# Exploiting Short-Lived Variables in Superscalar Processors

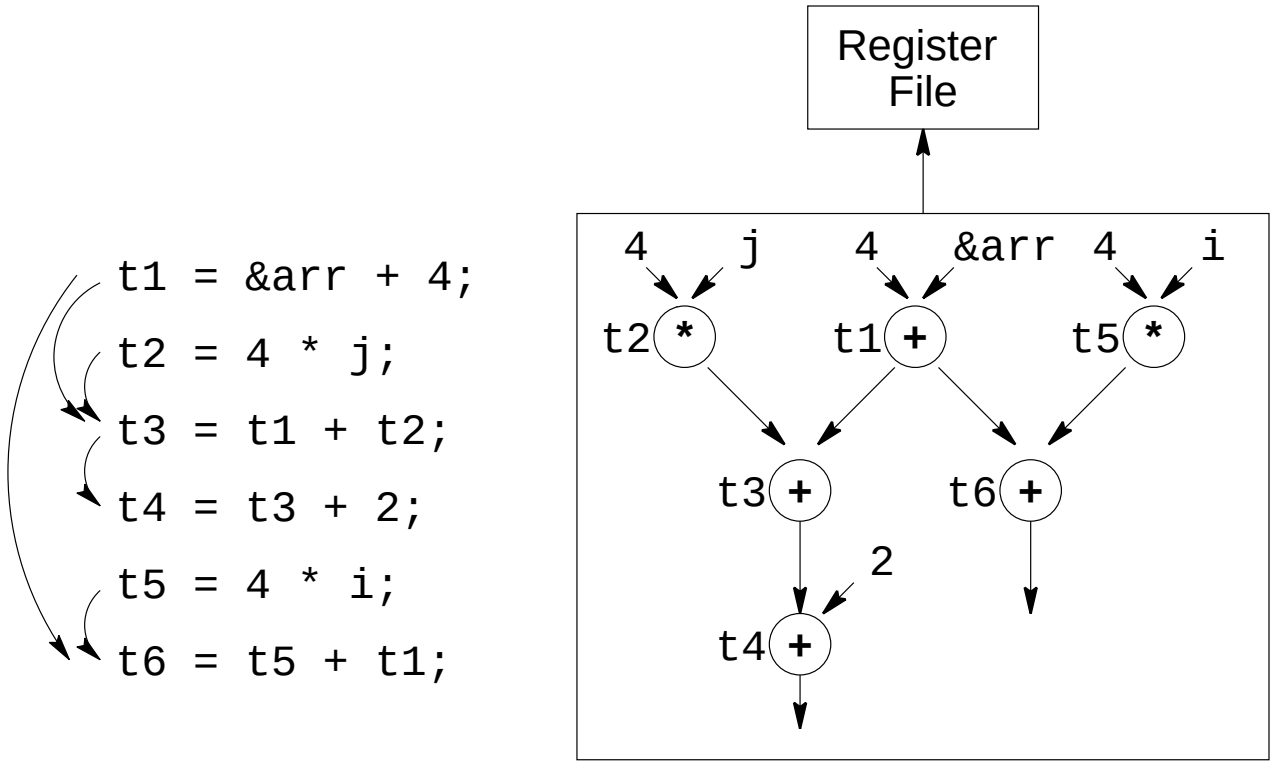
Luis Alfonso Lozano *
Guang R. Gao

ACAPS Group
School of Computer Science
McGill University

* Currently affiliated with Hewlett-Packard
California Language Laboratory

MICRO - 28
November 1995

# Preamble




- Dynamically scheduled processors build the data flow graph for a "window" of instructions
- Many of the values produced are only used by other instructions in the window
- Many values are "Short-Lived"

# Presentation Outline

- Introduction
- Motivating Example
- Problems to be studied
- Issues and solutions
- Allocation of Short-Lived Variables
- Experimental results
- Related work
- Conclusions

# Introduction

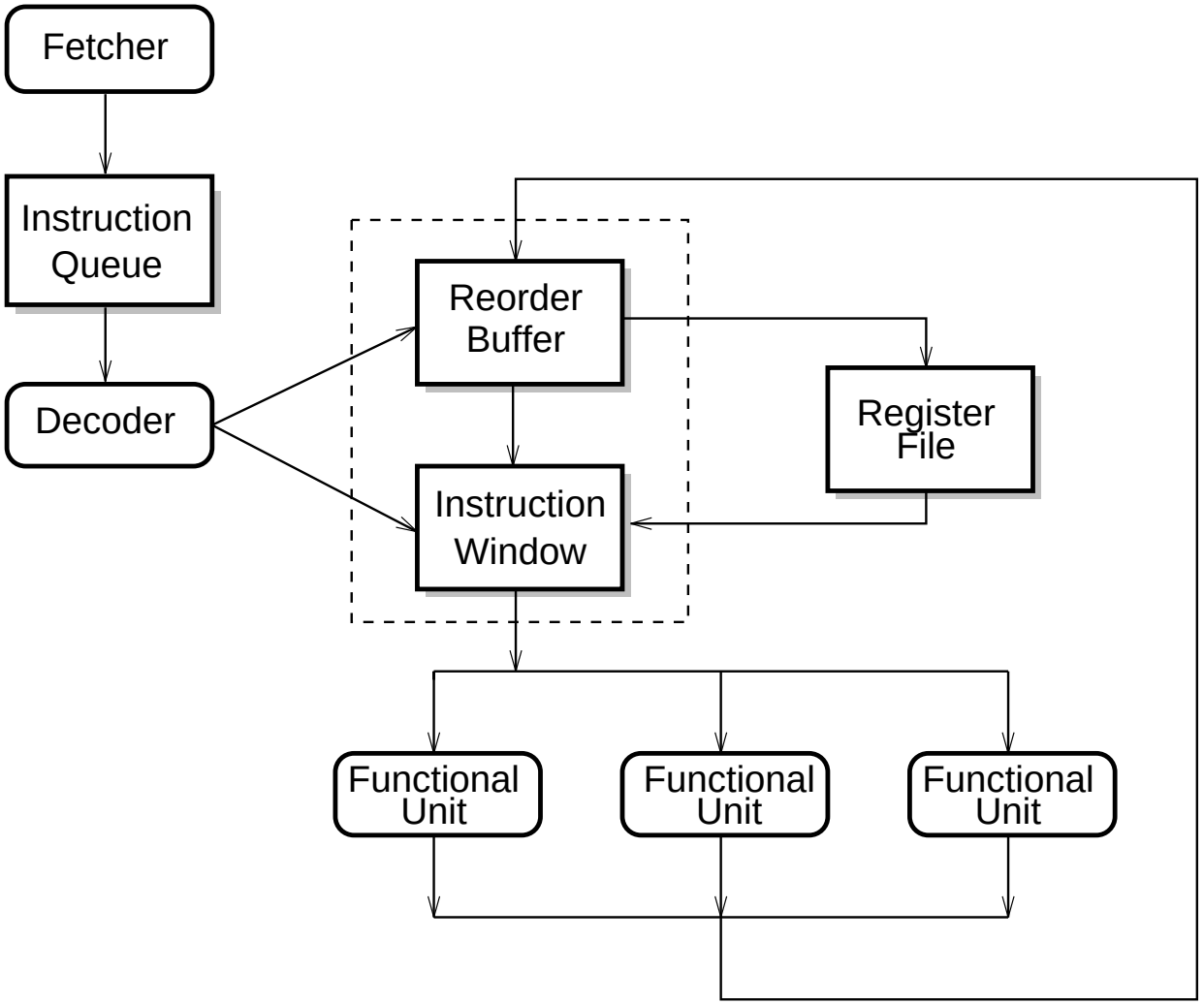
- Superscalar processors provide aggressive techniques to exploit parallelism

  - Dynamic Scheduling
  - Register Renaming
  - Speculative Execution

- Up to now, these features have remained hidden from the compiler

- Exposing these mechanisms to the compiler ⇨
  - Increase performance
  - Reduce hardware complexity

# Introduction (Cont.)




- A large percentage of values are consumed while they reside in the Reorder Buffer
- The length of their live ranges is shorter than the length of the RB.
- They are committed to the Register File, even though they are not read from it.

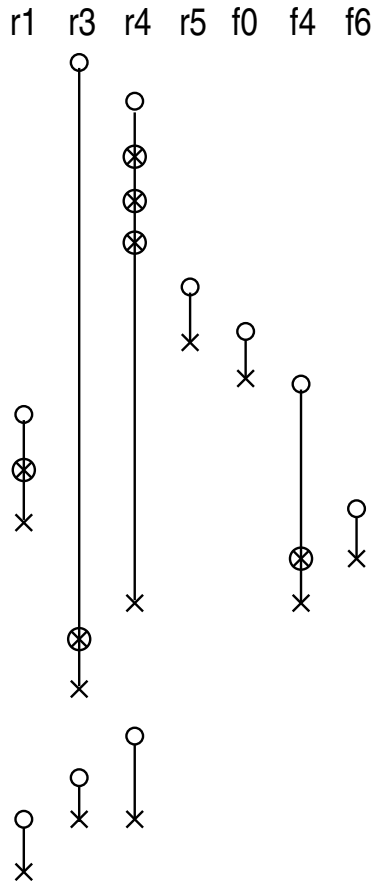
# Short-Lived Variables: A Motivating Example

```
for (I = 1; I <= N; I++) {
  X[I][1] = ((double) (I-1))
            /(double) (N-1);
}
```

**(a) C Program**

```
L5:
  lw r3,0(r6)
  slli r4,r3,#2
  add r4,r4,r3
  slli r4,r4,#3
  add r4,r7,r4
  add r5,r3,#-1
  movi2fp f0,r5
  cvti2d f4,f0
  lhi r1,(LC0>>16)&0xffff
  addui r1,r1,(LC0&0xffff)
  ld f6,0(r1)
  divd f4,f4,f6
  sd 8(r4),f4
  add r3,r3,#1
  sw 0(r6),r3
  lw r4,0(r6)
  addi r3,r0,#4
  sle r1,r4,r3
  bnez r1,L5
```

**(b) Assembly Code**




**(c) Live Ranges**

# Short-Lived Variables : Experimental Evidence

How often do the useless commits occur?

| Benchmark | Buffer Size | | |
|---|---|---|---|
| | 8 | 16 | 32 |
| Alvinn | 85.80 | 88.90 | 89.89 |
| Bubble | 93.82 | 95.93 | 97.14 |
| L8 | 89.66 | 98.81 | 98.81 |
| L14 | 91.65 | 95.93 | 97.08 |
| L8unroll | 89.54 | 98.70 | 98.70 |
| L14unroll | 91.77 | 95.87 | 97.12 |
| Linpack | 92.32 | 92.61 | 93.33 |
| Quickrand | 81.81 | 87.52 | 88.19 |
| Tomcat | 95.95 | 97.61 | 97.78 |
| Whetstone | 83.29 | 90.21 | 93.04 |
| Average | 89.56 | 94.21 | 95.11 |

# Problems to be Studied

## Problem 1 - Reduction of Useless Commits

Find a combined hardware/software mechanism to reduce the number of useless commits caused by short-lived variables

**Importance**

- Reduction of useless commits ⇒ Reduction of write ports to the register file
- Simplifies the structure of the data path and Register File
- Complexity of the Register File is O(R x W)

## Problem 2 - Allocation of Short-Lived Variables

Develop a new register allocation mechanism such that short-lived variables are not allocated to physical registers

**Importance**

- Register names used only for the allocation of long-lived variables
- Optimizations for ILP increase register pressure ⇒ Need to use names as efficiently as possible

# Reduction of Useless Commits :

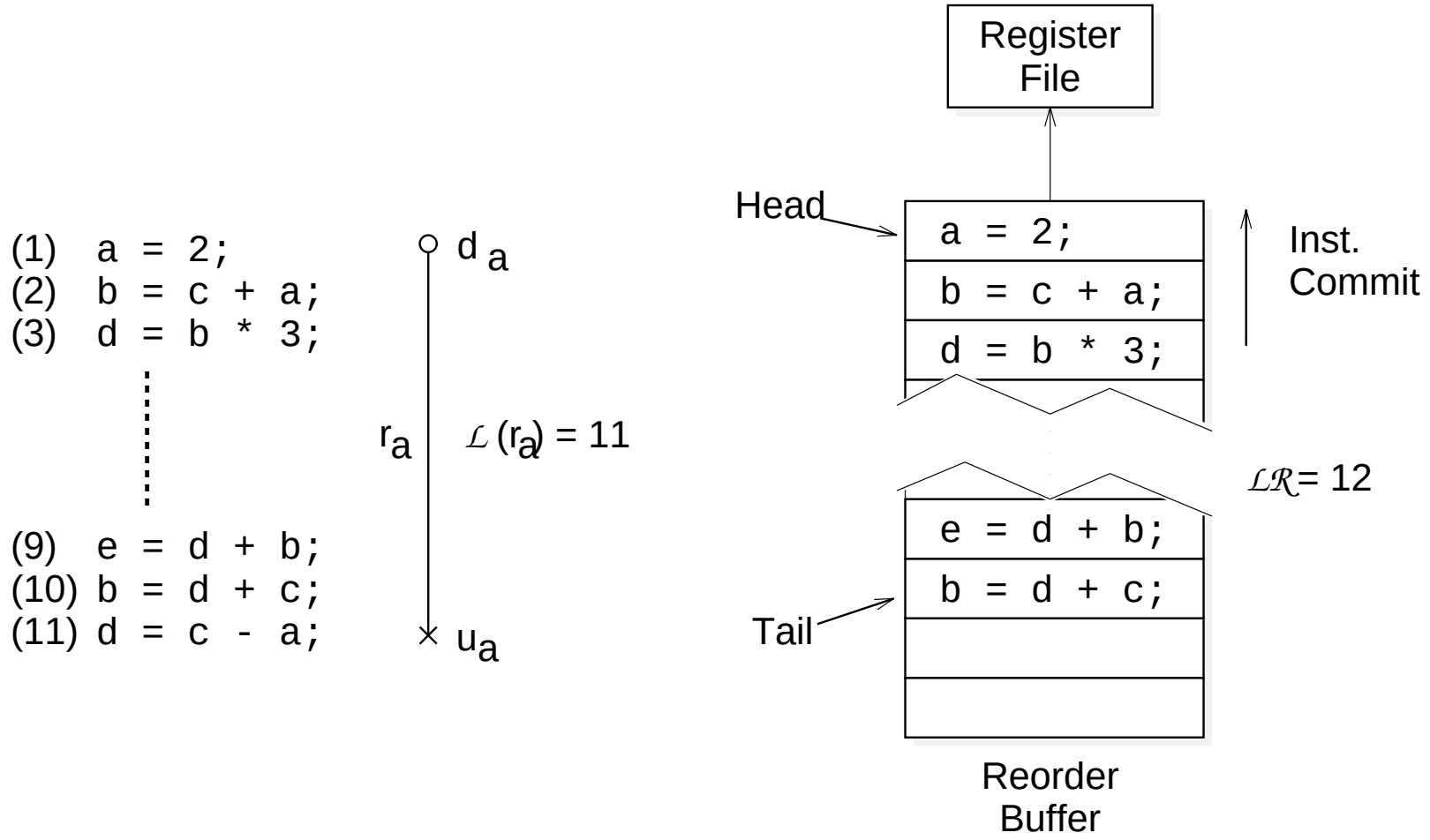
# 3 Issues

1. Mapping Short Live Ranges to Useless Commits

2. Avoiding Useless Commits in the presence of Speculative Execution

3. Providing Precise Interrupts

# Issue 1
# Mapping Short Live Ranges to Useless Commits

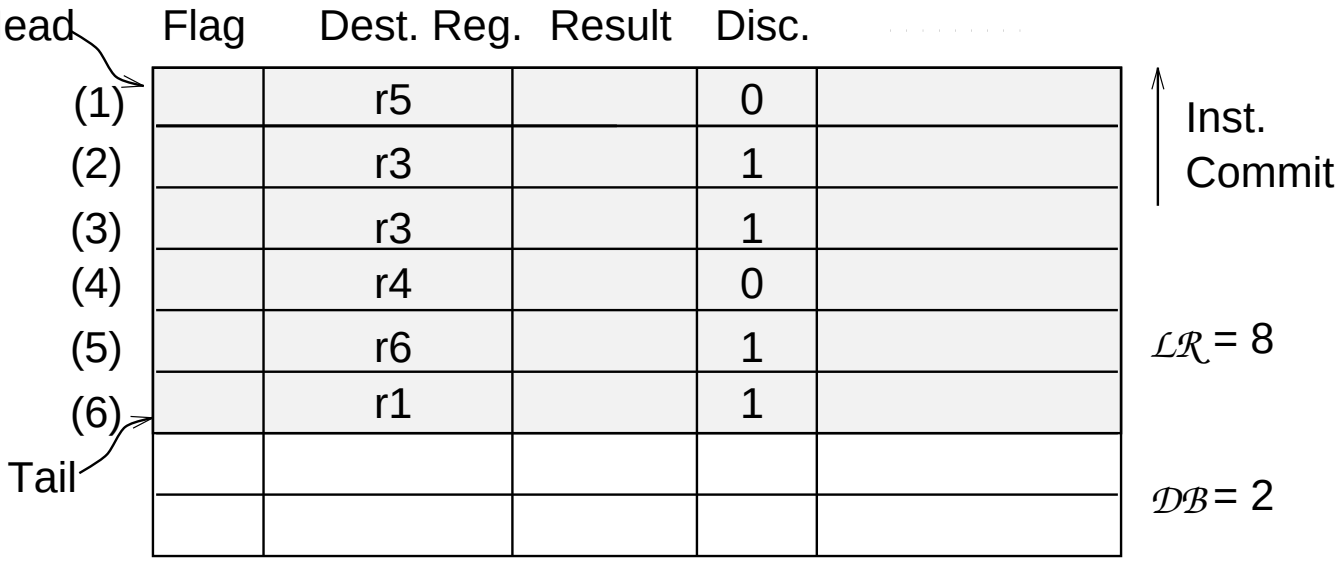



- Definition Point (1) can leave reorder buffer before last use (11) enters
- To avoid this we need to put more constraints on the commit process

# Issue 1 - Solution
# Keeping a minimum number of entries in RB

```
(1)   add r5,r0,r3
(2)   slli &r3,r4,#2
(3)   add &r3,r3,r4
(4)   slli r4,r3,#3
(5)   add &r6,r4,r3
(6)   addi &r1,r5,#8
```

(a) Assembly code

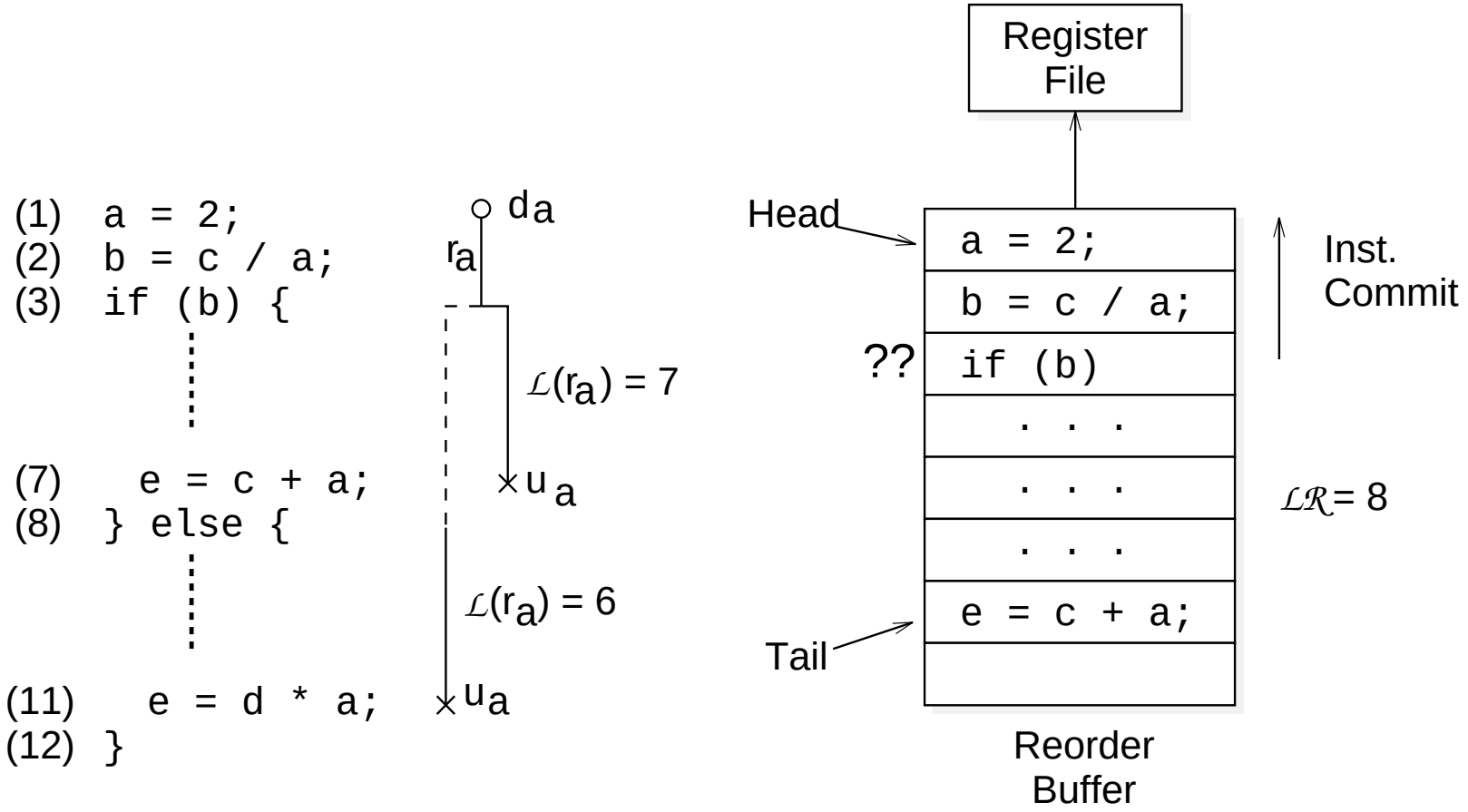



(b) Reorder buffer representation

- An instruction cannot be committed unless there is a minimum number of instructions in the reorder buffer

- Special care has to be taken in order not to stall the decoder:
  Minimum number of instructions = LR - DB

# Issue 2
# Dealing with Instruction Speculation

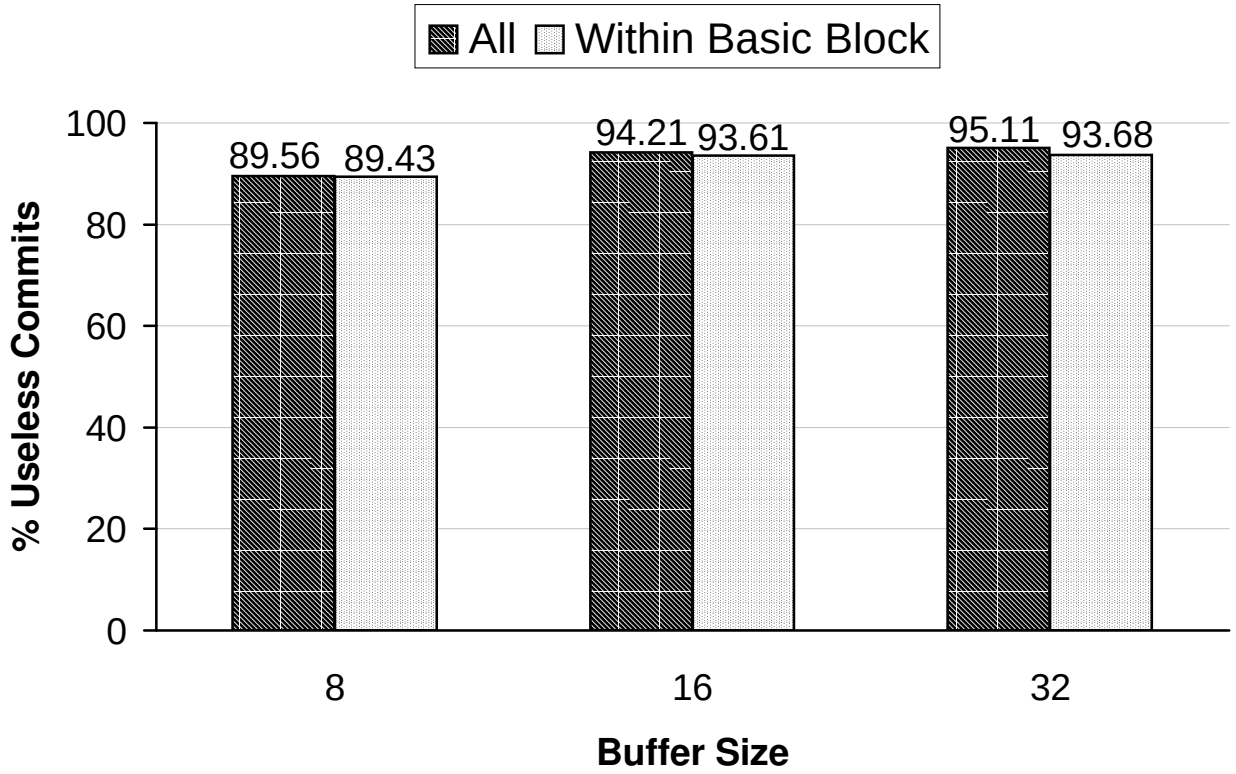



- Definition point (1) cannot be discarded when last use (7) is present because last use could be speculatively executing
- Impose more constraints on the commit process ??

# Issue 2 - Solution

**Consider only Short-Lived variables that do NOT cross basic block boundaries**

**Experimental Evidence:**

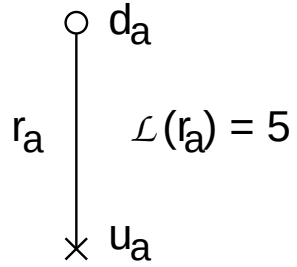


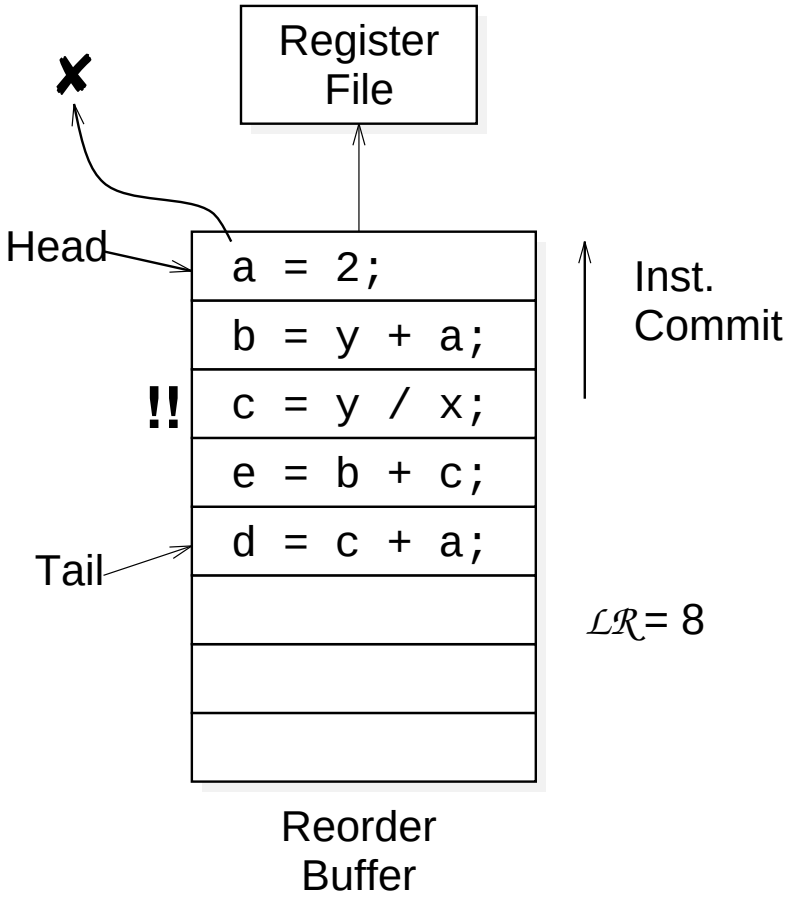

**Advantage:**

- No hardware modifications required

# Issue 3
# Providing Precise Interrupts

```
(1)   a = 2;
(2)   b = y + a;
(3)   c = y / x;
(4)   e = b + c;
(5)   d = c + a;
```




- If we discard value of "a" in instruction (1),

  where will instruction (5) get the value of "a"
  if instruction (3) causes an interrupt?

# Allocation of Short-Lived Variables

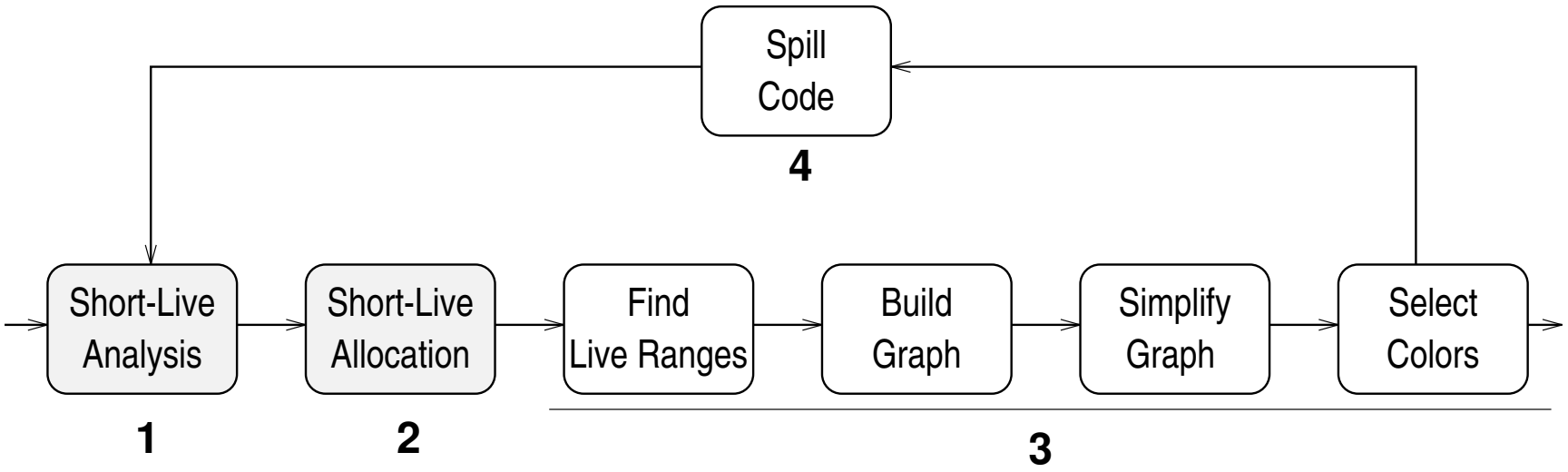
**Purpose:**

Allocate the Short-Lived variables into the space provided by the renaming mechanism supported by the reorder buffer

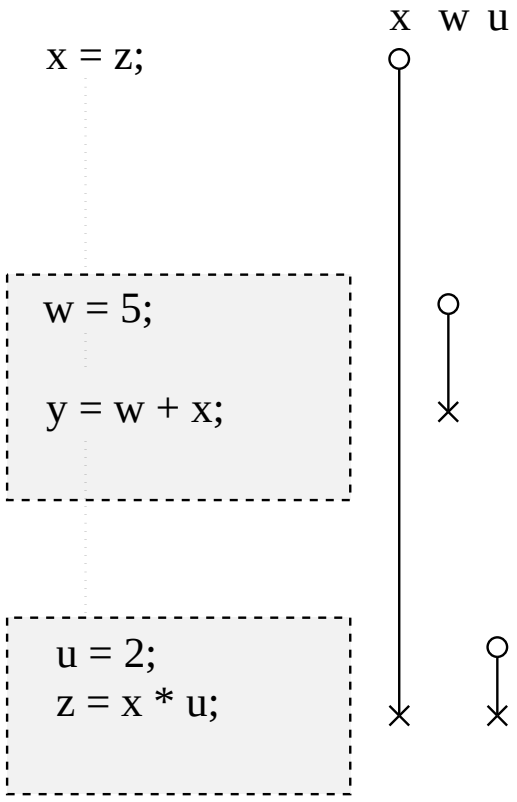
- Each location provided by the reorder buffer is treated as a "Symbolic Register"

- The actual location for a "Symbolic Register" is dynamically assigned

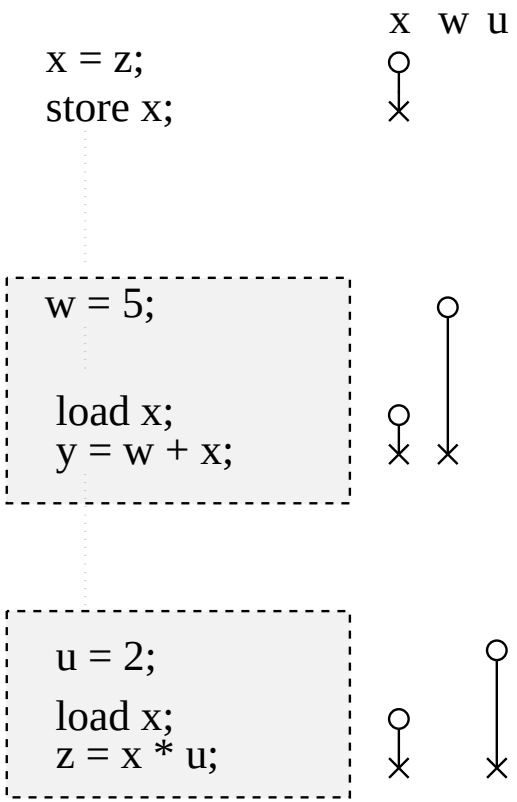
**Allocation in 4 steps:**



# Allocation of Short-Lived Variables: Effect of the introduction of spill code




(a) Before spilling x

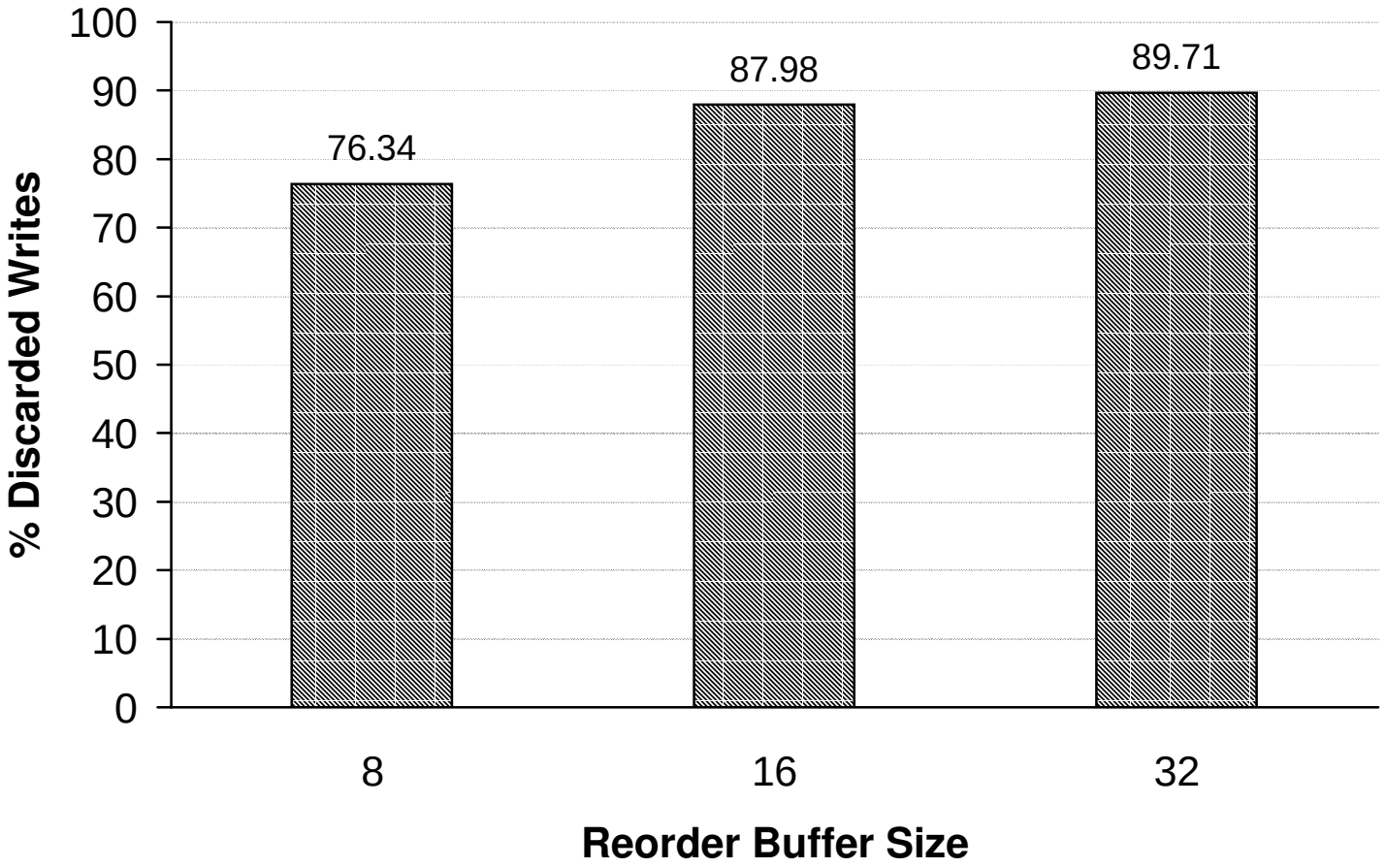



(a) After spilling x

- Long-Lived variables that are spilled become Short-Lived
- Introduction of spill code may transform some Short-Lived variables into Long-Lived

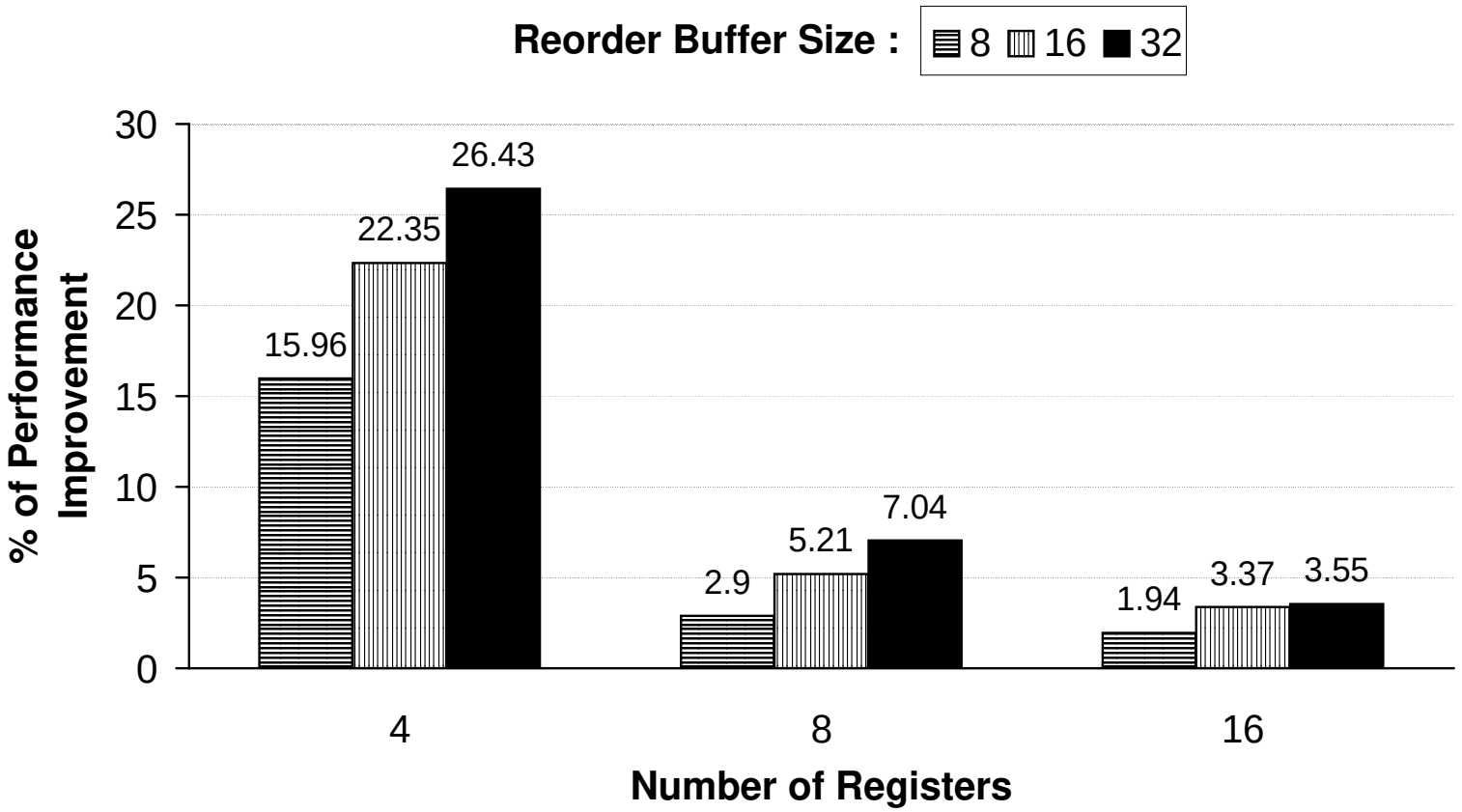
# Experimental Results

## 1. Reduction of Useless Commits



# Experimental Results

## 2. Allocation of Short-Lived Variables



# Related Work

1. A. Pleszkun and G. Sohi.
"The performance potential of multiple functional unit processors"
ISCA 15, 1988

2. M. Franklin and G. Sohi
"Register traffic analysis for streamlining inter-operation
communication in fine-grain parallel processors"
MICRO 25, 1992

3. Pleszkun et al.
"WISQ: A restartable architecture using queues"
ISCA 14, 1987

4. Hoogerbrugge and Corporaal
"Register file port requirements of transport triggered architectures"
MICRO 27, 1994

# Conclusions

- Our combined software/hardware scheme is able to discard almost 90% of the writes to the register file

- This mechanism can be used to reduce the number of write ports to the register file without affecting performance

- The modified allocation scheme can be used to reduce register pressure and the amount of spill code required. Up to 26% performance improvement can be obtained.