CS 4380 Project 3: Experimenting with Caching

Introduction

As you have learned, the concept of caching plays a critical role in the memory hierarchy (arguably it is the POINT of the memory hierarchy). In this project you will simulate and study the effects of adding various caches to the 4380.

Part 1: 4380 Emulator and Assembler Mark 2

In order to meet the requirements of this lab it will be necessary to extend the 4380 emulator and assemblers with support for the following additional instructions:

Move Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
14	ISTR	RS	RG	DC*	DC*	Store integer in RS at address in RG
15	ILDR	RD	RG	DC*	DC*	Load integer at address in RG into RD
16	ISTB	RS	RG	DC*	DC*	Store byte in RS at address in RG
17	ILDB	RD	RG	DC*	DC*	Load byte at address in RG into RD

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

JMP Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
2	JMR	RS	DC*	DC*	DC*	Update PC to value in RS
3	BNZ	RS	DC*	DC*	Address**	Update PC to Address if RS != 0
4	BGT	RS	DC*	DC*	Address**	Update PC to Address if RS > 0
5	BLT	RS	DC*	DC*	Address**	Update PC to Address if RS < 0
6	BRZ	RS	DC*	DC*	Address**	Update PC to Address if RS = 0

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.
** When represented in an assembly file the Address shall be represented as a label. Such labels MUST be associated with instructions.

Compare Instructions: (Note: All comparisons are on signed values)

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
29	CMP	RD	RS1	RS2	DC**	Set RD = 0 if RS1 == RS2 | Set RD = 1 if RS1 > RS2 | Set RD = -1 if RS1 < RS2
30	CMPI	RD	RS1	DC**	Imm*	Set RD = 0 if RS1 == Imm | Set RD = 1 if RS1 > Imm | Set RD = -1 if RS1 < Imm

* When represented in an assembly file the Imm value shall be represented as a numeric or character literal.
**This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

Unless otherwise specified below - all requirements from projects 1 and 2 remain in force for the assembler and emulator.

Part 2: It's about time...

Caches are meant to increase the speed of a processor by reducing the number of clock cycles spent waiting on memory. To measure this speedup, we must first be able to measure the number of clock cycles spent on memory operations in general. To make this possible you must modify your emu4380 program to meet the following requirements:

Req1) The emulator must implement a global `unsigned int` variable called **mem_cycle_cntr**. This variable must be declared in emu4380.h and defined in emu4380.cpp, and shall be initialized to zero at program startup.

Req2) Project 3 work must be completed in a separate branch called project-3. When all requirements are met, the branch should be merged into main (BUT NOT DELETED).

Req3) The emulator must contain the following memory access functions which must adhere to their associated definitions and descriptions:

- `unsigned char readByte(unsigned int address)`: Returns the unsigned char located at index `address` in **prog_mem**. Also increments the global **mem_cycle_cntr** by 8 when called.
- `unsigned int readWord(unsigned int address)`: Returns the unsigned int located at index `address` in **prog_mem**. Also increments the global **mem_cycle_cntr** by 8 when called.
- `void writeByte(unsigned int address, unsigned char byte)`: Places the value in `byte` at index `address` in the **prog_mem** array. Also increments the global **mem_cycle_cntr** by 8 when called.
- `void writeWord(unsigned int address, unsigned int word)`: Places the value in `word`, beginning at index `address`, in the **prog_mem** array. Also increments the global **mem_cycle_cntr** by 8 when called.

These functions shall be declared in emu4380.h and defined in emu4380.cpp

Req4) All programmatic memory accesses performed in the emulator (i.e. as the result of executing bytecode), including instruction fetches and data accesses, shall utilize the appropriate memory access functions described in Req3 above. **Note:** The initialization of the PC to the first four bytes of program memory at startup shall NOT be considered a memory operation for timing purposes. This operation may be performed by either accessing memory directly (as in prior projects), or by resetting the mem_cycle_cntr to zero after memory access function call(s) are utilized to perform the task.

Req5) When a TRP #0 instruction is encountered - in addition to exiting with a return value of 0, the emulator shall first print the message: "Execution completed. Total memory cycles: NUM_CYCLES\n" - where NUM_CYCLES is the current value of **mem_cycle_cntr.**

Part 3: A pair of caches...

Now that your emulator is reporting the number of cycles spent accessing memory it is time to implement your first cache by extending your emulator to meet the following requirements:

Req6) The emulator shall be modified to accept two optional command line arguments (indicated by flags) in addition to the required input filename. These arguments shall be the size of program memory (originally specified as an optional command argument) and the type of cache configuration to be used by the emulator. Both optional arguments shall be indicated by a corresponding flag; -m for memory size, -c for cache configuration. In both cases the flag shall be followed by whitespace, and then the appropriate/desired value. For example "-m 1024" to specify a memory size of 1024 bytes. If the -m flag is invoked but not followed by a valid memory size the emulator shall print the message "Invalid memory configuration. Aborting.\n", and exit with a return value of 2.

When specifying cache configuration, the value following -c (if present) must be an unsigned integer value which shall indicate which of 3 available cache configurations is to be utilized by the emulator. The valid arguments are as follows:

- No argument : Default to no cache
- 0: No cache
- 1: Direct Mapped Cache
- 2: Fully Associative Cache
- 3: 2-Way Set Associative Cache (extra credit)

If an invalid argument is provided the emulator shall print the message "Invalid cache configuration. Aborting.\n", and exit with a return value of 2.

NOTE: This requirement supersedes previous requirements concerning command line arguments.

Req7) The emulator shall implement a direct mapped cache of 64 cache lines with a block size of 16 bytes (the number of cache lines and block size shall be easily configurable at compile time - i.e. not hard coded to specific values). The cache shall implement a write-back policy.

Req8) The emulator shall implement a fully associative cache of 64 cache lines with a block size of 16 bytes (the number of cache lines and block size shall be easily configurable at compile time - i.e. not hard coded to specific values). The cache shall implement a least-recently-used eviction policy with write-back.

Req9) The emulator must contain the following function:

`void init_cache(unsigned int cacheType)` : Initializes the emulator's cache in accordance with the argument values described in Req6. This function must be called before the memory access functions described in Req3 can be called.

This functions shall be declared in emu4380.h and defined in emu4380.cpp

Req10) When the emulator is invoked with the optional command line argument indicating that a cache is to be used, the memory access functions described in Req3 shall utilize the appropriate cache described in requirements Req7 and Req8 above, and in Req12 below, to simulate cached memory accesses. When a memory access results in a hit within the cache the **mem_cycle_cntr** shall be incremented by 1 INSTEAD of 8. When a memory access results in a miss requiring one or more blocks to be read from memory into the cache an additional 8 cycles shall be added to **mem_cycle_cntr** for the first 4 bytes read, and 2 cycles for each subsequent 4 byte word read into the cache.

The above requirement implies that for a miss causing a single cache block to be read before returning the requested value: 1 + 8 + 2*3 = 15 clock cycles shall be added to **mem_cycle_cntr**. If two cache blocks must be read, then 15 + 2 * 4 = 23 clocks cycles are required.

If a memory access results in a miss that also causes one or more evictions; if the block(s) currently present in a cache line must be written back to memory then an additional 8 cycles shall be added to **mem_cycle_cntr** for the first 4 bytes written, plus 2 cycles for each subsequent 4 byte word written; for a total of 14 cycles to write back the first block. If two blocks must be written back, then an additional 8 cycles would be required to write back the second block. Thus, in the **worst** case a memory operation could cause 23 + 22 = 45 memory clock cycles to be consumed. (You may want to ask yourself how many clock cycles are required in the best case to preserve your sanity at this point...)

Req11) When no cache is being utilized, the emu4380's Fetch phase shall require 8 memory cycles to obtain the first word of an instruction, and 2 cycles for the second

word of the instruction, for a total of 10 clock cycles to fetch an instruction in the absence of a cache.

Part 4: Adding our third cache... (Extra Credit +10%)

With your first two caches operating correctly you may obtain extra credit by implementing a 2-Way Set Associative cache of the same size (same number of cache lines, and with the same block size) as the previously described caches.

> **Req12**) The emulator shall implement a 2-way set associative cache of 64 cache lines with a block size of 16 bytes (the number of cache lines and block size shall be easily configurable at compile time - i.e. not hard coded to specific values). The cache shall implement a least recently used (LRU) eviction policy with write-back with each set.

Part 5: Compare and contrast

Using the provided sample programs, compare and contrast the performance of your processor with a) no cache, b) the direct mapped cache, c) the fully associative cache, and d) (optionally) the 2-way set associative cache. **You must generate tables and/or plots showing comparisons for each of the example programs**. Discuss what you see in each case and explain WHY the observed outcomes are happening.

Part 6: What if?

Once you have collected and discussed the data from Part 5, perform the same experiment but this time with 32 cache lines and a block size of 32 bytes for each of the cache configurations. Once again, generate tables and/or plots showing comparisons for each of the example programs. Discuss what you see in each case and explain WHY the observed outcomes are happening.

What to Turn In:

When you have completed the project you will need to submit the URL for your repository on the assignment page in Canvas. Your repository should be private **but shared with your course grader and/or instructor.** Instructions for granting access to your repo for the course grader will be provided by your instructor.

You must also submit a pdf containing your results and discussion for Parts 5 and 6 of this project.

Note: If you are invoking the Non-Late policy please insert a note at the top of your README file that states “INVOKING THE NON_LATE POLICY” followed immediately by your work log. (The rest of your README should still contain relevant project details including those discussed in previous projects).