DRAM Request-Manager For Multi-Core Processor

BY

KURISETI RAVI SRI TEJA - 2019CS10369
GATTU KARTHIK - 2019CS10348

Contents

1 Approach Used To Solve 2

2 Assumptions 3

3 TestCases 3

4 Instructions to Run 3

5 Limitations 4

1 Approach Used To Solve

- We assumed that the input commands are stored in a text file with only ".main" part written.(No .data part will be there because there are no commands like syscall and la)
- We read the text file as distinct lines and used a built in regex to split it at commas and whitespaces and convert it into a vector of strings.
- All empty lines in text were removed.
- All such vectors were stored in a 2D vector (vector of vector of strings).
- We used maps to store the contents of registers and to store how many times each command was called.
- We have implemented in such a way that if all the commands which have encountered in a particular cycle are of non (lw,sw) commands then they all will be executed at a time on corresponding cores.
- If there are any syntactic errors then the execution in that core will be stopped by printing the appropriate message.
- If there are any (lw,sw) commands then the corresponding dram request will be raised and this will be pushed into the queue.
- The execution of these requests will be done in the order in which they have been raised.
- When any lw,sw is executed then in other cores if there aren't any dram requests raised then the commands in other cores will be executed along with this lw,sw command.If any other core has raised any dram request then execution in that core will be halted till the priorly raised dram requests have executed.
- After all the execution has been done till the maximum number of cycles given then registers along with their contents will be displayed and the number of times each command is executed will be printed.
- For memory we created a 256*1024 int array(1024 bytes in each row) and for counting row-buffers we used a global variable which was passed

by reference into the corresponding function which takes 0 or 1 or 2 or 3 depending on the functionality.

- We incorporated the delay of MRM assuming that the accessing of memory is done by binary searches with three pointers available that point to the first,middle and the last rows with the delay in cycles of the MRM approximated as the number of binary searches done to find that row.
- We assumed that the DRAM is of finite size(size=10) and the execution in that core halts if the lw/sw command cannot be pushed into DRAM.

2 Assumptions

- Every file should end with “exit:” and begin with “main:”.
- Assumed there are no comments in the given testcase files.
- Took absolute addresses for memory.

3 TestCases

- We have considered the test files which containing all types of instructions and included them here also.
- Tested with files containing syntactic errors.

4 Instructions to Run

- Move the files related to the testcase to the directory containing the code.
- Open the terminal and enter the command **make all** to execute the code. Then use the command **make run** to run the executable.Then input the number of files and then input the file names containing commands successively and input the maximum number of cycles,value of row-access and col-access delays.

- The output will be displayed on the terminal itself.
- Use the command **make clean** to remove all the executables produced.

5 Limitations

- Didn't reorder instructions due to large waiting time in cycles so in some cases throughput value could be lesser than anticipated.