Assignment-4 Submission

Indian Institute of Technology Delhi

COL216: Computer Architecture

Name: Deepanshu **Entry Number: 2019CS50427**
Name: Prashant Mishra **Entry Number: 2019CS50506**

1 Introduction

The assignment is about designing and implementing a strategy for efficient ordering of DRAM requests at runtime.

2 Major design choices

Every Engineering problem requires some sort of trade off. This is no other and hence has some strengths and weaknesses. We have made following design choices and following that, we have the strengths and weakness of this approach.

1. We are taking an array of queues. We are in total taking 1024 queues, one for each row in the memory.
2. We are using unordered map for storing labels and their corresponding address.
3. The safety of execution of an instruction is determined by a separate function isSafe(). Every instruction has to pass through this function before execution.

3 Strengths of the Approach

1. This implementation is close to the actual implementation that involves queues.
2. This implementation makes the code more efficient taking lesser number of cycles by storing lw/sw instructions and executing other independent commands in parallel.
3. Time complexity is linear in most cases.

4 Weaknesses of the Approach

1. It requires c++ version of 11 or more as it involves using auto as an iterator.
2. This type of approach might take longer time to run in some cases where almost all the requests are made to single row only.
3. Evertime, we need to traverse through all the 1024 queues.

5 Testing Strategy

5.1 Testing the C++ code

In this , we took the C++ code and run against it for different input files(i.e test cases) which has MIPS commands written on it and finally matched MIPS output for same commands with C++ output.

5.2 Error Handling

Following are the possible errors that can occur while running the code:

1. **Syntax Error :** Following are the various possibilities of syntax error:
   (a) Instructions which are not defined.
   For example : addu $t1,$t2,$t3
   (b) Statement involving unknown characters:
   For example : add $t1 ; $t2 , $t3
   (c) Lesser number of registers used for operation than required:
   For example : add $t1 , $t2
   (d) More number of operators used for operation than required :
   For example : add $t1,$t2,$t3,$t4,$t5
2. **Memory Limited Exceed:** There is a possibility of memory limit exceed if total memory used for storing instruction and data exceed 2^{20} bytes .
3. **Segmentation Fault :** It may occur when we to excess the memory which is not available in the memory.
4. **Large offset used:** If the user tend to give the offset out of bound i.e larger than 2^{20} or less than 0. Even if it is not divisible by 4, we are flagging it as an error.

5.3 Choosing the test cases

All the test cases are provided in the zip file. We covered all the cases extensively ranging from error handling to looping. Some of the cases are as follows:

1. **Empty File :** Gives total commands as 0 and it takes 1 clock cycle to process the entire file.
2. **Basic Operation :** Following are some basic operations to separately test the functioning of the individual command correctly.
   (a) Binary Operation :
   For example : add $t1,$t2,$t3
   (b) Conditional jump :
   For example : beq $t1,$t2, label
   (c) Loading/Storing :
   For example : lw $t1, label
   (d) Unconditional jump :
   For example : j label
   (e) Boolean Operation :
   For example : slt $t1,$t2,$t3
3. **Complex Operations:** Following are some basic operations to separately test the changing of order for optimisation.
   (a) lw/sw used in loop. This includes multiple adding of lw/sw instructions in the queue and then subsequent implementation using reordering of commands.
   (b) lw/sw in a loop where queue number changes. For example, we are accessing the address from $t0 and the increment due to loop leads to changing the queue number of the instruction. (More details in the text case.txt)
   (c) Cases with