COL216 Minor Assignment

Ishita Chaudhary | 2019CS10360

Since this assignment is an extension of assignment 3, some major things to remember are as follows. For more information about the functions involved, kindly read the document for assignment 3.

The following data types and functions are used in C++ code (task1.cpp and task2.cpp):

- DRAM: an array to represent memory and store instructions and data.

  Its size is 2^{18} (1024x1024/4 bytes, as given in the problem statement, with 4 bytes per instruction) is declared with the data type inst_data. This data type is of the form (Boolean, string, integer, integer, integer, string, Boolean, integer, integer).

  Boolean: this is true if the keyword is an instruction and false if it's a data.

  String: this stores the label name. If there is no label, this is an empty string.

  Boolean: this is false if the corresponding operation involves an integer and true if it involves a register.

  This array represents memory stores the instructions and data
- r: an array to store 32 registers. It has $r0 at index 0 to $r30 at index 30. It also has $zero at last, i.e., index 31.
- An unordered map label_map is created to maintains (label name, index) pair.
- An integer PC maintains the index of the memory array, which has to be executed next.
- buffer_row: it's an array with data type same as DRAM and has a size of 256 (each row has 256 columns, i.e., 1024/4 columns).
- no_of_updatesBR: this integer stores the total no. of updates in buffer row.
- buffer_row_no: this stores the row number of DRAM currently activated in buffer_row. It is initialised to -1, implying it no row is activated.

Approach

TASK 1:

For this task, I have considered all 10 instructions that were stated in Assignment 3(i.e., add, addi, sub, mul, j, bne, beq, lw, sw, slt). After each instruction (except lw and sw), the registers are modified accordingly.

In case of lw, first buffer_row_no is checked if they required row is activated or not. In case it is not, the previous row in buffer is copied back to DRAM and cycle count is incremented with random access delay. The required row is copied to buffer row and again the cycle count is incremented by row access delay. Data is copied to register and cycle count is incremented by column access delay.
In case of sw, it works very similar to lw, except the fact that in this case instead of updating the registers we are updating the column offset at buffer row and writing it back to DRAM in column access delay and row access delay cycles respectively.
Finally, the row present in buffer is copied back to DRAM and statistics for the test case are printed.

TASK 2:
I have considered only four functions, add, addi, lw and sw.
I have implemented two seperate functions to check if the next instruction is safe to be executed or not.
In case of sw, add and addi are always safe and other two are not (because DRAM can be accessed only once during a cycle).
In case of lw, add and addi are safe only when the registers involved in the operation are not involved in the lw instruction. This has been taken care of.
A function count_cycles will return the number of cycles the corresponding lw or sw function needs to execute completely.
To run the safe instructions, each instruction is checked for safety, and it is run if and only if three conditions are satisied, i.e., it should be safe, should execute within the number of cycles returned by count_cycle, and the address stored in PC must not exceed the last stored instruction in DRAM.
After the execution of all safe instructions, the cycle count is restored to run the corresponding sw/lw instruction.

Strengths and Weaknesses
STRENGTHS:
The code for task1.cpp handles all 10 instruction.
The code is logically correct and provides detailed accurate results when tested over variety of test cases covering all posiibilities and permutations of different instructions.
After each successful completion of an instruction, it prints the cycle number, activity in DRAM, executed instruction, modified registers and memory blocks.
Both the programs raises exception, errors and syntax errors along with the line number at required places.
Both the code handles labels as well.

WEAKNESS:
The code for task2.cpp handles only four instructions, namely, add, addi, lw and sw.
The code doesn’t make a queue of the independent instructions that can be executed after each cycle.

Testing Strategy
To run a C++ file (task1.cpp in this case), run the following commands on your terminal (ubuntu machine used).

```
$g++ task1.cpp -o output -std=c++11
$./output TEST_FILE_NAME.txt integer1 integer2
```

where integer1 is ROW_ACCESS_DELAY and integer2 is COL_ACCESS_DELAY

There are three test cases in the folder. One exclusively for task1 and the other two can be tested on both, and the difference in the non-blocking (task2) program can be confirmed for its accuracy.