UIC – ECE 366: Computer Organization - Fall 2020

Project 4: Cache Simulator

Introduction:

For this project, you can form your own group (1-3, by default I assume you work by yourself) and will write a python program, which takes as input, a text file **`mc.txt`** (containing the machine code of a MIPS program in hex), some user input (cache configurations and running mode), and output some important information of this MIPS program's cache behavior:

Assumptions:

- Relevant instructions are of memory access (`lw, sw, lb, sb`)
- Replacement policy: LRU (least recently used)
- Write allocate policy: upon sw and sb miss, the block in memory will be brought to cache.

Cache configurations:

`N`-way `S`-set `SA` cache with block size of `b` Bytes

1. `(N=1, S=1, b=32)`
   a simplest cache, with only 1 block, size of 8 words.
2. `(N=1, S=4, b=128)`
   a DM cache with 4 sets, each containing only 1 way / block, block size of 32 words.
3. `(N=2, S=4, b=64)`
   a 2-way 4-set, SA cache, block size of 16 words
4. `(N=4, S=1, b=64)`
   a 4-way FA cache, block size of 16 words

(extra credit: 5pts) support any cache configurations by allowing the user to input a valid configuration of `(N, S, b)`

Program functionality:

i. Asks the user to provide a cache configuration (can use 1, 2, 3, 4 to indicate which of the above configuration to use).

ii. Asks the user to select between a "detailed display mode", and a "fast mode".

iii. During "detailed display mode", the program pauses (waiting for user's key stroke to continue) after each memory access instruction (lb, lw, sb, sw), and displays the following information in a neat way:

- memory address breakdown (in binary): how the address is split into block # in memory and in-block offset; what are the tag bits and set_id for this access;
- current cache information: which set will be checked for this access, the information of each way (valid bit, tag bits) in this set;
- hit or miss, updated hit count / miss count so far;
- cache update log:
  - upon hit: cache update information wrt LRU replacement;
  - upon miss: cache update information (way, tag, valid bit, LRU info).

After the program finishes, the final data should be shown: total number of hit, total number of miss, and overall hit rate for the entire run of the program.

iv. During the "fast mode", your program only produces the final data of hit / miss count and hit rate.

You can use MARS to help verifying your program's correctness, with **Tools -> Data Cache Simulator** to compare the cache behavior.

Submission instructions:

Individually, upload the following files on Bb:

1. **`report.pdf`:** **a self-contain PDF report**
2. `sim.py`: cache simulator

PDF Report components:

Part A) (10 pts) Project reflections

- List out your group members and activity log.
- What are some of the features that you find useful in Python?
- What would you advise other students about the projects for this course in the future?

Part B) (90 pts) Cache Simulator Results

1. **(75 pts) Simulator showcase:** Demonstrate with screenshots the interface and results of your simulator. Use version 1 of the Test Program and illustrate the results for all 4 cache configurations. Choose some representative screenshots to showcase your output style, particularly for the detailed mode.

2. **(15 pts) Overall conclusions:** an illustrative table or chart – use your simulator to collect cache hit / miss results by running the given 3 versions of the test program with the given 4 cache configurations. Your table / chart should attempt to answer the following questions, well annotated.

   Which cache configuration works the best overall? Does parameters (i.e., different versions) matter for cache performance?

You will be graded by your program functionality, output design, and final results (table / chart) presentation.

(extra credit 5 pts): flexible cache configurations
Allow the user to specify N, S, and b (assuming to be valid) for any cache configuration.

(extra credit 5 pts): Evaluation on Perceptron
Now that your simulator can run pretty much any MIPS program and produce interesting data with regard to processor and cache performance, it can be used to examine the performance of your perceptron program. Compare various cache configurations with the common setting of 256KB in capacity, find out which configuration (b, N, S) produces the best result for your Perceptron program, and show an overall chart of comparison. Attach your perceptron assembly file in submission if you have done this so that your results can be reproduced and verified.

Appendix: Test Program - set an array of numbers, then find the min value among every neighborhood of n numbers and store the min to an array some distance after

```
# version 1
# parameter = (0x2070, 3)

ori $8, $0, 2
addi $9, $0, 0x60

sw_loop:

sw $8, 0x2000($9)
addi $9, $9, -4
beq $9, $0, sw_done

addu $8, $8, $8
sub $8, $0, $8
addi $8, $8, -3

beq $0, $0, sw_loop

sw_done:
addi $8, $0, 0x2070
addi $10, $0, 0x2060
addi $9, $0, 0x2000

outer_loop:
addi $14, $0, 3
lw $11, 0($9)

inner_loop:
addi $9, $9, 4
lw $12, 0($9)
slt $13, $12, $11
beq $13, $0, skip
addu $11, $0, $12

skip:
addi $14, $14, -1
bne $14, $0, inner_loop

sw $11, 0($8)
addi $8, $8, 4
slt $13, $9, $10
bne $13, $0, outer_loop
```

```
# version 2
# parameter = (0x2078, 5)

ori $8, $0, 2
addi $9, $0, 0x60

sw_loop:

sw $8, 0x2000($9)
addi $9, $9, -4
beq $9, $0, sw_done

addu $8, $8, $8
sub $8, $0, $8
addi $8, $8, -3

beq $0, $0, sw_loop

sw_done:
addi $8, $0, 0x2078
addi $10, $0, 0x2060
addi $9, $0, 0x2000

outer_loop:
addi $14, $0, 5
lw $11, 0($9)

inner_loop:
addi $9, $9, 4
lw $12, 0($9)
slt $13, $12, $11
beq $13, $0, skip
addu $11, $0, $12

skip:
addi $14, $14, -1
bne $14, $0, inner_loop

sw $11, 0($8)
addi $8, $8, 4
slt $13, $9, $10
bne $13, $0, outer_loop
```

```
# version 3
# parameter = (0x2090, 11)

ori $8, $0, 2
addi $9, $0, 0x60

sw_loop:

sw $8, 0x2000($9)
addi $9, $9, -4
beq $9, $0, sw_done

addu $8, $8, $8
sub $8, $0, $8
addi $8, $8, -3

beq $0, $0, sw_loop

sw_done:
addi $8, $0, 0x2090
addi $10, $0, 0x2060
addi $9, $0, 0x2000

outer_loop:
addi $14, $0, 11
lw $11, 0($9)

inner_loop:
addi $9, $9, 4
lw $12, 0($9)
slt $13, $12, $11
beq $13, $0, skip
addu $11, $0, $12

skip:
addi $14, $14, -1
bne $14, $0, inner_loop

sw $11, 0($8)
addi $8, $8, 4
slt $13, $9, $10
bne $13, $0, outer_loop
```