Cache README

Nishi Baranwal, CS23BTECH11041
Paidala Vindhya, CS23BTECH11044

Contents

1 Download

- First, download the zip file **Lab7_CS23BTECH11041_CS23BTECH11044.zip** and extract it in your computer.
- The folder will contain -
  - Header file - riscv_header.h
  - Source files - main.cpp, assembler_gen_func.cpp, simulator_gen_func.cpp, cache_gen_func.cpp, b_format.cpp, i_format.cpp, j_format.cpp, r_format.cpp, s_format.cpp, u_format.cpp, hexcode_generator.cpp
  - Input file - input.s : write your assembly code in this file. If you have to run another file, please copy it into the Lab7_CS23BTECH11041_CS23BTECH11044 folder.
  - Makefile
  - Test Cases, Test Cases 2, Test Cases 3 - file containing cases we tested our code with while developing. Test Cases corresponds to the testing of the assembler part, Test Cases 2 corresponds to the testing of the simulator part, while Test Cases 3 corresponds to the cache part.
  - README file
  - Report file
- Open the folder in terminal and run the command

  ```
  make
  ```

  This command will create the .o object files and the riscv_sim executable.
- You can also run the command **make all** instead of make to do the same thing.

2 How to Use

- This project extends the simulator and assembler we have made previously to simulation of Cache.
- For usage of the Assembler and Simulator parts please see the How To Use - Assembler and Simulator Parts section
- Since we have already created the executable file : riscv_sim (by using the command make - please see the Download section).
  To use the executable we have the syntax :

  ./riscv_sim

- To use the Cache parts of the project, we have the following commands
  - **cache_sim enable config_file**
    - Enables cache simulation for D-cache within the simulator. Uses parameters from config_file for cache settings.
    - The cache configuration is provided as an input file (config_file) in the following format:
      SIZE_OF_CACHE (number)
      BLOCK_SIZE (number)
      ASSOCIATIVITY (number)
      REPLACEMENT_POLICY (FIFO or LRU or RANDOM)
      WRITEBACK_POLICY (WB or WT)
      - SIZE_OF_CACHE: Total size of the cache (specified in Bytes).
      - BLOCK_SIZE: Size of each cache block (specified in Bytes)
      - ASSOCIATIVITY: Associativity (1: Direct mapped, 0: Fully associative, any other number: set associative) - up to 16
      - REPLACEMENT_POLICY: FIFO (First In First Out), LRU (Least Recently Used), RANDOM
      - WRITEBACK_POLICY: WB: WriteBack with Allocate, WT: WriteThrough without allocate
      - Example config file:
        32168
        16
        8
        LRU
        WT
    - The replacement policy and write-back policy need not be all caps
    - The SIZE_OF_CACHE, BLOCK_SIZE, ASSOCIATIVITY must be a power of 2.
  - **cache_sim disable**
    - Disables cache simulation. This is the default state at the start of the simulator.
  - *The above 2 commands can not be executed while a file is being executed or loaded.*
  - **cache_sim status**
    - Prints the enable/disable status of the D-cache simulation.

        * If cache simulation is enabled, it also prints the cache configuration values in the format:
          Example:
          Cache Size: 32168
          Block Size: 16
          Associativity: 8
          Replacement Policy: LRU
          Write Back Policy: WT

    - **cache_sim invalidate**
        * invalidates all entries of the D-cache.

    - **cache_sim dump myFile.out**
        * Writes all the current D-cache entries to the file "myFile.out" in the following format. Notice that only the valid entries have been printed in order of their Set value.
          Set: 0x00, Tag: 0x100, Clean
          Set: 0x01, Tag: 0x123, Clean
          Set: 0x01, Tag: 0x736, Dirty
          Set: 0x02, Tag: 0x145, Dirty
          Set: 0x10, Tag: 0x321, Clean

    - **cache_sim stats**
        * prints the cache statistics for the executing code at the current instance in the format:
          D-cache statistics: Accesses=100, Hit=93, Miss=7, Hit Rate=0.93

    - If cache simulation is enabled, a file named **filename.output** will be generated where filename is the name of the executed file without the extension. This file would contain the cache simulation data in a format as shown:
      Example output:
      R: Address: 0x20202, Set: 0x02, Miss, Tag: 0x202, Clean
      W: Address: 0x10306, Set: 0x06, Hit, Tag: 0x103, Dirty
      R: Address: 0x20511, Set: 0x11, Miss, Tag: 0x205, Clean

    - When the simulator's **run** command is invoked, the cache statistics are printed at the end.

3 Cleaning

- To remove all the object files and the executable file created by the command make run the following command :

  make clean

- After running the above command your directory will be left with just the files that you had initially downloaded, the input files (if added any) and the output files - **output.hex**, **input_file_name.output**, and any other output file for which the **cache_sim dump** command has been used.

4 How To Use - Assembler and Simulator Parts

- Since we have already created the executable file : riscv_sim (by using the command make - please see the Download section).
  To use the executable we have the syntax :

  ./riscv_sim

- To run the assembly code in file (let's say the file name is input.s)
  - run **./riscv_sim** on terminal. Now, we need to use the following commands to do their respective functions.
  - **load** <**file_name**>
    - Make sure the file being loaded is present in the folder. We have provided a file 'input.s', assembly code can be written in that file and run as well.
    - The above will load the input file. containing RISC-V assembly code. All registers will get initialized to the default value. If any data is present in the **.data** section, it will get loaded in the **.data** section of memory. If no errors in input file, the hexcode of the instructions will get loaded in **.text** section of memory
    - If any errors are present in input file, it will be printed. The code will not be 'run' unless the corrected file is loaded again.
    - There should be single space between 'load' and <file_name>.
  - **run**
    - The above will execute the given RISC-V code and update registers, memory, etc. It runs the given RISC-V code till the end.
  - **regs**
    - The above will print the values of all registers in hex format. (64-bit registers).
  - **exit**
    - The above will exit the simulator.
  - **mem** <**addr**> <**count**>
    - The above will print 'count' memory locations starting from address **addr** (the memory will be stored in Little Endian format).
    - Here, count should be non-negative value. If negative value given, error message - **"Invalid count value. Count value should be non-negative"** is printed.
  - **step**
    - The above will run one instruction and print the **"Executed** <**instruction**>**; PC=**<**address**>**"** message.
  - **break** <**line**>
    - The above sets a mark to stop the code execution once the line is reached, preserving registers and memory state.
    - Make sure that the <line> is a valid line number. Otherwise, error message - **"Breakpoint cannot be set. Line exceeds the last line."** is printed.
  - **del break** <**line**>

    - The above deletes the breakpoint at the specified line. If no breakpoint is present, **"Breakpoint at <line> is not set"** message will be printed.
  - **show-stack**
    - The above prints the stack information, and the line elements are pushed onto and popped from the stack frame. The stack frame is only updated on function invocations.

- The input file (containing the assembly code) should follow the following formats:
  - The input instructions should follow the format :

    instruction_name appropriate_operands

    where, there is only 1 space in between the instruction and the first operand. Also, only 1 space between "," and second operand and so on. Similarly, one colon after the label and then one space.
  - Example :

    addi x1, x0, 4
  - The program should start from the first character in each line.
  - There should not be any blank lines in the input assembly code file.
  - There should only be one instruction per line.
  - There should not be any space after the instruction (even on the same line).
  - There should not be any comments in the input assembly code file.
  - There should not be any pseudo instructions in the input assembly code file.
  - Label names cannot have spaces or colons.
  - For the **.data** section, if it is present, then **.text** must be present in the input file.
  - In the .data section for each type of data - .dword, .word, .half, .byte - if multiple values are present in the same line then they should be separated by a comma and a single space.
  - There should not be any space after the (last) data value (even on the same line).
  - The output for each line will be printed in the **output.hex** file.
  - In the event of format not being correct, the Error will be printed on the **Terminal** and the **output.hex** file.