   Compile time: If (ITE node) imm1 = D(i) ⊕ D(j) for NTJ and imm2 = D(i) ⊕ D'(j) for TJ
   else imm1/imm2 = D(i) ⊕ D(j)
   Run time: resolved branch condition select justifying signature;
3. `SIJC imm1, imm2`(SIgnature Justifying Checking)
   Compile time: set imm1 = S(i) and imm2 = D(i)
   Run time: update G = G ⊕ imm1 ⊕ NTJ/TJ and then check if (G = imm2).

PASCAL
University of California, Irvine

On-Chip Control Flow Checker



Embedding Three Additional Instructions Algorithm (I)

Derive CFG of the given program;
Assign a unique state code D(i) to every node V_i;
For every node V_i in the CFG do:
  If (V_i is not an MBI node) then
    Compute its reference signature as:

$$S(i) = D(i) \oplus D(pred(V_i));$$

    Place the instruction "`SIC imm1, imm2`"
    at beginning of node V_i and before `SIJ`, if any;
    Assign the values of `imm1` and `imm2` as:

imm1 = S(i) and imm2 = D(i).

  Else /*V_i is an MBI node*/

(to be continued)

Embedding Three Additional Instructions Algorithm (II)

Else /*V_i is an MBI node*/

Select a primary node (assume V_j);

Assign reference signature of V_i as:

$$S(i) = D(i) \oplus D(j);$$

Place the instruction "`SIJC imm1, imm2`"

as beginning of node V_i and before SIJ, if any;

Assign the values of imm1 and imm2 as:

imm1 = S(i) and imm2 = D(i);

For every node V_k is an immediate predecessor of V_i

(to be continued)

Embedding Three Additional Instructions Algorithm (III)

For every node V_k is an immediate predecessor of V_i

Place instruction "`SIJ imm1, imm2`"

into node V_k and after SIC/SIJC instructions;

Assign the values of imm1 and imm2 as:

If $V_k \rightarrow V_i$ is a taken path:

imm1 = X(don't care) and imm2 = $D(k) \oplus D(j)$;

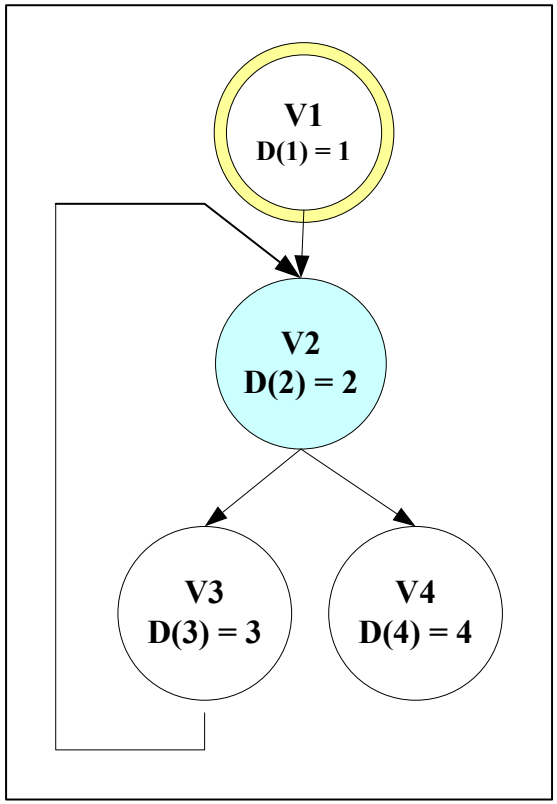
Else if $V_k \rightarrow V_i$ is a not-taken path:

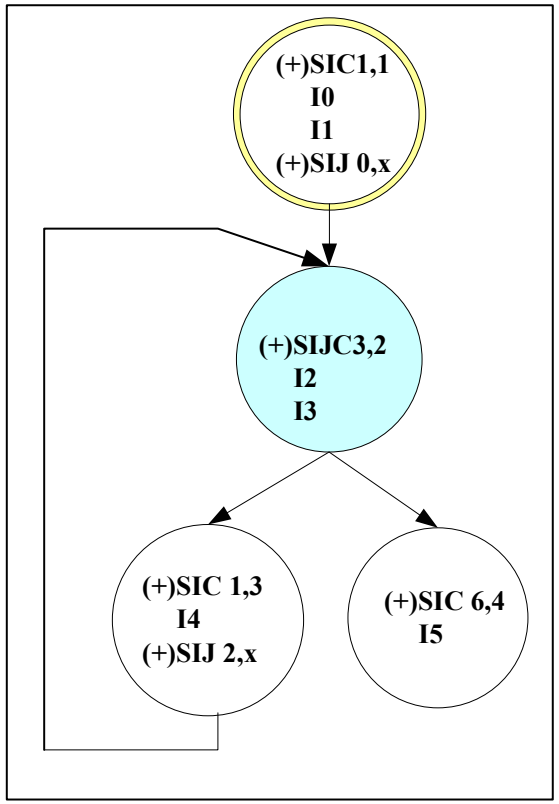
imm1 = $D(k) \oplus D(j)$ and imm2 = X;

Else $V_k \rightarrow V_i$ is not a conditional branch path

imm1 = $D(k) \oplus D(j)$ and imm2 = X;

Wrap-up Example





☞Additional instructions for the control flow checking are a maximum of one or two for each node.

PASCAL
University of California, Irvine

Conclusion

1. Protect processors from the control flow error
2. Develop an assigned-signature control flow checking algorithm
3. Improve the checking efficiency by the dedicate instructions and an on-chip hardware checker

PASCAL
University of California, Irvine

Thank You!

For more information:

URL: `http://pascal.eng.uci.edu`

Email: `xiaobinl@uci.edu`

PASCAL
University of California, Irvine