CS810: Advanced Computer Architecture

Assignment 4

10 October 2023

Shashank P (200010048), Tabish Khalid Halim (200020049)

The objective of this assignment was to count the number of **Inconsequent Instructions** present during a program's run-time.

1 Introduction

Inconsequent Instructions, in general, need not be executed for the program to execute. In this case, we counted the following types of **Inconsequent Instructions**:

1. **Register Inconsequent**: Write after Write to the same register without a read in between to that register and all the registers that this instruction depends on recursively.
2. **Memory Inconsequent**: Write after Write to the same memory address without reading to that address in between, and all the memory addresses that this instruction depends on recursively.
3. **Branch Inconsequent**: A branch instruction that is predicted correctly and all the instructions that this branch instruction depends on
4. **Total Inconsequent**: Total count of all possible Inconsequent Instructions across all types

Among these, there are **Register Root**, **Memory Root** and **Branch Root** Inconsequentials. These are the root instructions using which other inconsequential instructions can be removed.

2 Experimental Setup

In our experiment, we used **Intel Pin** tool with the given **5 SPEC CPU 2017** benchmarks. The branch predictor accuracy was manually (arbitrarily) set to **80%**. Since each benchmark had dynamic instructions on the order of **10's of Billions**, the following sampling strategy was chosen

- **Range Size:** Number of Instructions to be kept in the buffer for applying the counting algorithm. $5 * 10^4$ instructions
- **Step Size:** Number of instructions to pass before resetting the buffer. 10^{10} instructions

3 Approach

The approach to get the count of inconsequent instructions is as follows:

1. One pass through the buffer array and fetch all the **Root Inconsequent** instructions and update the root count.
2. Remove the Inconsequent Instructions obtained from the previous step.
3. Repeat the first step but update the **Non-Root Inconsequent** counter
4. This goes on until **No Instructions are Left** or **There are no Inconsequent Instructions to remove**

The above steps can be extrapolated for all types of counters.
A bash script copies the **Intel Pin** file, builds the tool, and runs it on all the benchmarks. The output file is then analyzed. It generates **.count** files under **data** directory containing the following information:

1. **All 7 Counter Values**: 2 Register, 2 Memory, 2 Branch and One General
2. **Top Register Inconsequent**: Top 10 dynamic instruction **trace** that caused the most **Root Register Inconsequents**
3. **Top Memory Inconsequent**: Top 10 dynamic instruction **trace** that caused the most **Root Memory Inconsequents**

During experimentation, we verified the trends in the count after every step of sampling, and there were no unusual variations in counts due to sampling, i.e., the inconsequentiality of instructions was uniform throughout the sampling process. This can be seen in the **.err** files under the **err** directory among the submitted files, which contains per sampling counts.

4 Plots

To visualize the data, we have plotted the following graphs. Each graph consists of a specific type of inconsequent with both its sub-types. This is done for all **5** benchmarks

4.1 Register Inconsequential Instructions

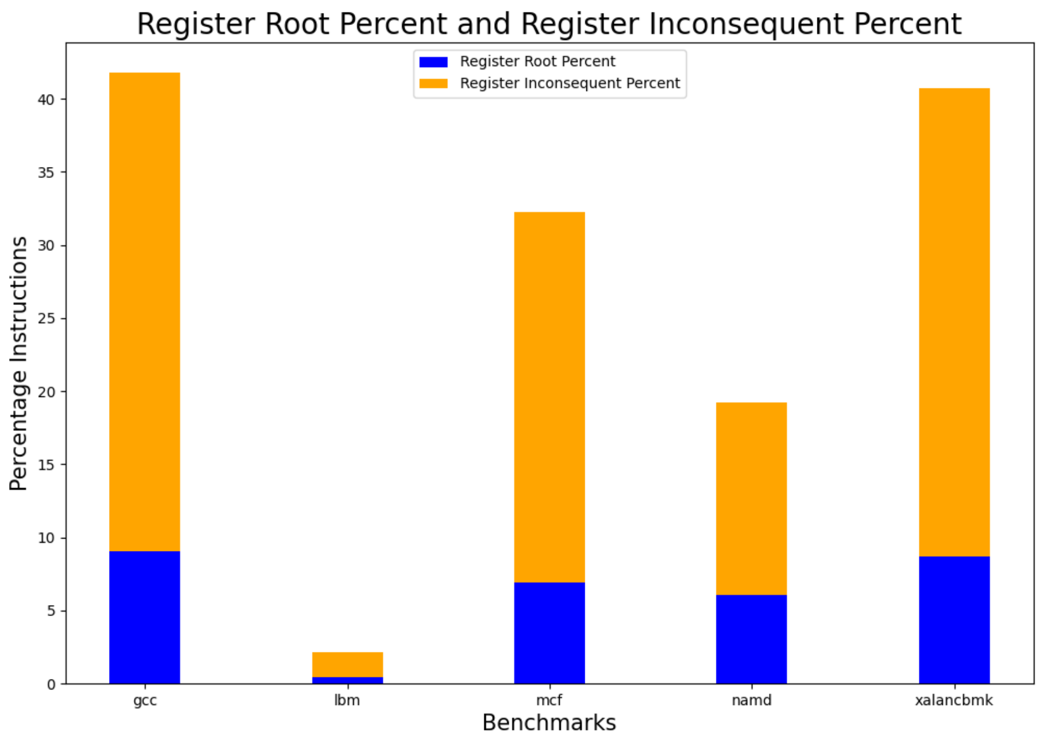

Figure 1: Register Inconsequential Instructions

- From the graph, we can see that around 10% of the instructions are root register inconsequential while 30% of them are dependency root inconsequential
- The **lbm** benchmark has the least amount of register root dependency while **gcc** has the highest. This may be due to very high memory accesses in **lbm** compared to **gcc**
- The dependency register inconsequential are very high compared to that of root register inconsequentials

4.2 Memory Inconsequential Instructions

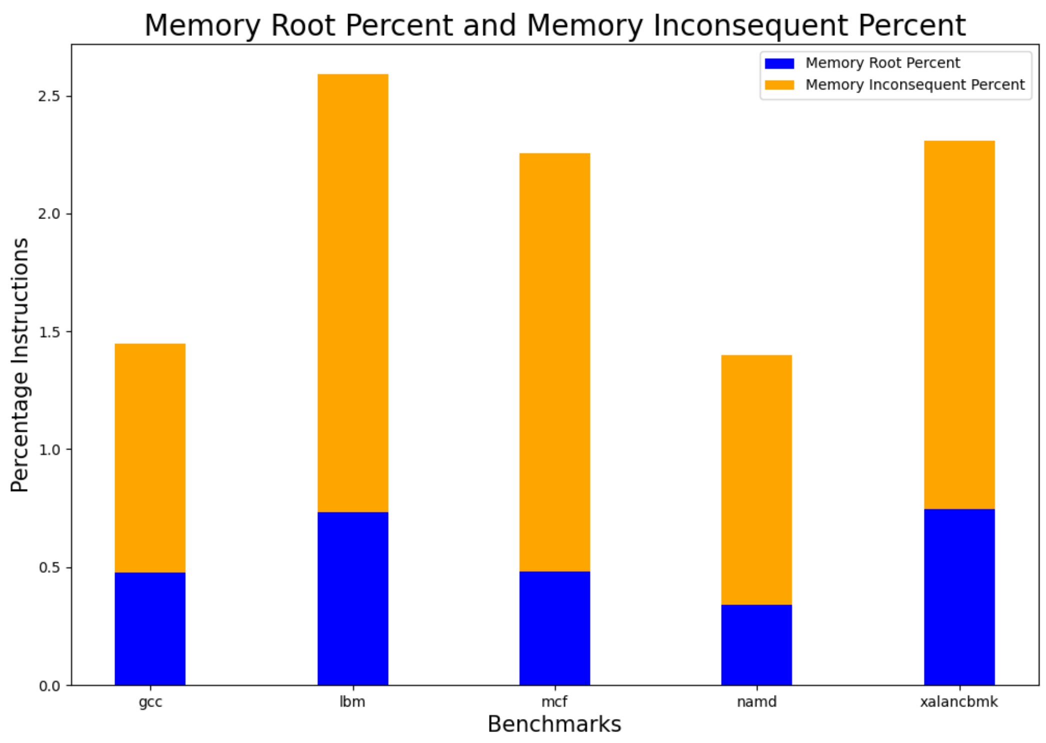

Figure 2: Memory Inconsequential Instructions

- The memory inconsequential instructions are around 2% of the total while the root memory inconsequential are around **0.5%**
- Comparitively **lbm** has the highest memory inconsequential instructions; this might be due to repeated memory accesses
- Comparatively, there is only very little difference between root and dependency memory inconsequential percentages per benchmarks
- The number of memory inconsequential is much less due to the fact that the address space is very big compared to that of a small number of registers

4.3 Branch Inconsequential Instructions

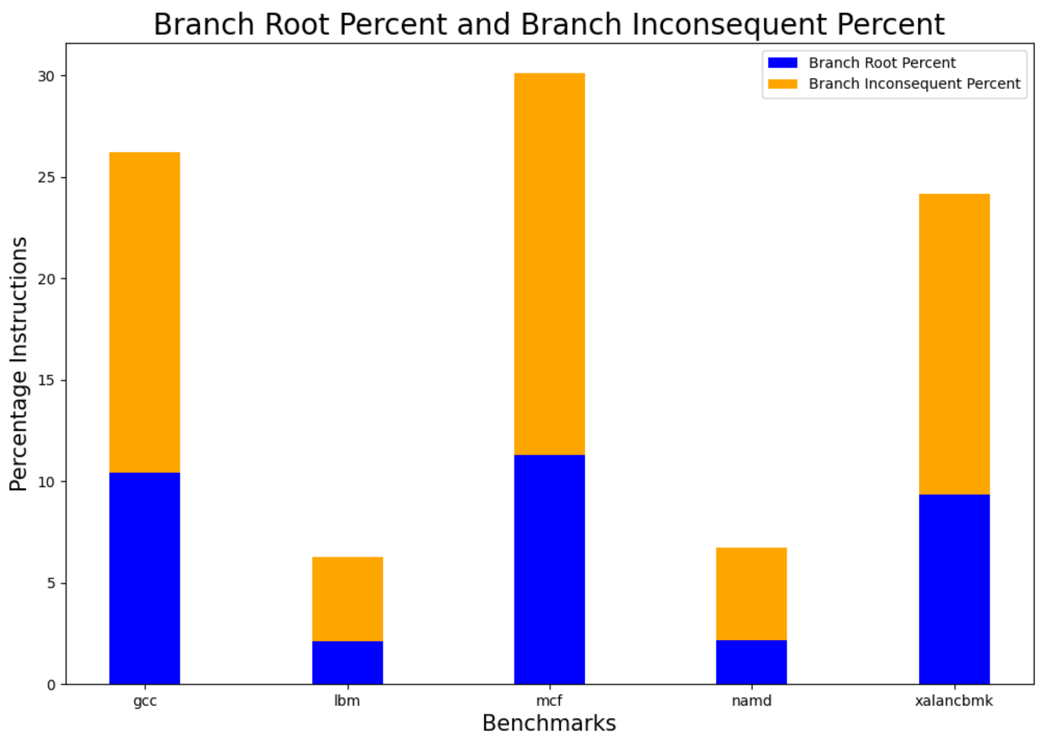

Figure 3: Graph of Branch Inconsequential Instructions

- Number of root branch inconsequential are around **7%** while the dependency branch instructions are around **20%**
- The number of root branches mostly depends on the accuracy of the predictor; the better the predictor, the more inconsequential instructions are present
- **mcf** has the highest branch inconsequential probably due to more predictable branches. **lbm** has the lowest due to the number of branching instructions

4.4 Inconsequential Instruction Counts

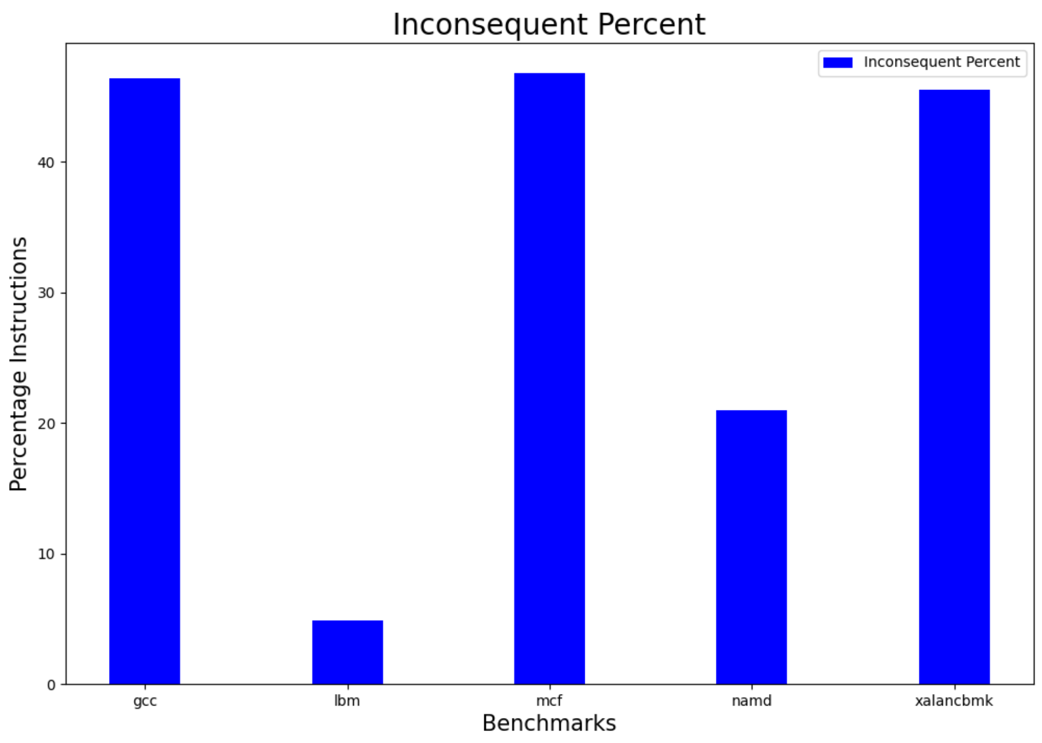

Figure 4: Inconsequential Instruction Counts

- In total there are around 30% of inconsequential instructions in the benchmarks
- **gcc, mcf and xalancbmk** have very high number of inconsequential instructions while **lbm** has the lowest

Metric	GCC	MCF	LBM	XALANCBMK	NAMD
Register root inconsequential instruction	9.0226%	6.92961%	0.414198%	8.69728%	6.05474%
Register inconsequential instruction	32.74%	25.3297%	1.72825%	32.0347%	13.1989%
Memory root inconsequential instruction	0.478686%	0.482686%	0.73312%	0.74653%	0.342014%
Memory inconsequential instruction	0.971372%	1.77274%	1.85662%	1.56049%	1.05604%
Branch root inconsequential instruction	10.3924%	11.2771%	2.08899%	9.32102%	2.15896%
Branch inconsequential instruction	15.8296%	18.8201%	4.15271%	14.8495%	4.56343%
Inconsequential Instruction	46.4002%	46.7652%	4.83219%	45.4971%	20.9922%

Table 1: Table containing Inconsequence Count Percentages per Benchmark

5 Tabular Representation

- Comparatively, there is more **register inconsequential** than branch inconsequential, and **branch** is more than **memory inconsequential**. This might be due to the spread of memory accesses compared to registers and good branch predictor accuracy.
- Most of the contribution to dependency inconsequential (memory, branch and general) is due to register dependencies
- From the table, it is evident that LBM benchmark has the lowest percent of inconsequential instructions across all the metrics, we can infer that this can be because of the lbm benchmark in nature is CPU intensive with large number of memory accesses which may lead to more efficient use of instructions.
- The number of branch inconsequential depend both on register dependency as well as branch prediction accuracy.

6 Instruction Analysis

We extracted a sub-trace of instructions that were part of root inconsequence instructions. The **Top 10** most occurring patterns were then filtered and written to the output file. A few of them for root register and memory are shown below.

6.1 Root Register Inconsequence Instructions

```
23720 -> movzx eax, word ptr [r8]  ->  Read: [r8, ]  Write: [eax, ] Read1: 1
    Read2: 0 Write1: 0
23721 -> cmp word ptr [rdx], ax  ->  Read: [rdx, ax, ]  Write: [rflags, ]
    Read1: 1 Read2: 0 Write1: 0
23722 -> jz 0x7aa0b0  ->  Read: [rip, rflags, ]  Write: [rip, ] Read1: 0 Read2
    : 0 Write1: 0
23716 -> add rdx, 0x2  ->  Read: [rdx, ]  Write: [rdx, rflags, ] Read1: 0
    Read2: 0 Write1: 0
23717 -> cmp rsi, rdx  ->  Read: [rsi, rdx, ]  Write: [rflags, ] Read1: 0
    Read2: 0 Write1: 0
23718 -> jz 0x7aa0e0  ->  Read: [rip, rflags, ]  Write: [rip, ] Read1: 0 Read2
    : 0 Write1: 0
23719 -> add r8, 0x2  ->  Read: [r8, ]  Write: [r8, rflags, ] Read1: 0 Read2:
    0 Write1: 0
23720 -> movzx eax, word ptr [r8]  ->  Read: [r8, ]  Write: [eax, ] Read1: 1
    Read2: 0 Write1: 0
```

```
121762 -> jz 0x8209d0  ->  Read: [rip, rflags, ]  Write: [rip, ] Read1: 0
    Read2: 0 Write1: 0
121764 -> ret   ->  Read: [rsp, ]  Write: [rip, rsp, ] Read1: 1 Read2: 0
    Write1: 0
```

```
11507 -> jz 0x8209b8  ->  Read: [rip, rflags, ]  Write: [rip, ] Read1: 0 Read2
    : 0 Write1: 0
11509 -> ret   ->  Read: [rsp, ]  Write: [rip, rsp, ] Read1: 1 Read2: 0 Write1
    : 0
```

1. Among the **Top 3** register root inconsequential, the **inconsequential instruction only** occurs due to **branches** in the vicinity. If there were no branches, the compiler could easily remove them statically.

2. In the above cases, if the outcomes of the branches may be different, there may not have been inconsequential

3. The registers that cause inconsequentiality of instructions are registers that modify the program's control flow and register flags. It might be that the the flags were set and then replaced without using.

6.2 Root Memory Inconsequence Instructions

```
Memory Address: 35745280
68064 -> mov word ptr [r9+rax*2], r13w  ->  Read: [r9, rax, r13w, ]  Write: []
     Read1: 0 Read2: 0 Write1: 1
68065 -> xor eax, eax  ->  Read: [eax, eax, ]  Write: [eax, rflags, ] Read1: 0
     Read2: 0 Write1: 0
68066 -> vucomisd xmm3, xmm2  ->  Read: [xmm3, xmm2, ]  Write: [rflags, ]
    Read1: 0 Read2: 0 Write1: 0
68067 -> movzx r13d, r8w  ->  Read: [r8w, ]  Write: [r13d, ] Read1: 0 Read2: 0
     Write1: 0
68068 -> setnbe al  ->  Read: [rflags, ]  Write: [al, ] Read1: 0 Read2: 0
    Write1: 0
68069 -> lea edx, ptr [rax+rsi*1]  ->  Read: [rax, rsi, ]  Write: [edx, ]
    Read1: 0 Read2: 0 Write1: 0
68070 -> movsxd rdx, edx  ->  Read: [edx, ]  Write: [rdx, ] Read1: 0 Read2: 0
    Write1: 0
68071 -> mov word ptr [r9+rdx*2], r10w  ->  Read: [r9, rdx, r10w, ]  Write: []
     Read1: 0 Read2: 0 Write1: 1
```

```
Memory Address: 35745280
26989 -> mov word ptr [r11+rax*2], r10w  ->  Read: [r11, rax, r10w, ]  Write:
    [] Read1: 0 Read2: 0 Write1: 1
26990 -> xor eax, eax  ->  Read: [eax, eax, ]  Write: [eax, rflags, ] Read1: 0
     Read2: 0 Write1: 0
26991 -> vucomisd xmm10, xmm7  ->  Read: [xmm10, xmm7, ]  Write: [rflags, ]
    Read1: 0 Read2: 0 Write1: 0
26992 -> movzx r10d, r8w  ->  Read: [r8w, ]  Write: [r10d, ] Read1: 0 Read2: 0
     Write1: 0
26993 -> setnbe al  ->  Read: [rflags, ]  Write: [al, ] Read1: 0 Read2: 0
    Write1: 0
26994 -> lea edx, ptr [rax+rcx*1]  ->  Read: [rax, rcx, ]  Write: [edx, ]
    Read1: 0 Read2: 0 Write1: 0
26995 -> movsxd rdx, edx  ->  Read: [edx, ]  Write: [rdx, ] Read1: 0 Read2: 0
    Write1: 0
26996 -> mov word ptr [r11+rdx*2], r9w  ->  Read: [r11, rdx, r9w, ]  Write: []
     Read1: 0 Read2: 0 Write1: 1
```

- Unlike register instructions, memory instructions become inconsequential, mostly not due to branch, as can be seen in the above examples
- Memory instructions become inconsequential either due to using the same registers to resolve the address or two different registers with the same resolution at run-time. The latter cannot be removed at compile time
- But once this memory instruction is removed, the registers used for resolving the address can then cause dependency inconsequential instructions before that memory instruction

7 Conclusion

The finding of this assignment reveals that register inconsequential is most common, followed by branch inconsequential, and then memory inconsequential. This pattern may be due to memory access distribution and branch prediction accuracy. Register dependencies significantly contribute to dependency inconsequential instructions across memory, branch, and general categories; they may need to be reduced for better code execution.