REPORT – LAB 7 – CACHE SIMULATOR

This project is an extension of a RISC-V simulator built in C, enhanced with cache simulator for D-cache. It supports the execution and debugging of RISC-V assembly code, providing detailed simulation of cache behavior. The simulator reads assembly instructions from a file, simulates execution with 64-bit registers and memory (including the stack), and tracks the program counter (PC) in real-time. The following cache-specific commands extend the simulator's functionality:

- cache_sim enable <configuration file> : Enables cache simulation using settings from the specified configuration file.
- cache_sim disable : Disables cache simulation (default state).
- cache_sim status : Displays the cache simulation status and configuration details if enabled.
- cache_sim invalidate : Clears all entries in the D-cache.
- cache_sim dump myFile.out : Exports valid D-cache entries to a file in a specified format.
- cache_sim stats : Shows current cache statistics, including access count, hits, misses, and hit rate.

FUNCTION PROTOTYPES AND INPUT FILES:

The function prototypes of all the functions inside the files are also written before the main function. Some functions are written before the main function, which do not require function prototype.

Input files: Assembly code is written in input files of form input1.s/input2.s/input3.s etc.. and after assembler files are executed it will create hex code file for each of the input file of the file name output1.hex, output2.hex, output3.hex etc.. Cache specifications are written in a file named config.txt. These three are the input files for the simulator file named excstack_1.c.

NOTE: This report contains the brief explanation of the parts related to cache. To understand the simulator part please refer to the previous Lab Assignment's report.

Part 1,2,3 are supported by my code.

MAIN FUNCTION

- Enable Command: The command cache_sim enable <config_file> enables the cache simulator using the specified configuration file. It loads the cache settings through the load_cache_config() function.

- Disable Command: The cache_sim disable command disables the cache simulation by invoking the disable_cache() function.
- Status Command: When cache_sim status is entered, it displays the current cache configuration and state using the display_cache_status() function.
- Invalidate Command: The cache_sim invalidate command clears all cache entries, marking them invalid via the invalidate_cache() function.
- Dump Command: The command cache_sim dump <filename1> writes the current cache contents to the specified file using the dump_cache1() function.
- Stats Command: The cache_sim stats command shows cache performance metrics such as hit/miss counts, using the show_cache_stats() function.

If an invalid cache command is entered, it outputs "Unknown cache_sim command."

LOAD FUNCTION

There is no major changes in the Load function from Simulator to Cache simulator.

The load function initializes and prepares memory for executing a program from a specified file.

- **Resets Simulator**: Clears previous data by calling reset_simulator(), initialize_memory(), and initialize_stack() to set up a clean environment.
- **Opens File**: Opens the specified file and reads instructions line by line.
- **Parses Data Directives**: Processes .data, .text, .dword, .word, .half, and .byte directives to allocate memory and store data in Memory based on their size.
- **Stores Instructions**: Loads instructions into memory sequentially, checking for memory overflow.
- **Loads Binary Instructions**: Reads the output file output.hex, stores binary instructions in memorybin, and loads each byte into Memory for execution.
- **Sets Program Counter**: Initializes the program counter (pc) to the start of the program.

RUN FUNCTION

Executes Instructions: Loops through each instruction, handling labels and jumps by pushing and popping functions onto a stack when needed.

Instruction Execution: Calls execute() on each instruction, and monitors for breakpoints to pause if reached.

Breakpoint Management: Checks and manages user-set breakpoints, halting execution at specified points.

Cache Statistics: At the end of execution, if caching is enabled, it shows cache statistics and saves them to a file via dump_cache().

EXECUTE FUNCTION

All the instruction types work similar to the Simulator, except Load and Store where Cache/Memory access is required.

Instead of accessing Memory/Cache in the Execute function, there is a separate handle_load_instruction and handle_store_instruction for load and store instructions respectively.

Load Instruction Handling (Opcode: 0x03):

- This case processes load instructions that transfer data from memory into registers. Fields such as rd, funct3, rs1, and imm are extracted.
- The effective address is computed by adding the base address in rs1 to the sign-extended immediate value imm.
- A boundary check is performed on the calculated memory address to prevent out-of-bounds access.
- Depending on funct3, different load types are supported: LB, LH, LW, LD, and their unsigned versions (LBU, LHU, LWU), allowing data loads of varying byte sizes.
- The function handle_load_instruction is then called to load data from either memory or cache, based on cache settings, and store the value in the specified register rd.

Store Instruction Handling (Opcode: 0x23):

- This case handles store instructions that write data from a register to memory. The immediate (imm), funct3, rs1, and rs2 fields are extracted from the instruction.
- The effective address is calculated by adding the base address in rs1 to the sign-extended immediate imm.
- A boundary check ensures that the memory address is within valid data memory bounds.
- Depending on funct3, different types of store operations (SB, SH, SW, SD) are supported, which allow storing 1 byte, 2 bytes, 4 bytes, or 8 bytes.
- The handle_store_instruction function is then called to store data in memory or cache, based on cache specifications, using the value in rs2.

CACHE SPECIFICATIONS STORING

Global Cache Variables:

- cachestat: Indicates whether caching is enabled (true or false).
- cachecount: Tracks the number of log entries in cache_log.
- cache_log: Array to store log messages related to cache events.

Cache Structures:

- CacheLine:
    - Fields include valid and dirty bits to indicate line status.
    - data: Stores the cached data.
    - last_access_time: For tracking least recently used (LRU) replacements.
    - tag: Stores the address tag for cache lookups.
- CacheSet:
    - Contains an array of CacheLine objects (lines), with the maximum number defined by MAX_ASSOCIATIVITY.
- Cache:
    - Sets: Array of CacheSet structures, with MAX_CACHE_SETS sets.
    - Fields include enabled status, cache_size, block_size, associativity, replacement and write policies (replacement_policy and write_policy), and statistics such as access_count, hit_count, and miss_count.

Instance:

- dcache is an instance of the Cache structure used for managing the data cache.

LOAD CACHE_CONFIG

The load_cache_config function initializes the cache configuration by loading values from a specified configuration file. It sets the cache size, block size, associativity, replacement policy, and write policy directly into the dcache instance.

After successful loading, the cache is enabled (dcache.enabled set to 1), and statistics counters like access_count, hit_count, and miss_count are initialized to zero. Each line in every set of the cache is marked invalid and clean (dirty bit cleared).

Once configured, cachestat is set to true, and the cache is cleared of any prior data by calling invalidate_cache.

DISABLE CACHE

The disable_cache function disables the cache simulation by setting cachestat to false, and invalidating cache, signaling that cache operations are inactive. It then outputs a message confirming the cache simulation has been disabled.

DISPLAY CACHE_STATUS:

The display_cache_status function checks if the cache simulation is enabled (cachestat). If enabled, it displays detailed cache configuration, including cache size, block size, associativity, replacement policy, and write policy. If disabled, it simply indicates that the cache simulation is inactive.

INVALIDATE_CACHE:

The invalidate_cache function resets the cache by iterating through each line in every set, marking them as invalid (by setting the valid and dirty flags to 0), clearing the tag, data, and last_access_time fields. This effectively clears all cache contents and prepares it for fresh use. A message "Cache invalidated" is printed to confirm the operation.

SHOW_CACHE_STATS:

The show_cache_stats function displays the current statistics for the data cache (D-cache). It outputs the total number of accesses, hits, and misses, along with the hit rate, which is calculated as the ratio of hits to total accesses. If there are no accesses, the hit rate is displayed as 0.0. This function helps in monitoring cache performance.

HANDLE_LOAD_INSTRUCTION FUNCTION:

It first checks if the cache simulator is enabled or not. If it is not enabled, it simply loads the value from memory and returns the value back to the execute function, where this value is written into the destination register.

If it is enabled,

- Incrementing access_count: The access count for the memory is incremented each time the function is called to track how many accesses have been made to the cache.
- Calculating Cache Parameters: The number of sets, block offset bits, and index bits are computed using the cache size, block size, and associativity. These values help in determining where to look for the requested data in the cache.
- Calculating Set Index and Tag: The function extracts the set index and tag from the memory address by performing bitwise shifts and masking operations. The set index determines which cache set to look in, and the tag is used to match the correct cache line.

- Cache Hit Check: The function checks each cache line in the selected set for a match by comparing the tag. If a matching line is found, it is a cache hit. The data is loaded from the cache into the destination register, and the last_access_time is updated for the LRU (Least Recently Used) policy. Then, the required statement of cache_log which contains lines like “R: Address: 0x20202, Set: 0x02, Miss, Tag: 0x202, Clean” is stored in the cache_log array after incrementing hit_count.
- Cache Miss Handling: If no cache line matches (i.e., a miss occurs), the miss count is incremented, and a log entry is made. The function then selects a cache line to replace, based on the cache's replacement policy (FIFO, LRU, or RANDOM).
- Replacement Policy:
  - FIFO: The function uses a round-robin approach for cache line replacement. It maintains a static index to replace the cache line in a sequential order, starting from the first and moving to the next for each new replacement.
  - LRU: The function tracks the last access time of each cache line. When a miss occurs, it replaces the cache line with the oldest accessed (least recently used) one by comparing the last access times.
  - RANDOM: A random cache line is selected for replacement when a miss occurs. The function uses the rand() function to randomly choose an index within the set’s associativity.
- Write-back/ Write-through: If the cache uses a write-through policy, the function immediately writes the data to memory after a cache miss. If the cache line being replaced is dirty and the write-back policy is in use, the function writes the dirty data back to memory before replacing it.
- Loading Data into Cache: The selected cache line is updated with the new data from memory. The tag is updated, the cache line is marked as valid, and the dirty flag is set to clean. The data is also loaded into the destination register, and a message is logged.

HANDLE STORE INSTRUCTION

It first checks if the cache simulator is enabled or not. If it is not enabled, it simply writes to the memory and goes back to the execute function.

Is its enabled,

- Incrementing access_count: The access count for the memory is incremented each time the function is called to track how many accesses have been made to the cache.
- Calculating Cache Parameters: The number of sets, block offset bits, and index bits are computed using the cache size, block size, and associativity. These values help in determining where to look for the requested data in the cache.
- Calculating Set Index and Tag: The function extracts the set index and tag from the memory address by performing bitwise shifts and masking operations. The set index

determines which cache set to look in, and the tag is used to match the correct cache line.

- Cache Hit Check: The function checks each cache line in the selected set for a match by comparing the tag. If a matching line is found, it is a cache hit. The data is stored in the cache according to the configured write policy.
  Write-through (WT): If the write policy is "WT", the data is written both to the cache and to memory.
  Write-back (WB): If the write policy is "WB", the data is only written to the cache, and the cache line is marked as dirty to indicate that it needs to be written back to memory later.
  If the replacement policy is "LRU", the access time of the cache line is updated for Least Recently Used (LRU) management.
  After handling the cache hit, the function logs the operation with a message indicating that the data was written and the cache status
- If no matching cache line is found (cache miss), the miss count is incremented.
  The function logs the miss along with the address, set index, tag, and the cache state (Clean).
  Write-through (WT) on Miss: If the cache uses the "WT" policy, the function writes the data directly to memory without updating the cache.
- If a cache miss occurs and write-allocate is enabled, the function must find a cache line to replace based on the configured replacement policy:
  - FIFO: The function uses a round-robin approach for cache line replacement. It maintains a static index to replace the cache line in a sequential order, starting from the first and moving to the next for each new replacement.
  - LRU: The function tracks the last access time of each cache line. When a miss occurs, it replaces the cache line with the oldest accessed (least recently used) one by comparing the last access times.
  - RANDOM: A random cache line is selected for replacement when a miss occurs. The function uses the rand() function to randomly choose an index within the set's associativity.
- If the cache line that is being replaced is dirty (i.e., it contains data that needs to be written back to memory), the function writes the dirty data to memory before replacing the line.
- The chosen cache line is updated with the new data. The tag is updated, the cache line is marked as valid, and the last access time is updated. If the write policy is "WB", the cache line is marked as dirty to indicate that it will need to be written back to memory later.

DUMP_CACHE1

The dump_cache1 function writes the contents of the data cache (D-cache) to a specified file. It first calculates the number of cache sets based on the cache size, block size, and associativity. Then, it iterates over each set and each line within the set,

checking if the cache line is valid. For valid lines, it writes the **set**, tag, and the line's status (dirty or clean) to the file. The file is closed after writing, and the function ends without printing any messages to the terminal.

DUMP_CACHE

The dump_cache function creates an output filename by removing the file extension from the input filename and appending **".output".** It then prints the cache logs to the console and writes the logs to the newly created output file. If the file cannot be opened, an error message is displayed. After writing the logs, the file is closed, and a success message is printed, indicating the file where the cache logs were saved.

The output file generated by the dump_cache function is named as <input filename>.output. This file contains cache logs, where each log entry is saved on a new line, detailing the cache activities recorded during execution. Each line typically reflects a cache access, status, or modification, formatted as a readable log entry from the cache_log array. For dump_cache1, the output file includes details of the data cache (D-cache) contents, listing valid cache lines with their set index, tag, and line status (either "Dirty" or "Clean").

ERROR HANDLING

- Includes all the Error handling implemented in Simulator
- In main function, if the cache command entered is not among the available ones, it will enter a error message "Unknown cache_sim command!”
- While loading into memory, if the address exceeds the MEMORY_SIZE it prints an out of bound error. And while writing into memory similar error message is printed.
- If the cache is disabled, and trying to print the cache_stats, it will print an error message “Cache Simulation Status: Disabled”.
- File opening errors are printed whenever the files are being tried to open, but due to some issue its not being opened.

ASSUMPTIONS

- No blank line in either of the input files.
- Pseudo Instructions are not supported for translation or execution.
- Maximum number of instructions in the input file is assumed to be 1000.
- Maximum length of Stack is assumed to be 1000

- Maximum label length and maximum command length is assumed to be 200 each
- Immediate/offset values provided with instructions is assumed to be in decimal only.
- The instructions are properly entered with valid spaces ie, there is only 1 space in between the instruction and the first operand. Also, only 1 space between "," and second operand and so on.
- Memory size is 0x60000 and text section from 0x00000 and data section starts from 0x10000. Stack starts from 0x50000.
- Jalr should have the format – jalr x0, 0(x1) and not the RIPES format, as the provided testcases had the same.
- EOL/EOF has to be properly entered.
- The instruction inside the label is to be written in the same line
- It is assumed that all memory accesses (load/store operations) are correctly aligned according to the data type being accessed.
- ****I assumed that each of the .dword/.word/.byte/.half should be written in separate lines.**
- ****Each of the input file(assembly code) should follow the form of input<no.>.s so that the hex file for each will follow name output<no.>.hex.**
- Assume that no read/write will span more than 1 cache block.
- MAX_ASSOCIATIVITY is assumed to be 512
- MAX_CACHE_SETS is assumed to be 1024

FUTURE FEATURES

- Comments: A line starting with a semicolon (;) can be treated as a comment and omitted. Similarly, anything after a semicolon (;) within a line can be omitted. This feature could be added.
- Pseudo Instructions: Support translation of specific pseudo instructions like mv, j etc.. This feature could be added.
- Implementing for many other replacement policies like LFU, LIFO etc..
- GUI implementation of simulator