SpecInt Branch and Register Data

David Morano

Northeastern University

1. Introduction

This document serves to present some branch data gathered from the SpecInt-2000 and SpecInt-95 benchmark suites. Data was accumulated over ten benchmarks programs. Seven programs were taken from the SpecInt-2000 suite and three programs were taken from the SpecInt-95 suite. The following SpecInt-2000 programs were used for gathering data :

- bzip2
- crafty
- gcc
- gzip
- mcf
- parser
- vortex

The following SpecInt-95 programs were also used :

- go
- ijpeg
- compress

All programs were compiled for the MIPS-1 ISA and used the native SGI compiler under IRIX 6.4 with the standard optimization turned on. All data collected was on executing 500 million instructions after skipping the first 100 million instructions. All microarchitecture structures and data accumulation apparatus were allowed to warm up during the first 100 million instructions.

2. Branch Path Lengths

The graphs in this section show percent density functions and percent distributions for branch path lengths in each of the five SpecInt programs that we have characterized. Figures 1 and 2 give the probability density and the probability distribution (respectively) for branch paths versus branch path lengths. The mean branch path length is **9.4** and the standard deviation is **10.4**.

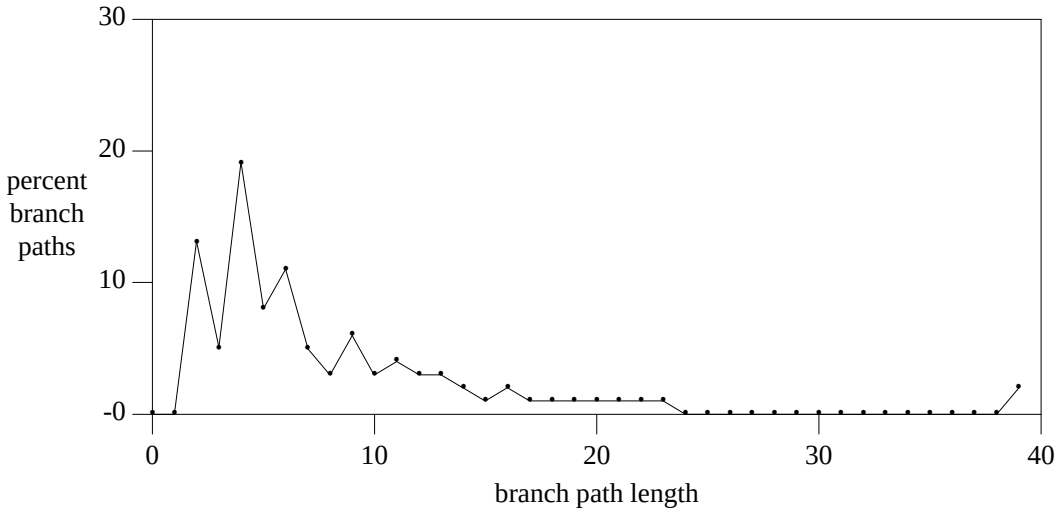

Figure 1. Cumulative Branch Path Length Density

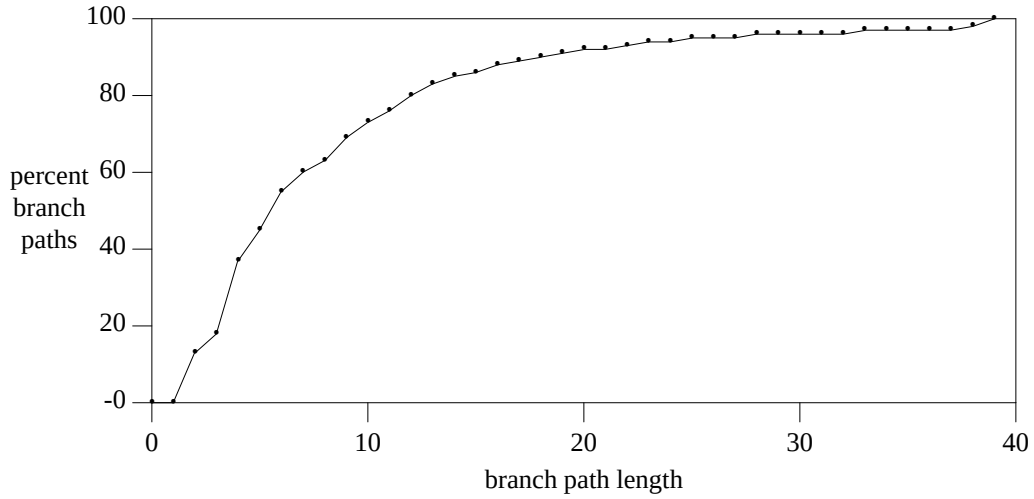

Figure 2. Cumulative Branch Path Length Distribution

3. Conditional Branch Traget Distances

The graphs in this section show probability density functions and probability distributions for conditional branch target distances for the mean of the benchmark programs. Different densisties and distributions are shown for forward conditional branches and backward conditional branches. Although backward conditional branches are most usually associated with program loops, this does not have to be the case. Loops can, of course, be effected with forward conditional branches and an unconditional branch, with or without possible backward conditional branches in the body of the loop. Loops may there for be indicated by strong peaks in either forward or backward conditional branches. Figures 3 and 4 show the density and distribution for the target distances for forward conditional branches. The mean forward conditional branch

target distance is **31.3** and the standard deviation is **91.6**.

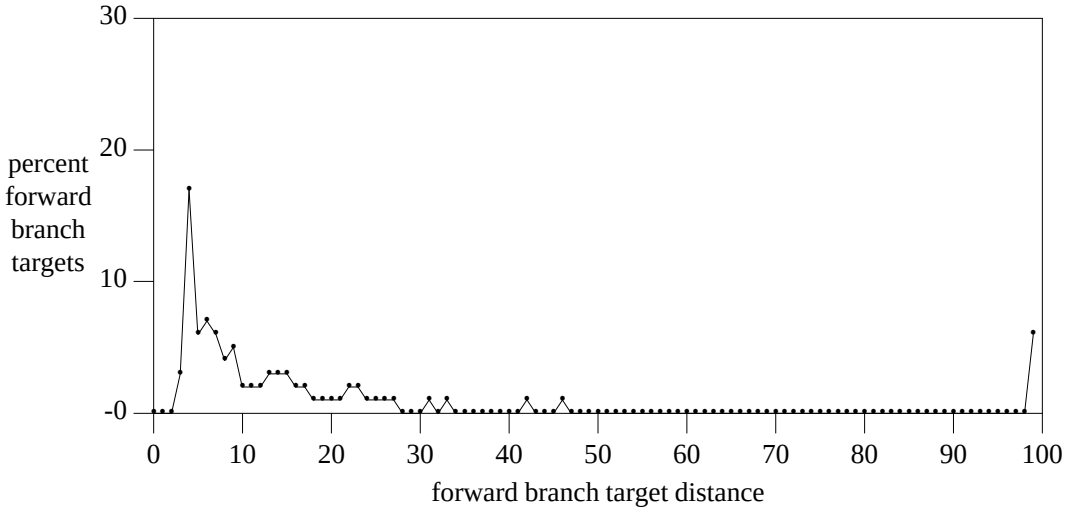

Figure 3. Cumulative Forward Conditional Branch Target Distance Density

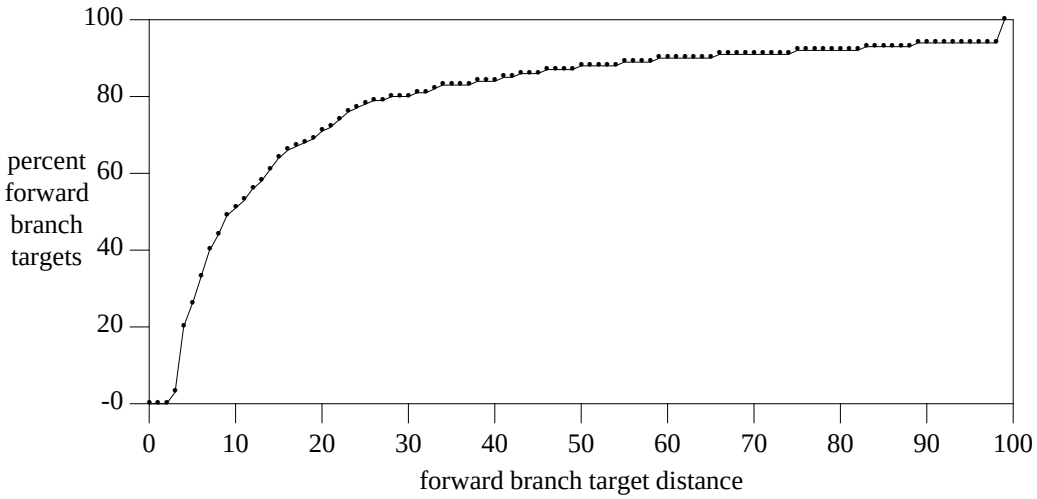

Figure 4. Cumulative Forward Conditional Branch Target Distance Distribution

Figures 5 and 6 show the density and distribution target distances for backward conditional branches. The mean conditional branch target distance is **41.8** and the standard deviation is **105.1**.

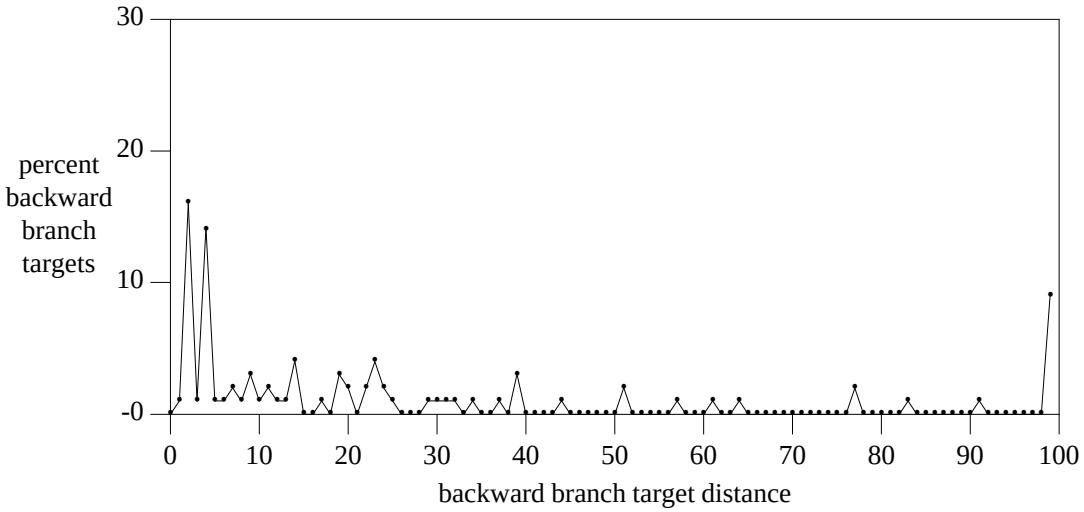

Figure 5. Cumulative Conditional Backward Branch Target Distance Density

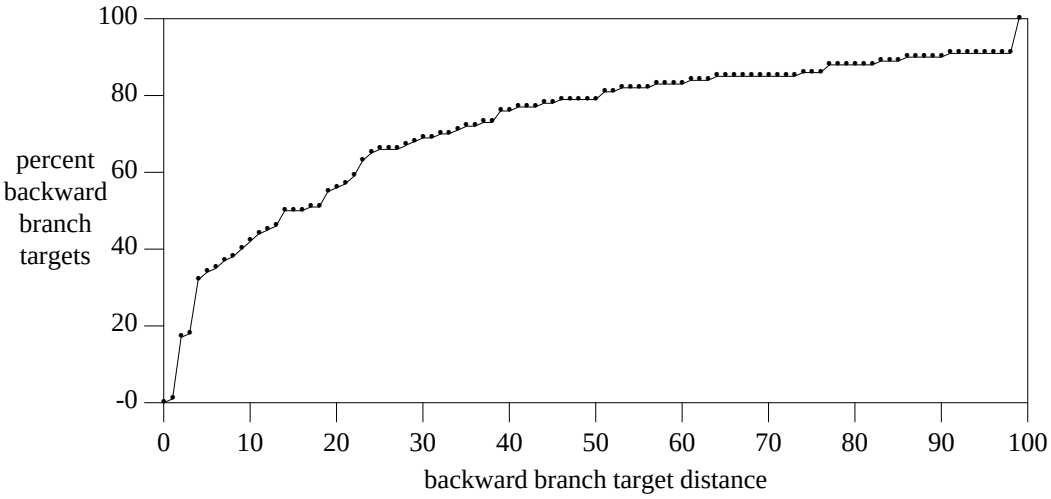

Figure 6. Cumulative Conditional Backward Branch Target Distance Distribution

4. Simple Single-Sided Hammock Branch Target Distances

The graphs in this section show a probability density function and distribution for simple single-sided Hammock conditional branch target distances for the mean of all benchmark programs investigated. Simple single-sided Hammock conditional branches have only forward going targets. A minimal simple Hammock conditional branch with a backward going target would have to be similar to a double sided Hammock branch. Figures 7 and 8 show the density and distribution respectively. The mean S-S Hammock branch target distance is **5.6** and the standard deviation is **3.6**.

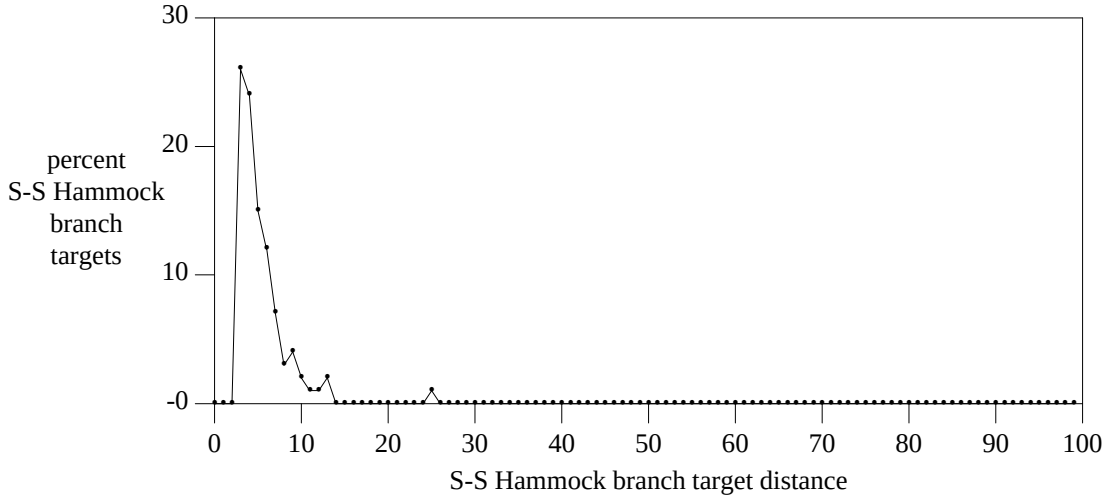

Figure 7. Cumulative SS-Hammock Target Distance Density

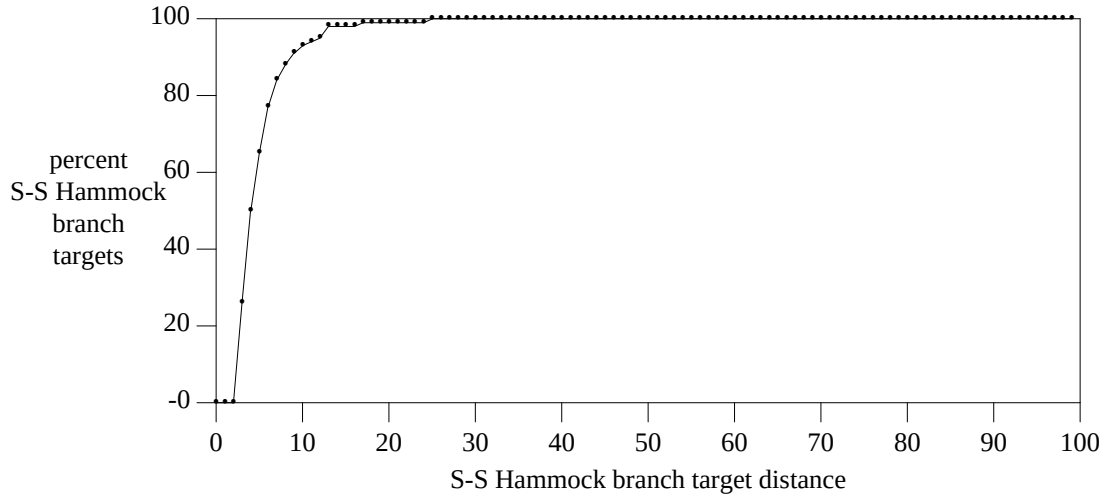

Figure 8. Cumulative SS-Hammock Target Distance Distribution

5. Register Usage Statistics

In this section, we show register statistics for the register lifetimes as well as for the def-use intervals. Further, information on which registers are used most often by the code is also provided. For the register lifetimes and def-use intervals, both a density and a distribution over the associated intervals is shown over all benchmarks programs. For the register read and write usage by the programs, densities on register addresses are provided (one for reads and the other for writes).

Figures 9 and 10 show the density and distribution respectively for the register lifetimes over the lifetime interval as measured in instructions. The mean register lifetime (in instructions) is **25.4** and the standard

deviation is **87.8**.

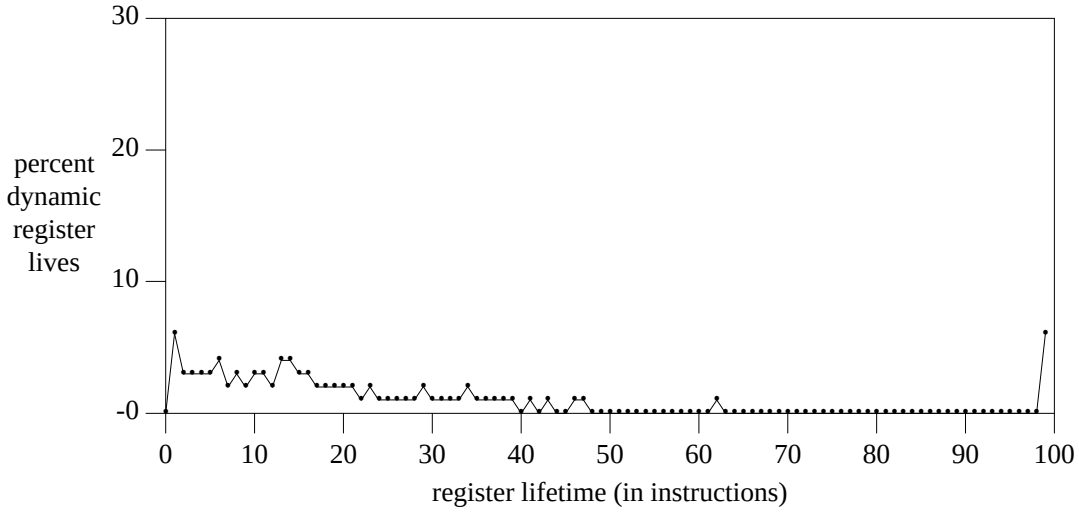

Figure 9. Cumulative Register Lifetime Density

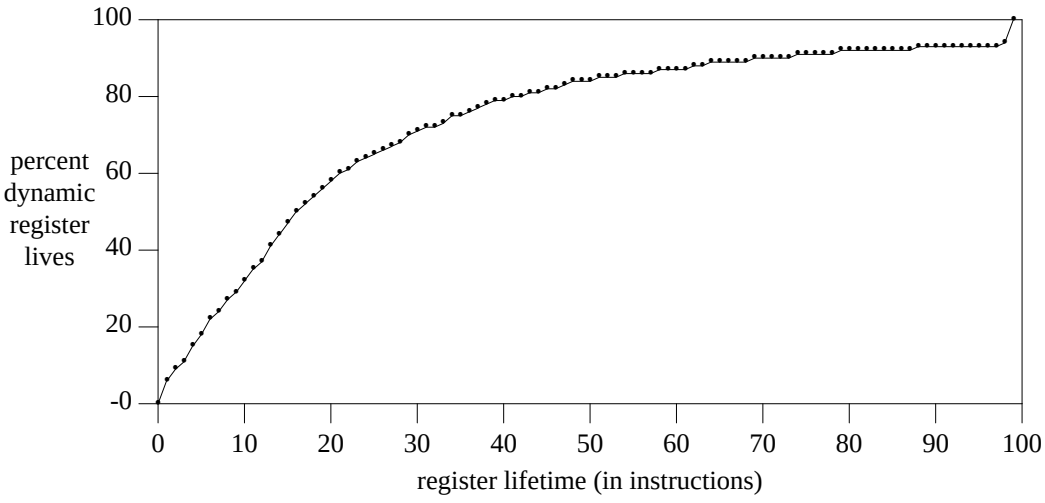

Figure 10. Cumulative Register Lifetime Distribution

Figures 11 and 12 show the density and distribution respectively for the register def-use intervals as measured in instructions. Registers $r0$ and $r28$ are exluded from this def-use data since they are only read-only or only written once during the entire execution of a program. The mean register def-use interval (in instructions) is **79.3** and the standard deviation is **326.3**.

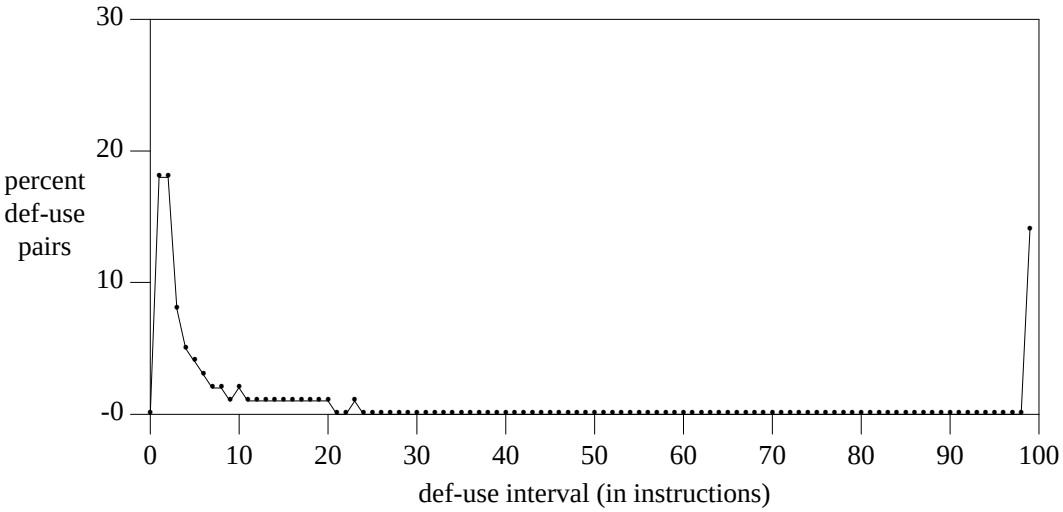

Figure 11. Cumulative Register Def-Use Density

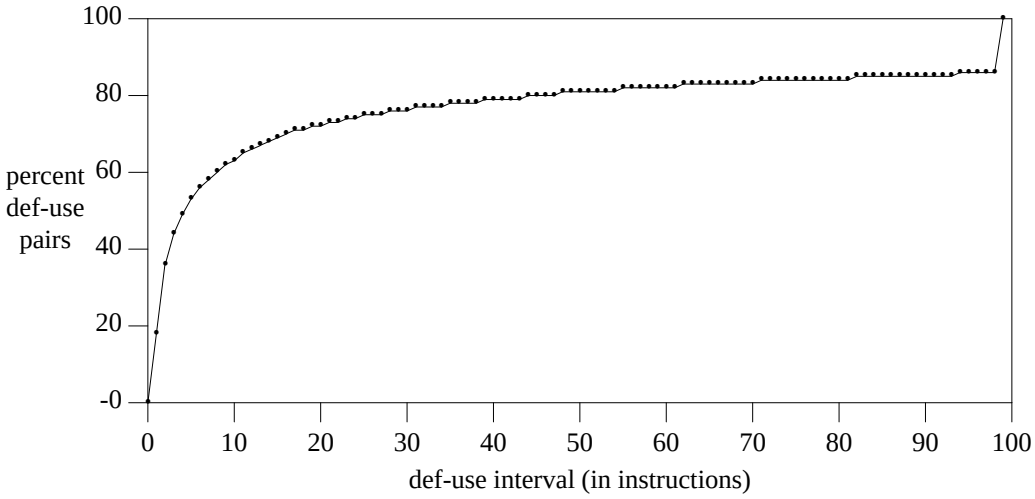

Figure 12. Cumulative Register Def-Use Distribution

Figures 13 and 14 show the densities over register addresses for register reads and register writes respectively. On the MIPS ISA, registers *r*0 through *r*31 are integer registers. Registers *r*64 through *r*85 are floating point registers. Other rgisters (which do not get much use) are various status or control registers (usually for floating pointer operations). It should be noted that register *r*0 is hardwired to the value zero and is a read-only register. For reference, register *r*29 is the stack pointer. Further, register *r*28 is the pointer to the global offset table and is only written once at the start of the program. The data shown is only for the execution of 500 million instructions after skipping the first 100 million, so the single write of register *r*28 would not show up in this data anyway. Registers *r*26 and *r*27 are never written or read by any of the benchmark programs investigated (the ten that we executed).

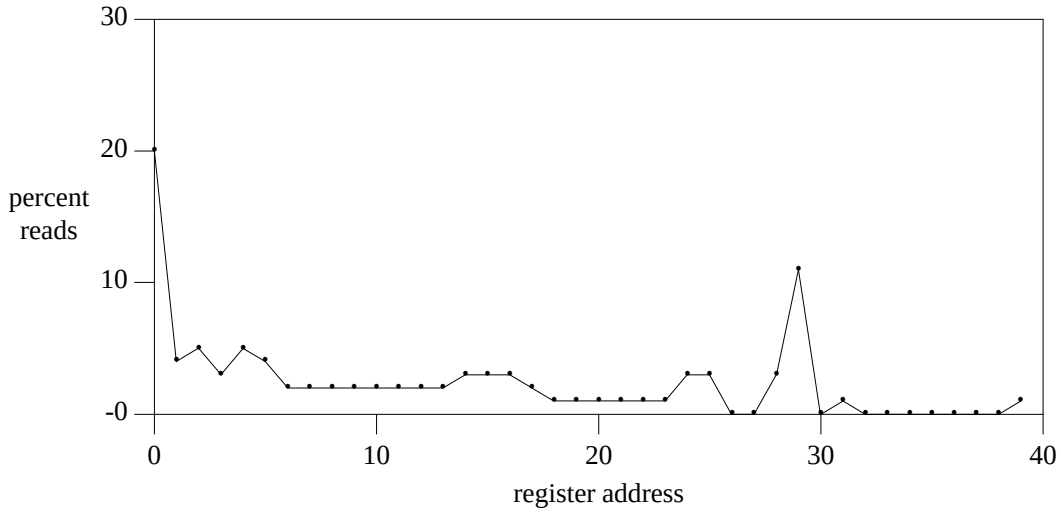

Figure 13. Cumulative Register Read Density Over Register Address

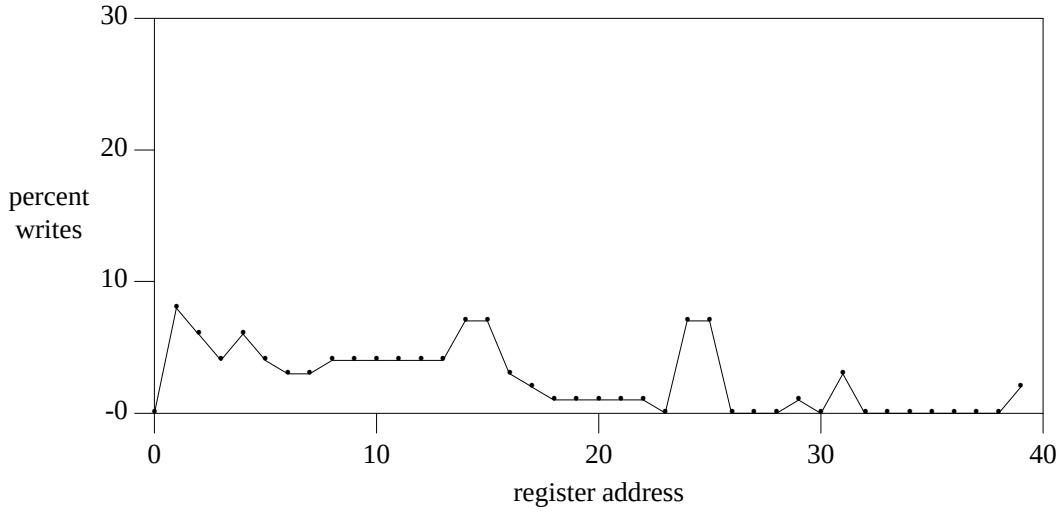

Figure 14. Cumulative Register Write Density Over Register Address

CONTENTS

LIST OF FIGURES