CS204 AY 2020-21 Sem2 Project: RISC-V 32I Simulator
Language for Source Code: Python

TEAM DETAILS: Group 3
Ayush Verma 2019CSB1147
Bhumika 2019CSB1152
Keshav Krishna 2019CSB1224
Rishabh Jain 2019CSB1286
Vishawam Datta 2019CSB1305

HOW TO RUN:
Clone the repository to your local system. After ensuring the installation of the following dependencies, run the steps as mentioned hereafter.
Libraries and installation Requirements:
-Python >=3.3
-importlib
-numpy

Steps to run via command-line interface:

1. Navigate to /src/command_line
2. Enter your machine code in a file with an appropriate file name.
3. Run the command python main.py <path to instruction file> on windows or python3 main.py <path to instruction file> on mac or Linux.
4. User will be prompted to set knobs for the simulator,which include knobs for pipelining and specifications for I$ and D$.
5. Meaningful messages from the simulator and cache stats will be displayed on the terminal.
6. Register and memory outputs will be stored in RegisterDump_<knob_config>.mc and MemoryDump_<knob_config>.mc respectively. User will be informed at the end of the program where the output has been directed to.

Steps to run via GUI interface:

1. Navigate to /src/GUI directory
2. Enter your machine code in a file with an appropriate file name.
3. Run the command python gui_mem.py <path to instruction file> on windows or python3 gui_mem.py <path to instruction file> on mac or Linux.
4. User has to set the cache specifications for I$ and D$ on the GUI and also select a pipeline configuration.
5. The user then has to click on the assemble button to load the program to memory.
6. Then, the user can choose to step through the program or run the program through the given buttons.
7. Contents in I$ and D$ will be printed on the GUI at the end of each cycle. Tag value and the contents in the blocks will be shown. The individual bytes will be separated by “|”.

8. Cache stats(accesses, hits, misses) and pipeline stats will be printed on the terminal at the completion of execution.
9. Meaningful messages from the simulator during program execution will be displayed on the terminal.
10. Register and memory outputs will be stored in RegisterDump.mc and MemoryDump.mc respectively.

NOTE- Command line program is much more stable and bug free than the GUI. Thus, for test cases it is recommended to use the command line program.

PHASE3 DESCRIPTION:

A memory Hierarchy has been implemented with L1 cache and the main memory. The L1 cache is set associative. Write through and write allocate policies have been followed. LRU block eviction has been followed in the cache module.

Cache size, block size in bytes must be a power of 2 and at least 4 bytes. Cache associativity must also be a power of 2, including 1.

It is a pipelined implementation that supports stalling, data forwarding, control hazard detection and static branch prediction of the 5-step-single-cycle instruction execution is done.

Program input code:
The input is taken in the form of a list of machine codes in the format:
<address of instruction> <machine code of the instruction> <optional '#' beginning comments>
Text segment has to be flagged by '~text' and the data segment by '~data'.
The program terminates with the "<address of instruction> 0x11" code.

Type of instructions supported:
R format - add, and, or, sll, slt, sra, srl, sub, xor, mul, div, rem
I format - addi, andi, ori, lb, lh, lw, jalr
S format - sb, sw, sh
SB format - beq, bne, bge, blt
U format - auipc, lui
UJ format - jal

The code runs the machine code by running all 5 stages of the execution process at the same time and stalls, forwards data whenever required. The code supports and successfully executes programs comprising the above instruction types and is tested on programs like Fibonacci, Factorial and Bubble Sort.

CONTRIBUTIONS:
As a team effort, no strict separation was followed. A loose outline of the work is as follows:

Ayush Verma: LRU Unit(Cache), Hazard Unit(Check_dependence, type of forwarding and type of stalling), Decode unit functions, Register And Memory output functions.

Bhumika: Decode unit, Control Circuitry including Control Signals and ALU Control, Buffers, Hazard Table, Data Forwarding, GUI(phase 3), some part of cache

Keshav Krishna: Buffers, Program Flow(pipelining), Control Circuit, some parts of memory, GUI(phase 1), some part of cache, GUI(phase 3)

Rishabh Jain: Memory, register, ALU, Control and program flow and GUI(phase 1), stalling mechanism, forwarding mechanism, control signal queues, Branch prediction, cache read/write methods and design

Vishawam Datta: Hazard Unit(Check_dependence, type of forwarding and type of stalling),LRU unit (cache), Program Flow, IAG module including BTB, memory module and some parts of Control