Dynamic Dead-Instruction Detection and Elimination

J. Adam Butts and Guri Sohi

University of Wisconsin–Madison

{butts,sohi}@cs.wisc.edu

ASPLOS-X

San Jose, California

October 8, 2002

Overview

Non-trivial # of instructions create values that are never used

- Most useless instances are generated by partially-dead instructions
- Aggressive compiler optimization increases incidence

Useless instances are predictable

- Predictor identifies **91%** of useless instructions

Exploit by dynamically eliminating resource costs

- Avoid handling useless instructions
- **May enable more aggressive code motion**

Outline

Overview

Useless Instructions

- Definition
- Frequency
- Characterization

Identifying Useless Instructions Dynamically

Eliminating Useless Instructions

Summary

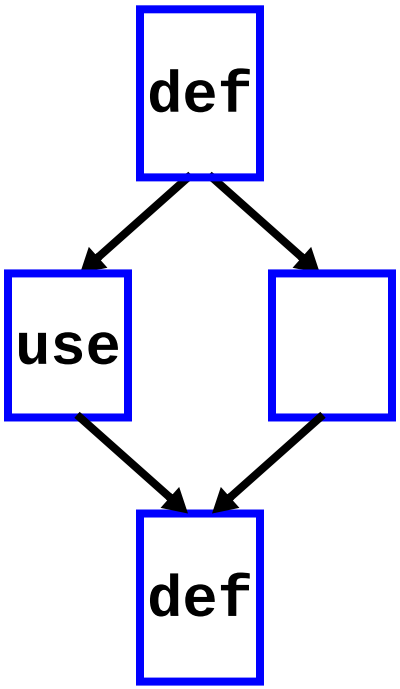
Useless Instructions

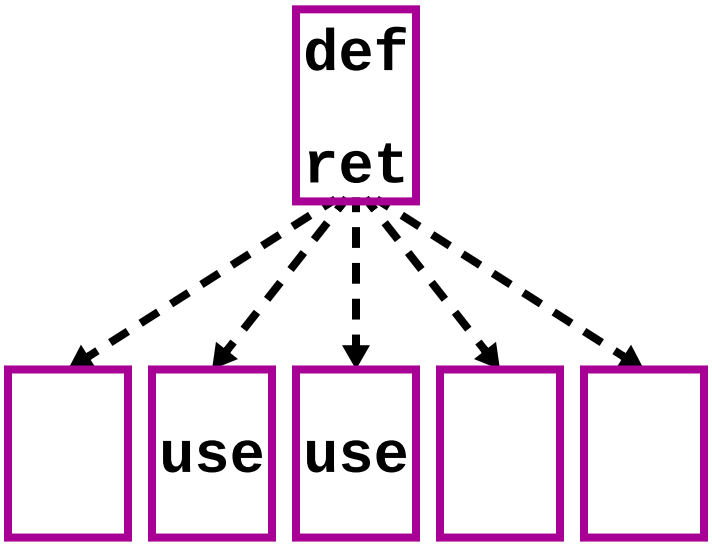
Definition: dynamic instruction that generates an unused value

- One static instruction may generate useful and useless instances

Sources of useless instructions

- Partially-dead instructions
- Dead instructions requiring interprocedural analysis to detect





Costs of Useless Instructions

Direct costs of useless instructions

- Register file bandwidth
- Instruction window slots
- Cache bandwidth
- ALU occupancy

Indirect cost: less effective compiler optimization

- Compiler balances estimated costs and benefits
- Useless instructions are a **cost** of optimization
- Some optimizations may be blocked by perceived cost

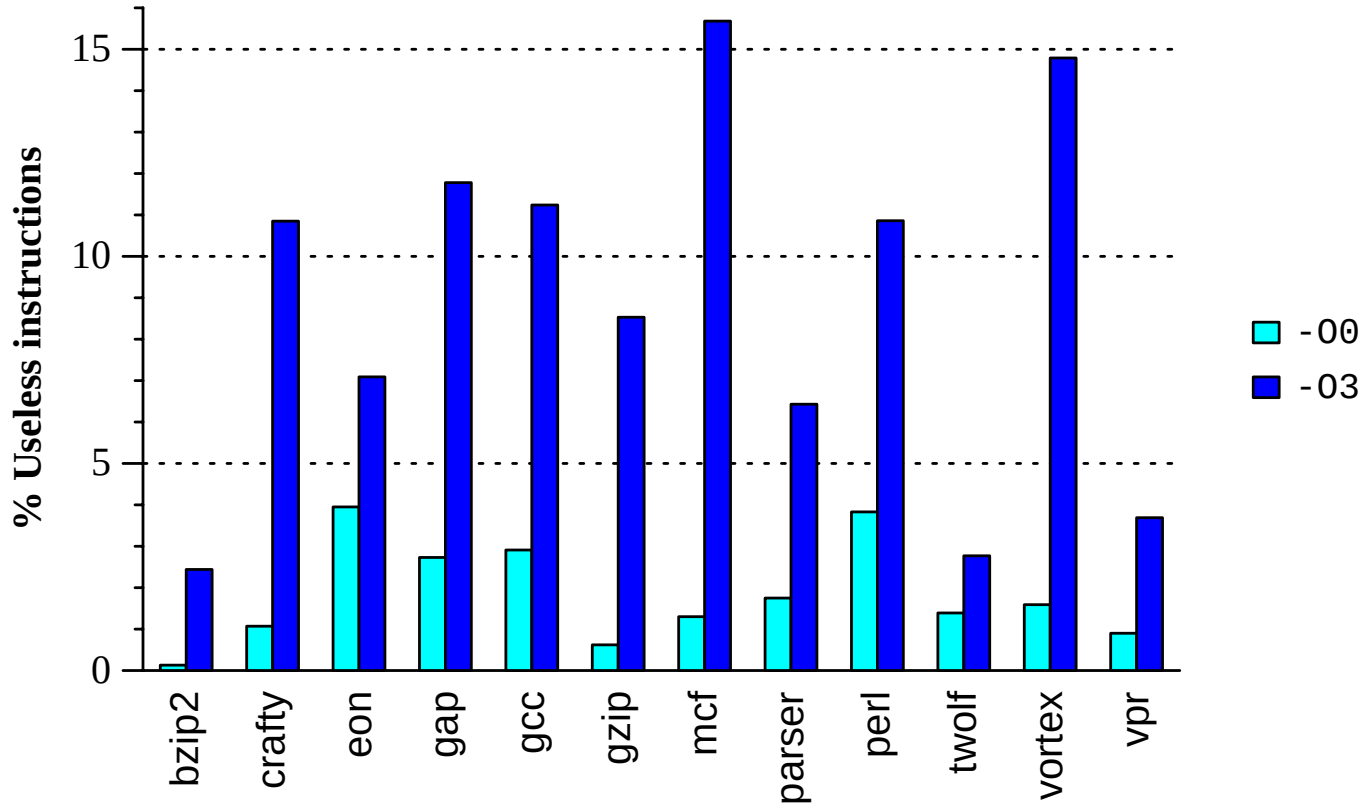
Potential for more aggressive optimization

Role of Compiler Optimization

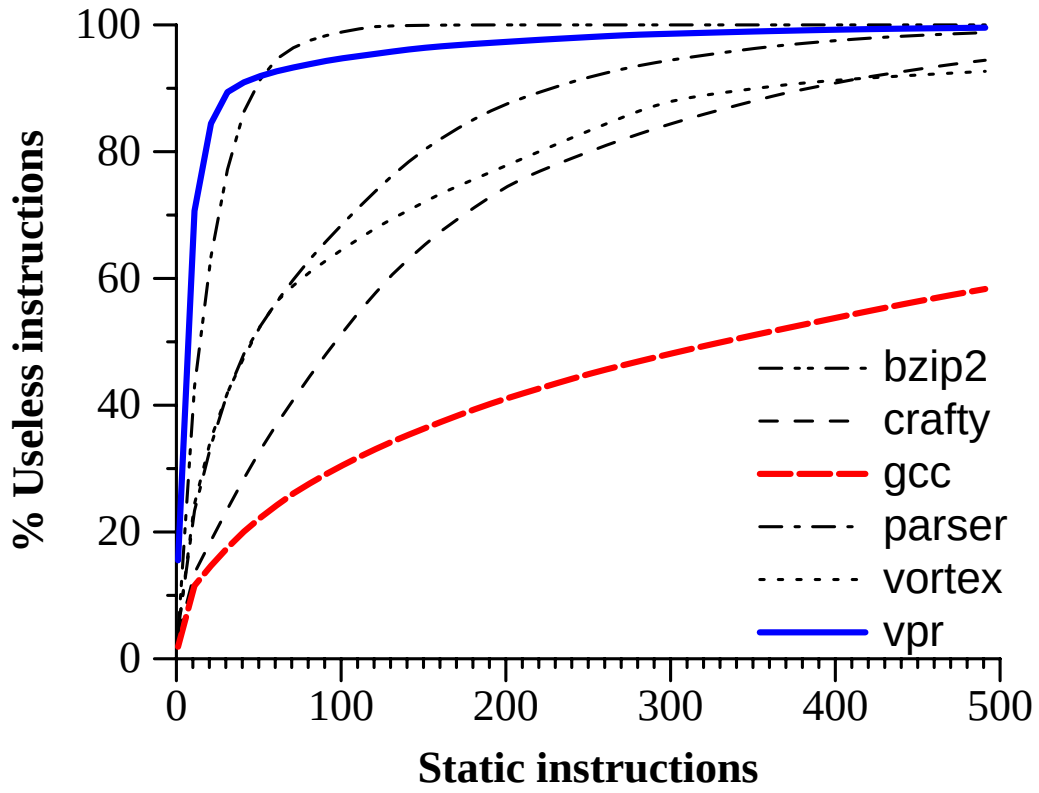
Optimization increases fraction of useless instructions

- Absolute number also increases
- Due primarily to hoisting during instruction scheduling

3 to 16% of non-NOP instructions are useless



Contribution by Static Instructions

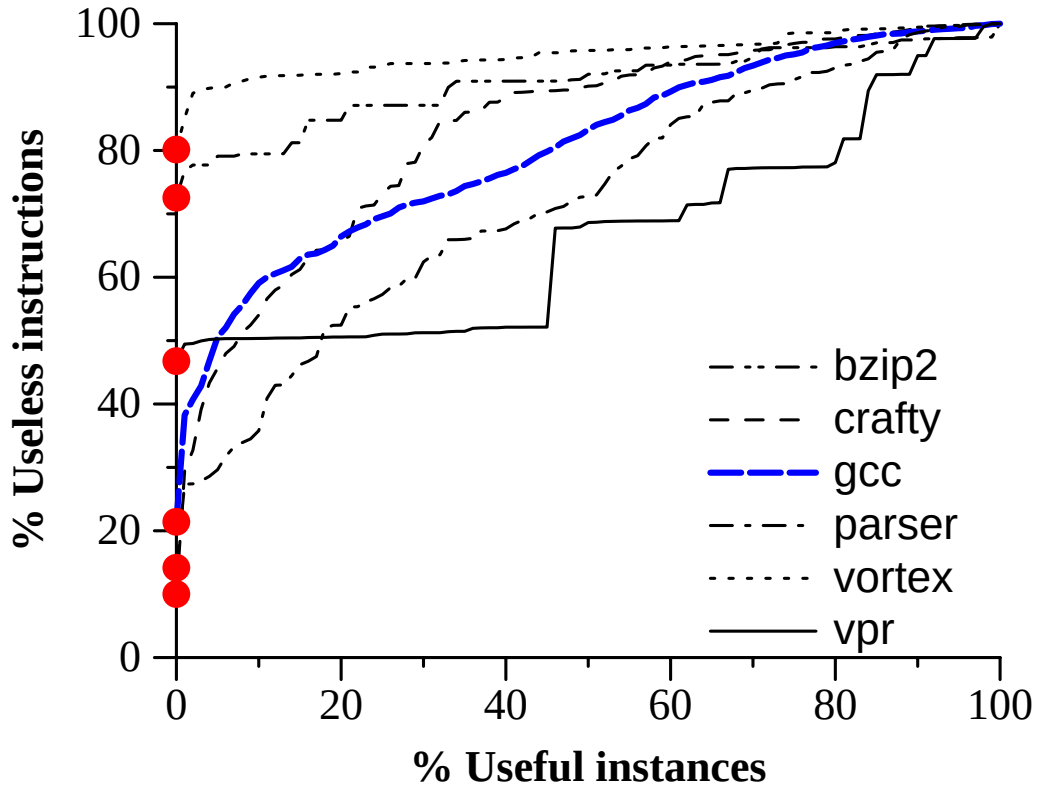


Small # of static instructions generate most useless instances

- Related to code footprint
- Contributing instructions exhibit temporal locality

Instruction-based prediction is likely to be effective

Contribution by Usefulness



Statically-dead instructions

- Analysis fails to prove dead
- Value live on unexercised path

Majority of useless instances from mostly-dead instructions

Must be able to distinguish useful instances

Outline

Overview

Useless Instructions

Identifying Useless Instructions Dynamically

- Predictor organization
- Forward control flow signatures
- Evaluation

Eliminating Useless Instructions

Summary

Useless Instruction Prediction

Goal: identify useless instructions early in pipeline

- Earlier identification → greater opportunity to exploit

Solution: remember static instructions with useless instances

- Cache-like predictor indexed with instruction PC
- Observe rename instruction stream to detect mispredictions

Problem: same static instructions also generate live values

- Results in frequent mispredictions (16%)

Forward Control Flow

How to differentiate useless and useful instances?

Future **control flow uniquely determines usefulness**

- Uses depend solely on path taken **after** value is generated

Future control flow is available

- "Deadness" predictions needed in **middle** of pipeline
- Control flow predictions are made in **early** pipeline stage

Not quite perfect

- Control flow predictions may not be correct
- Pipeline depth limits lookahead

Control Flow Signatures

Signature encodes predictions for upcoming indirect or conditional branches

- If next branch is indirect (e.g., return), encode **target address**
- Otherwise, encode available predicted branch **directions**
- Type and number of predictions is also encoded

Generating a prediction requires a PC and signature match

Future Signature Formats

Predictor Performance

Predictor organization

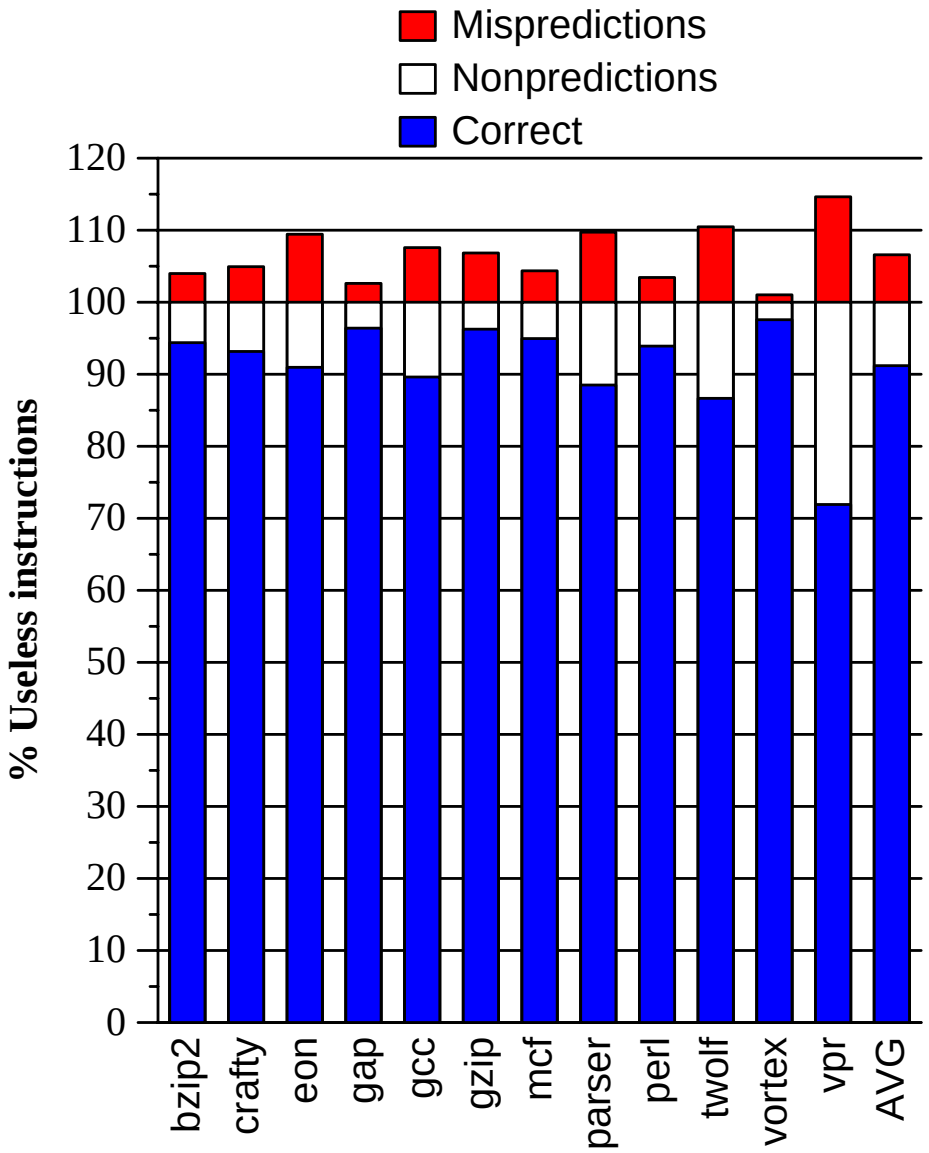
- 2K entry, 4-way set assoc.
- 5-bit future control flow sig.
- Confidence mechanism
- 4.6 KB, relaxed timing

Improved accuracy

- **7%** misprediction rate

High coverage

- **91%** of useless instructions are identified **correctly**



Outline

Overview

Useless Instructions

Identifying Useless Instructions Dynamically

Eliminating Useless Instructions

- Mechanism of elimination: example
- Evaluation

Summary

Eliminating Useless Instructions

Verification table

	dead	PUT
r5:	0	—
r6:	—	—

Predicted Useless Table

	V	instruction	ROB
4:	0	—	—

Reorder Buffer

	instruction	PUT
6:	ldq r5, 8(sp)	—
7:		
8:		
21:		

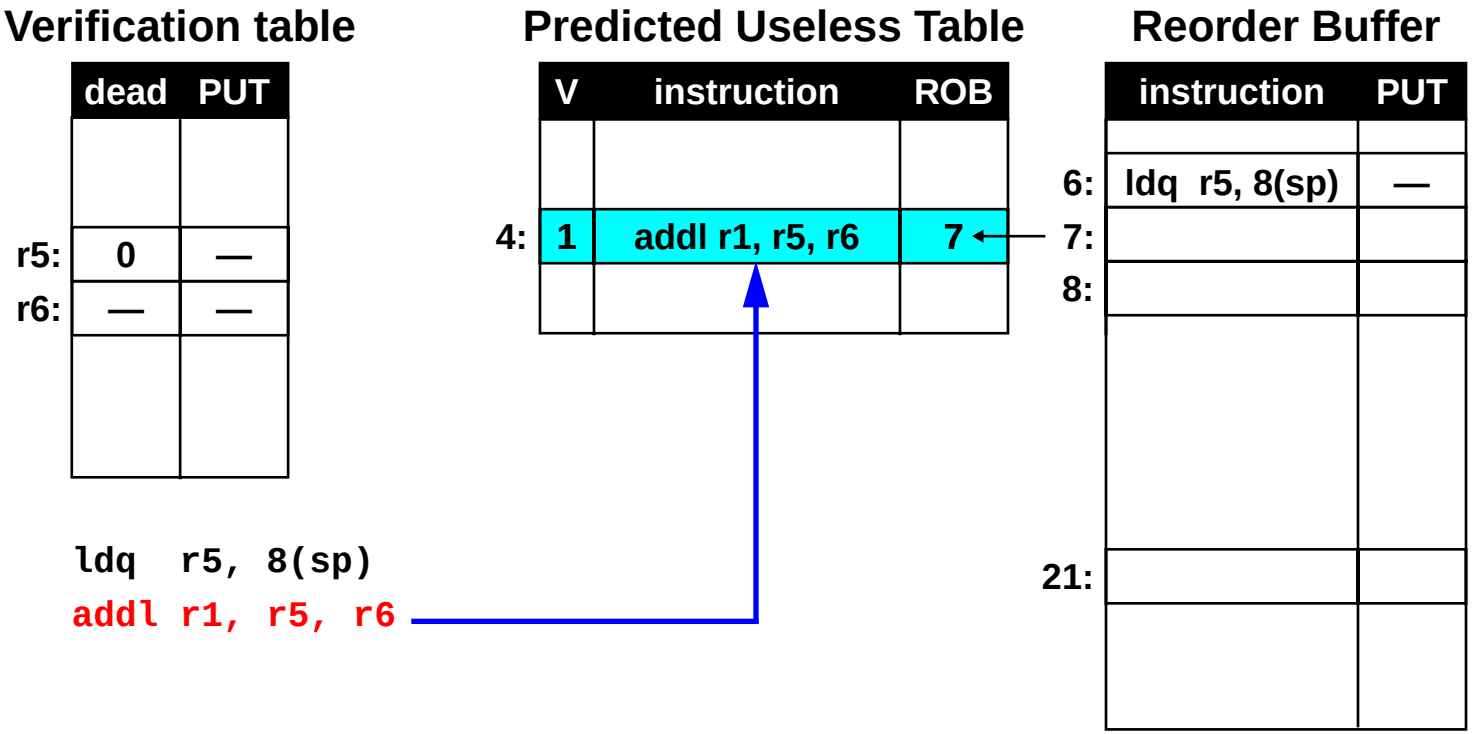
```
ldq  r5, 8(sp)
```

Add VT (verification table) to track prediction status

Add PUT (predicted useless table) to enable recovery

Augment ROB with pointers into PUT

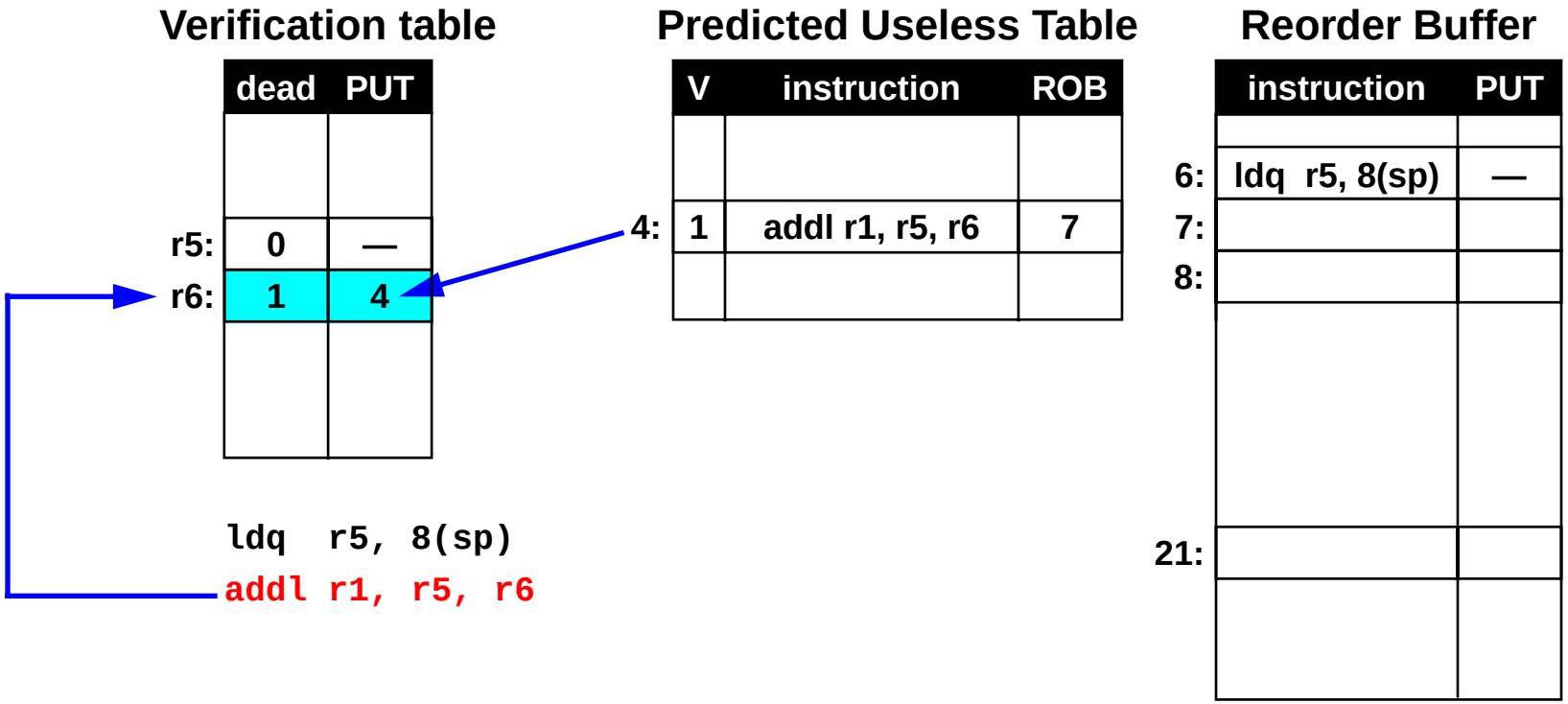
Eliminating Useless Instructions



When an instruction is predicted to be useless:

- A free PUT entry is allocated for the instruction

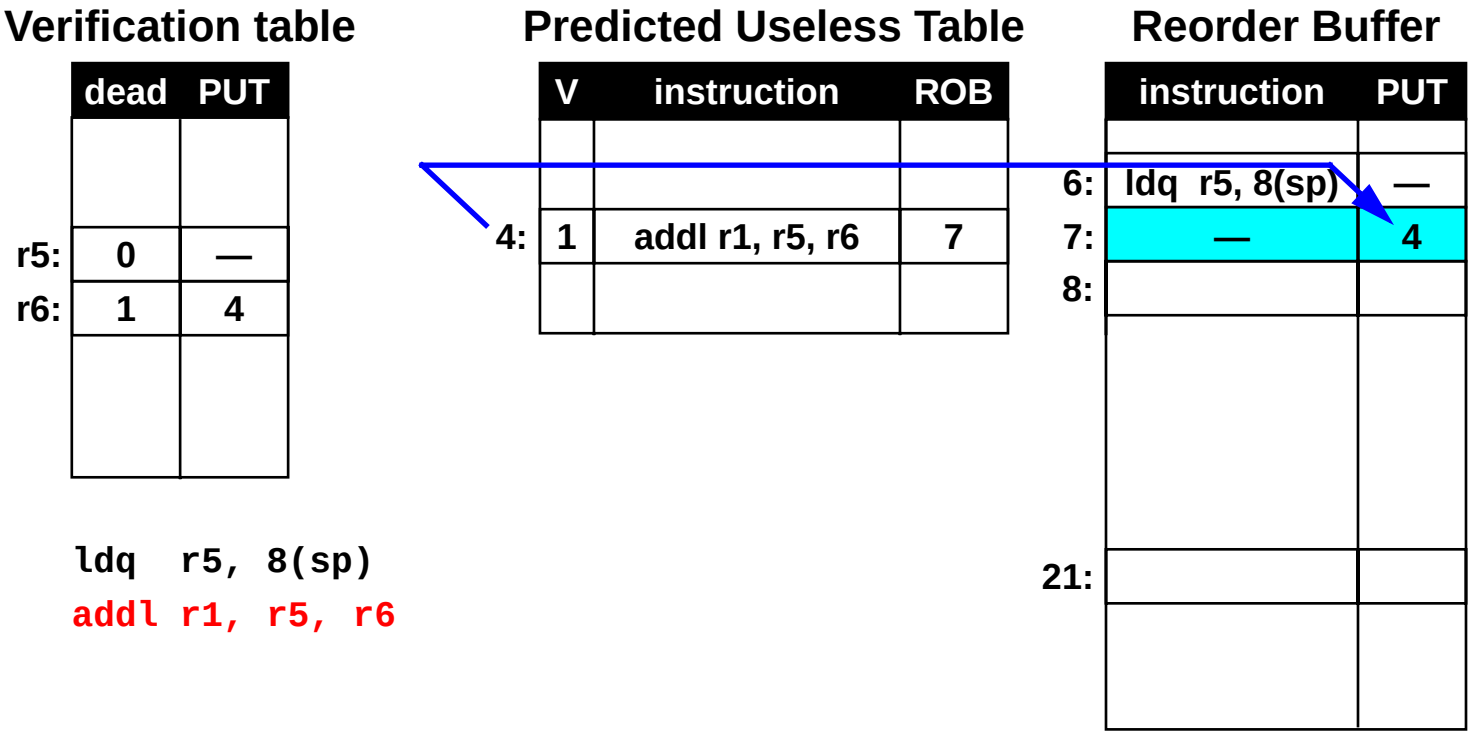
Eliminating Useless Instructions



When an instruction is predicted to be useless:

- A free PUT entry is allocated for the instruction
- The VT is updated with a pointer to the PUT entry

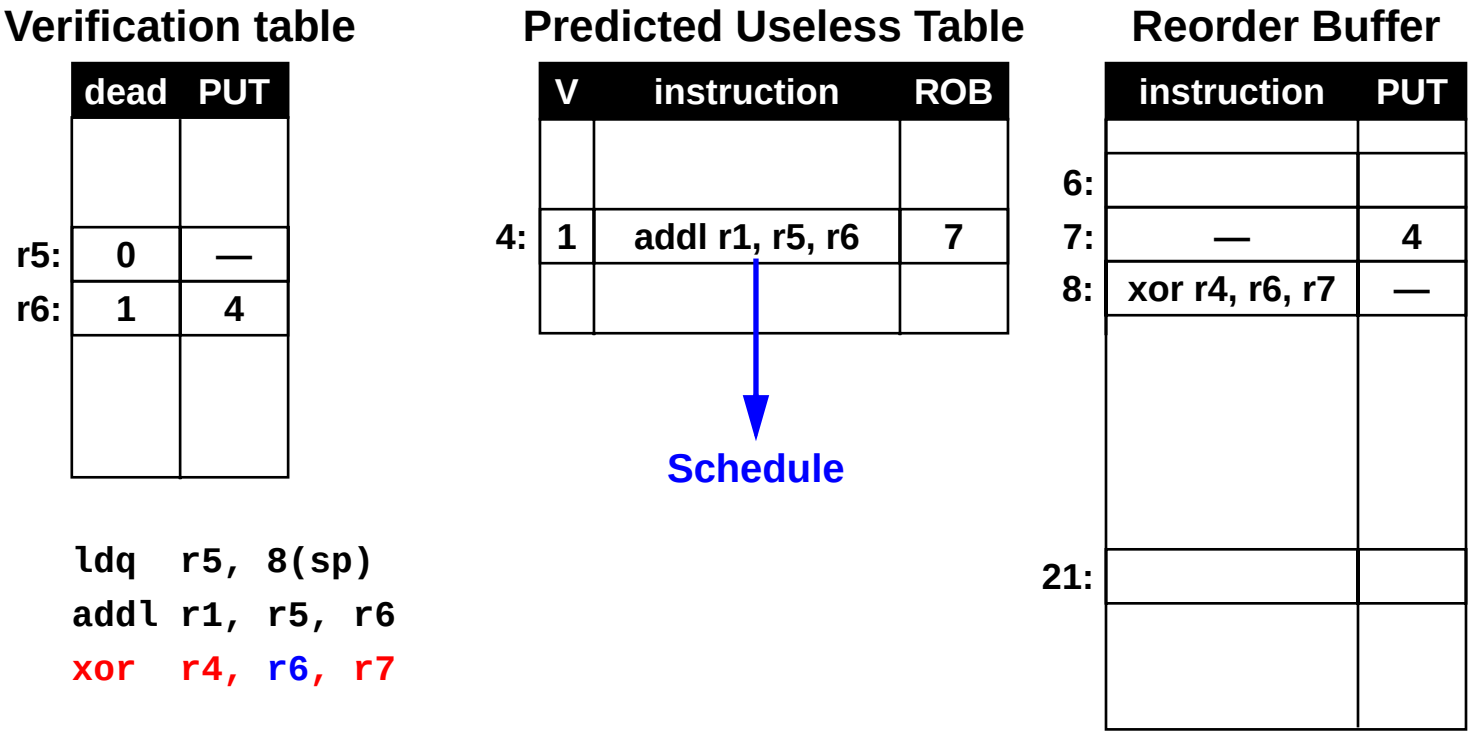
Eliminating Useless Instructions



When an instruction is predicted to be useless:

- A free PUT entry is allocated for the instruction
- The VT is updated with a pointer to the PUT entry
- A dummy entry is placed in the ROB with a pointer to the PUT entry

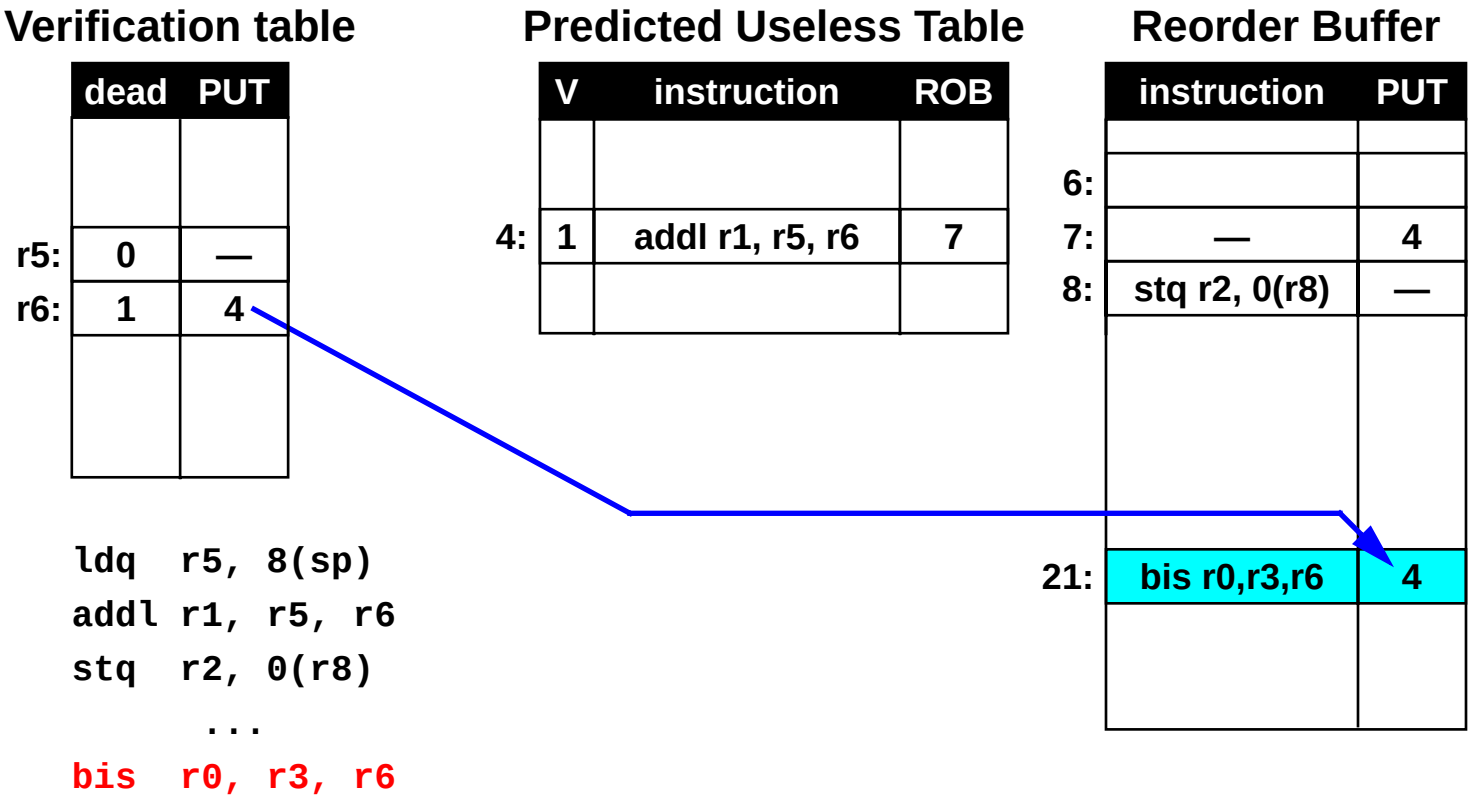
Eliminating Useless Instructions



Subsequent instructions are handled normally except:

- A use of a predicted useless register causes the corresponding PUT entry to be placed into the instruction window for scheduling
- Exceptions cause entire PUT to be flushed into the window

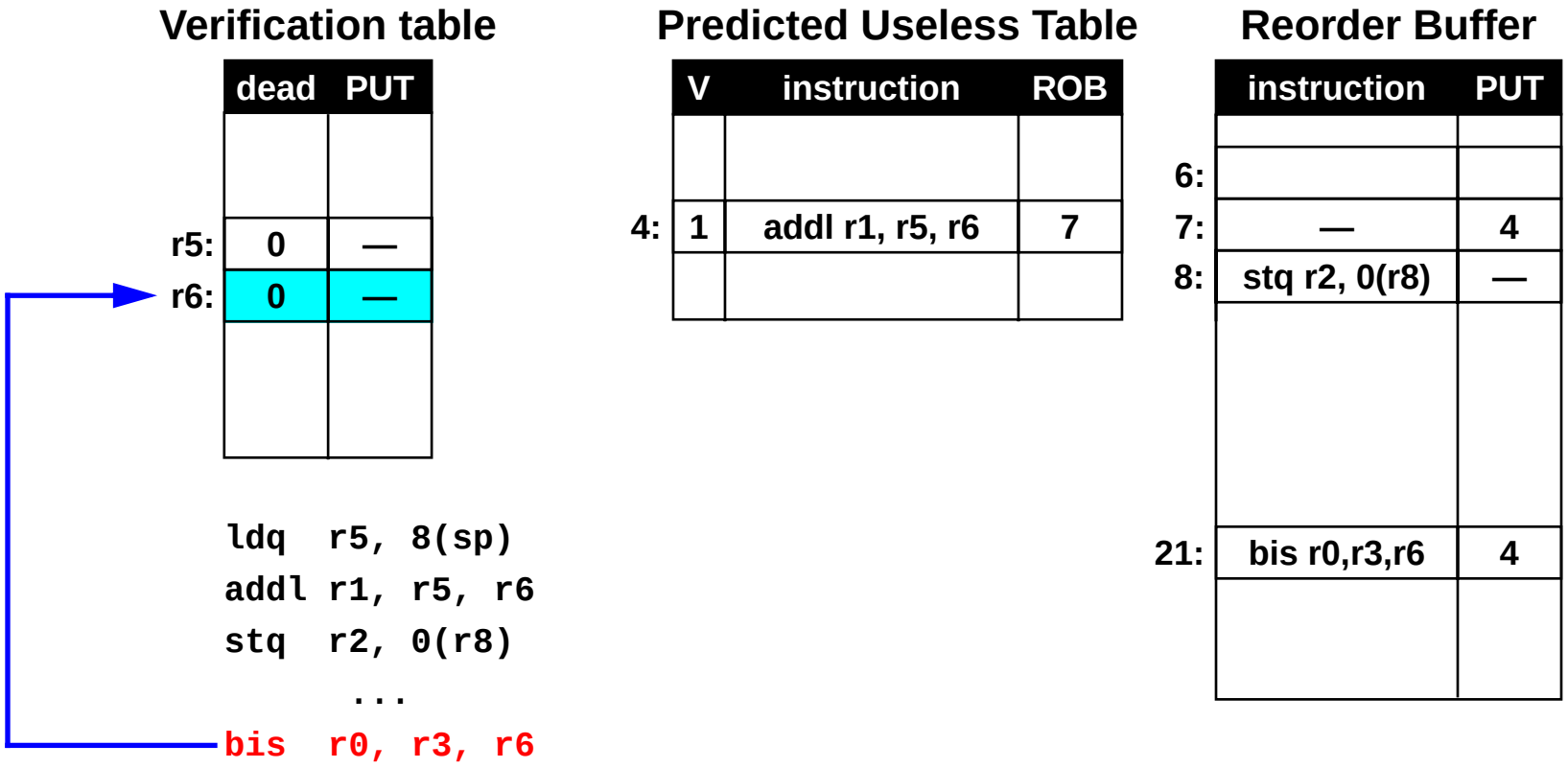
Eliminating Useless Instructions



When an overwrite of the register is observed:

- The PUT pointer is copied into the ROB entry of the verifying insn

Eliminating Useless Instructions



When an overwrite of the register is observed:

- The PUT pointer is copied into the ROB entry of the verifying insn
- The VT is updated normally

Eliminating Useless Instructions

Verification table

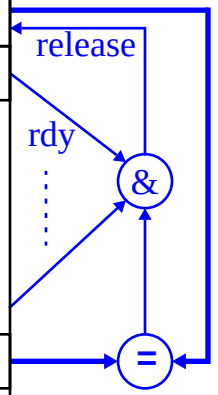
	dead	PUT
r5:	0	—
r6:	0	—

Predicted Useless Table

	V	instruction	ROB
4:	1	addl r1, r5, r6	7

Reorder Buffer

	instruction	PUT
6:		
7:	—	4
8:	stq r2, 0(r8)	—
21:	bis r0,r3,r6	4




```
ldq  r5, 8(sp)
addl r1, r5, r6
stq  r2, 0(r8)
      ...
bis  r0, r3, r6
```

Retiring a predicted useless instruction:

- The overwriting instruction must be ready to retire
- All intervening instructions must be ready to retire

Eliminating Useless Instructions

Verification table

	dead	PUT
r5:	0	—
r6:	0	—

Predicted Useless Table

	V	instruction	ROB
4:	0	—	—

Reorder Buffer

	instruction	PUT
6:		
7:		
8:	stq r2, 0(r8)	—
21:	bis r0,r3,r6	4

```
ldq  r5, 8(sp)
addl r1, r5, r6
stq  r2, 0(r8)
       ...
bis  r0, r3, r6
```

Retiring a predicted useless instruction:

- The overwriting instruction must be ready to retire
- All intervening instructions must be ready to retire
- The ROB and PUT entries are reclaimed

Evaluation

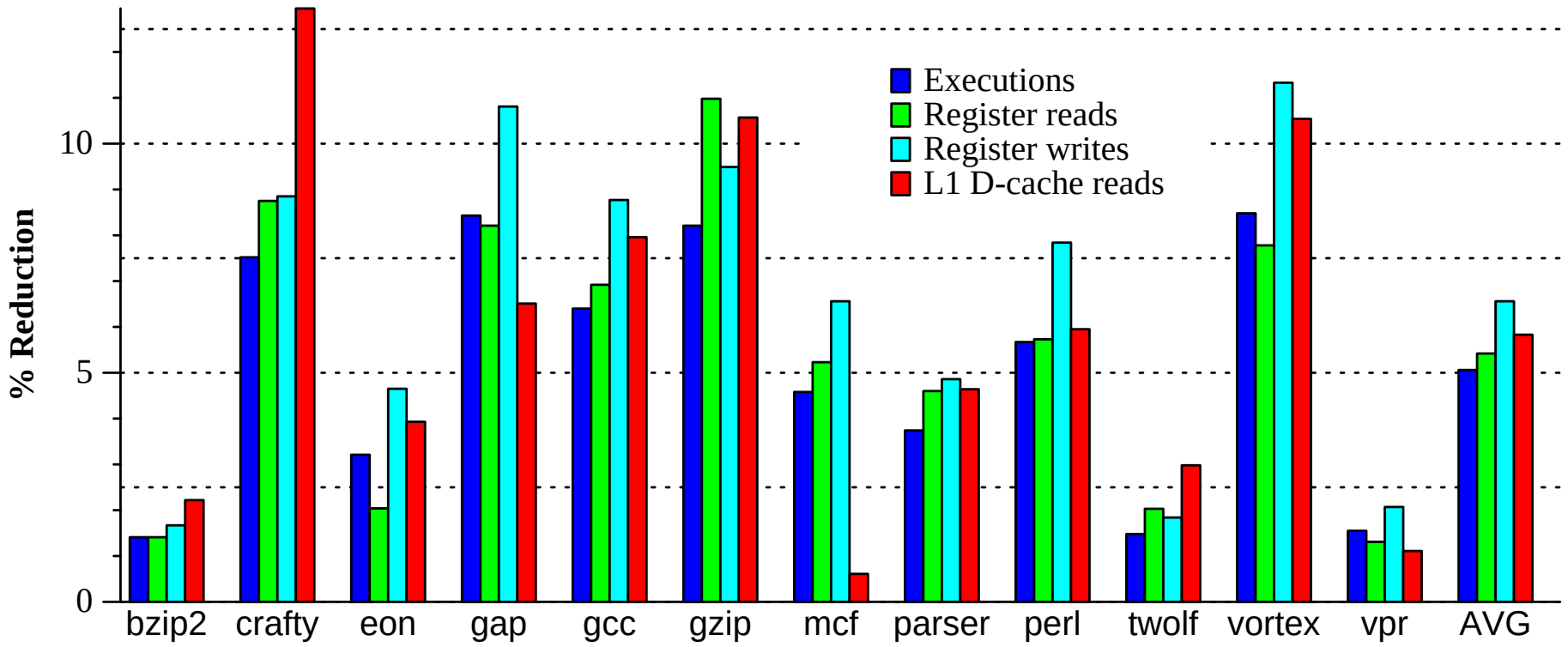
Execution-driven simulation

- 4-wide fetch, issue, retire
- 256-entry ROB, 64-entry scheduling window
- 12 KB YAGS branch predictor, RAS, cascaded indirect predictor
- 64 KB 2-way set associative L1 caches, unified 2MB 4-way L2

Useless instruction elimination parameters

- ROB threshold: **224 entries**
  - When ROB occupancy exceeds threshold, **abort prediction**
  - Prevents prediction verification from stalling retirement
- Predicted useless table: **32 entries**

Results: Resource Utilization



~5% reduction in utilization of many critical resources

- Several benchmarks see **>10%** reductions

Relative reductions depend on instruction mix

- ALU operations vs. loads
- 0-, 1-, and 2-input instructions

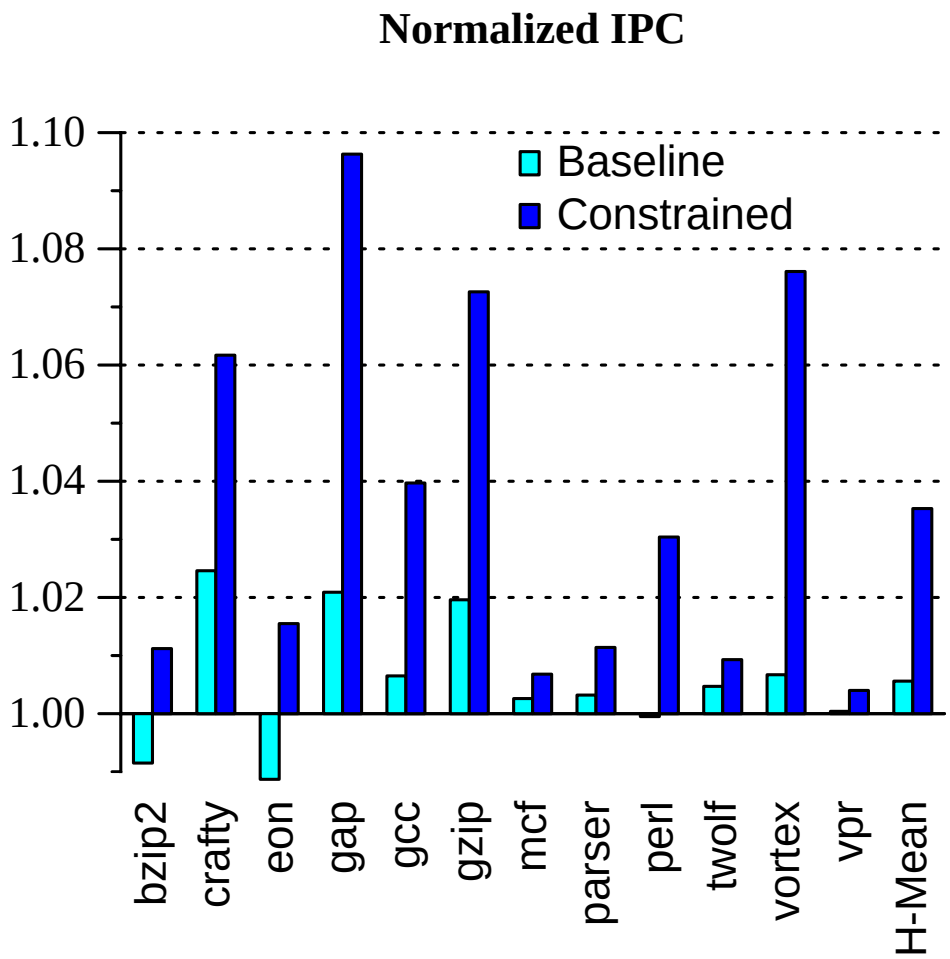
Results: Performance

Indirect performance impact

- Useless instructions are cheap
- Performance improvement under **resource contention**
  - Limited resources
  - Many useless instructions
  - SMT processors?

80% of ideal performance

- **NOT** mis-speculations
- Retirement holdup
- Instructions not eliminated
  - Unidentified (predictor)
  - Unverifiable (ROB size)



Outline

Overview

Useless Instructions

Identifying Useless Instructions Dynamically

Eliminating Useless Instructions

Summary

- Related work
- Conclusions

Related Work

Partial dead-code elimination (Knoop, Rüthing, and Steffen)

- Compiler algorithm to push down partially-dead instructions
- Does not address interprocedural sources
- At odds with benefits of hoisting

Exploiting dead value information (Martin, Roth, Fischer)

- Compiler identifies useless saves and restores around procedures
- Does not address partially-dead code

Conclusions

Programs exhibit non-negligible fraction of useless instructions

- Partially-dead instructions from code-motion are dominant source

Useless instances of instructions can be accurately predicted

- **91%** of useless instructions identified with **7%** misprediction rate
- Future control flow used to distinguish useless and useful instances

Useless instruction elimination

- Reduces resource utilization by an average of **5%**
- Results in performance improvement under contention

Optimizing compilers can ignore cost of useless instructions

Handling Loads

Mispredicted useless loads may be delayed

- Handle as any OoO processor

Loads may have side effects

- Memory-mapped I/O, page faults, illegal addresses
- Must execute these loads

Solutions

- ISA change to mark such loads
- Generate address and probe TLB to verify cacheable page

Aborted Predictions

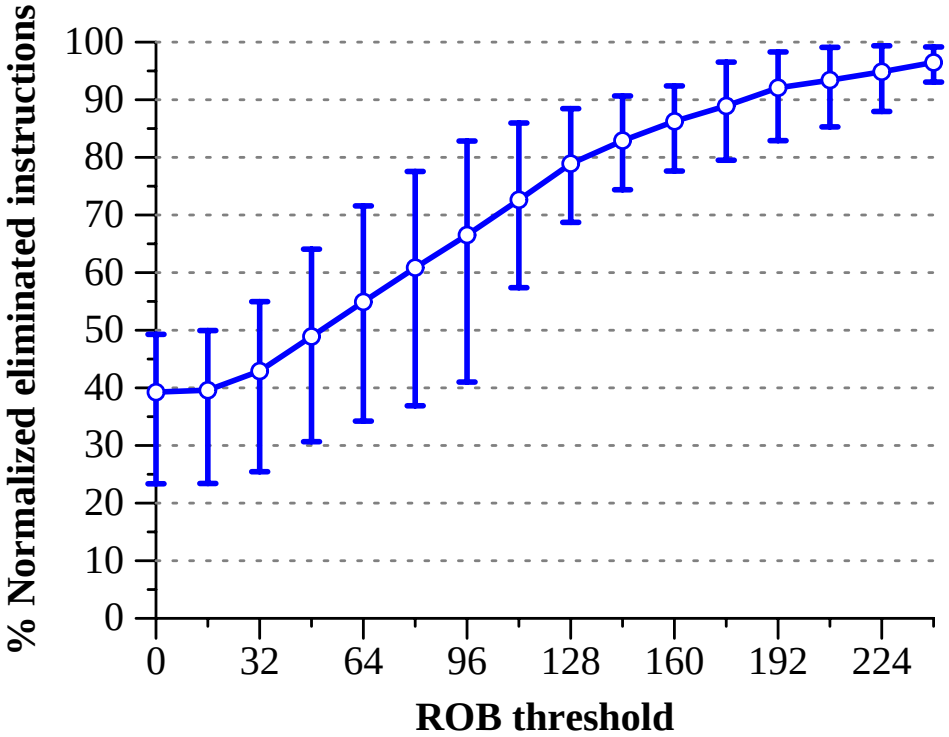
What if verifying instruction is not observable within ROB?

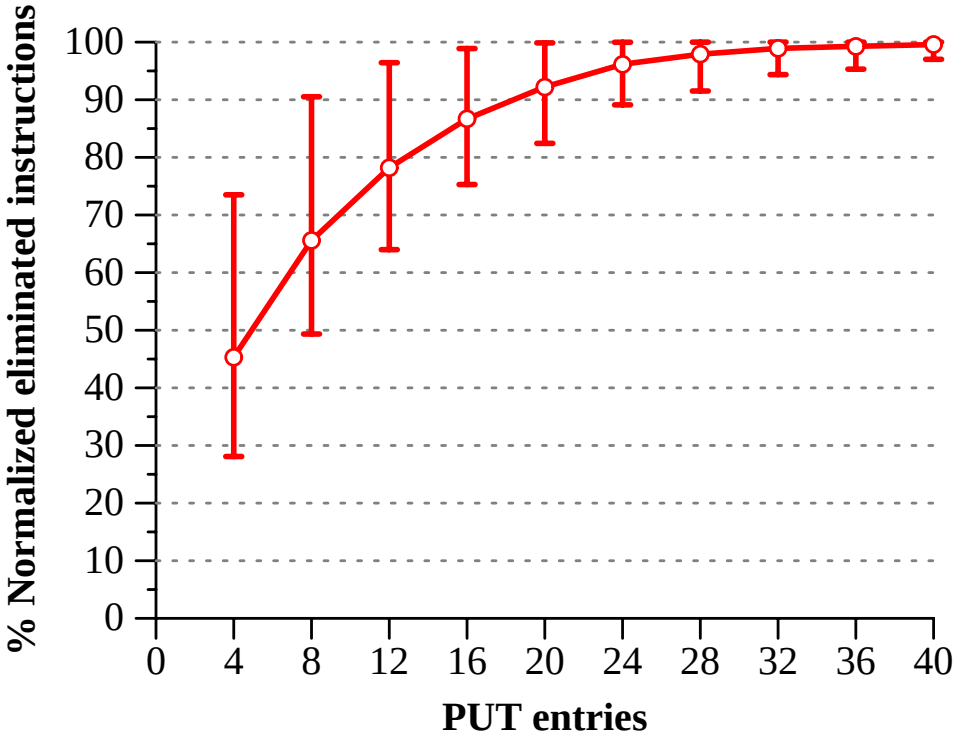
- Must detect or ROB will fill and machine will stall indefinitely
- Schedule instruction as for a misprediction (abort prediction)
- Stall from full ROB results in performance penalty

Avoid full stall by implementing ROB threshold

- If unverified instruction is head of ROB, **and**
- ROB occupancy exceeds threshold, then
- Abort prediction early

Parameter Sensitivity





Higher ROB threshold

- More instructions eliminated
- Performance peaks earlier due to diminishing returns plus larger retirement holdup

Larger PUT size

- More instructions eliminated
- More hardware overhead
- Scaling required with number of instructions in flight

Useless Instruction Breakdown

79% of useless instructions successfully eliminated

Causes of non-elimination

- **Non-predictions** (coverage)
- **Aborted predictions** (ROB full)
- **False mispredictions**
- **PUT full**