Memory Cache Simulator and MIPS Assembly

Colt Uhlenhopp & Jack Utzerath

2/6/2023

CST-307

Part 1:

Wave.prg & 1 Processor(Po):

	2 Way	2 Way	2 Way	2 Way	4 Way	4 Way	4 Way	4 Way	8 Way	8 Way	8 Way	8 Way
	LRU	FIFO	RAND	LFU	LRU	FIFO	RAND	LFU	LRU	FIFO	RAND	LFU
8 Blocks	Hit: 72.7% Miss: 27.3%	Hit: 72.19 % Miss: 27.809 %	Hit: 70.937 % Miss: 29.063 %	Hit: 72.483 % Miss: 27.517 %	Hit: 73.125 % Miss: 26.875 %	Hit: 72.454 % Miss: 27.546 %	Hit: 71.112 % Miss: 28.888 %	Hit: 72.687 % 27.313 %	Hit: 73.359 % 26.641 %	Hit: 73.067 % 26.933 %	Hit: 70.820 % 29.180 %	Hit: 73.096 % 26.904 %
16 Blocks	Hit: 80.858 % M:19. 142	Hit: 80.274 % 19.726 %	Hit: 79.457 % 20.543 %	Hit: 80.333 % 19.667 %	Hit: 81.441 % 18.559	Hit: 80.741 19.259	Hit: 79.807 20.193	Hit: 80.624 % 19.376	Hit: 81.762 % 18.282 %	Hit: 81.033 % 18.967 %	Hit: 79.428 % 20.572 %	Hit: 81.150 % 18.850 %
32 Blocks	Hit: 85.585 % 14.415 %	Hit: 84.885 % 15.115 %	Hit: 84.418 % 15.582 %	Hit: 85.031 % 14.969 %	Hit: 85.702 % 14.298 %	Hit: 85.147 % 14.853 %	Hit: 84.301 % 15.699 %	Hit: 85.381 % 14.619 %	Hit: 86.169 % 13.831 %	Hit: 85.614 % 14.386 %	Hit: 82.628 % 17.362 %	Hit: 85.760 % 14.240 %
64 Blocks	Hit: 89.524 % 10.476 %	Hit: 88.970 % 11.030 %	Hit: 88.065 % 11.935 %	Hit: 88.882 % 11.118 %	Hit: 88.845 % 10.155 %	Hit: 89.057 % 10.943 %	Hit: 88.299 % 11.701 %	Hit: 89.087 % 10.913 %	Hit: 90.020 % 9.980 %	Hit: 89.349 % 10.651 %	Hit: 86.781 % 13.219 %	Hit: 89.203 % 10.797 %

Wave_Burst.prg & 1 Processor (Po):

	2 Way	2 Way	2 Way	2 Way	4 Way	4 Way	4 Way	4 Way	8 Way	8 Way	8 Way	8 Way
	LRU	FIFO	RAND	LFU	LRU	FIFO	RAND	LFU	LRU	FIFO	RAND	LFU
8 Blocks	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %	Hit: 95.444 %

	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %	Miss: 4.556 %
16 Blocks	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %
32 Blocks	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %
64 Blocks	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %

Wave.prg 2 Processors (Po and P1):

	2 Way	2 Way	2 Way	2 Way	4 Way	4 Way	4 Way	4 Way	8 Way	8 Way	8 Way	8 Way
	LRU	FIFO	RAN D	LFU	LRU	FIFO	RAN D	LFU	LRU	FIFO	RAN D	LFU
8 Blocks	Hit: 67.552 % Miss: 32.448 %	Hit: 67.26 % Miss: 32.74 %	Hit: 66.034 % Miss: 33.966 %	Hit: 67.085 % Miss: 32.915 %	Hit: 67.669 % Miss: 32.331 %	Hit: 67.26 % Miss: 32.74 %	Hit: 65.101 % Miss: 34.899 %	Hit: 67.348 % Miss: 32.652 %	Hit: 68.106 % Miss: 31.894 %	Hit: 68.106 % Miss: 31.894 %	Hit: 65.130 % Miss: 34.870 %	Hit: 67.756 % Miss: 32.244 %
16 Blocks	Hit: 74.263 % Miss: 25.737 %	Hit: 74.380 % Miss: 25.620 %	Hit: 72.833 % Miss: 27.167 %	Hit: 73.913 % Miss: 26.087 %	Hit: 74.876 % Miss: 25.124 %	Hit: 75.080 % Miss: 25.920 %	Hit: 72.221 % Miss: 27.779 %	Hit: 74.584 % Miss: 25.416 %	Hit: 75.197 % Miss: 24.803 %	Hit: 74.73 % Miss: 25.27 %	Hit: 72.542 % Miss: 27.458 %	Hit: 75.051 % Miss: 24.949 %
32 Blocks	Hit: 78.903 % Miss: 21.097 %	Hit: 78.494 % Miss: 21.506 %	Hit: 77.181 % Miss: 22.819 %	Hit: 78.203 % Miss: 21.797 %	Hit: 79.757 % Miss: 21.243 %	Hit: 78.494 % Miss: 21.506 %	Hit: 76.247 % Miss: 23.753 %	Hit: 78.669 % Miss: 21.331 %	Hit: 79.136 % Miss: 20.864 %	Hit: 79.08 % Miss: 20.92 %	Hit: 76.889 % Miss: 23.111 %	Hit: 78.903 % Miss: 21.097 %

64 Blocks	Hit: 80.245 % Miss: 19.755 %	Hit: 79.895 % Miss: 20.201 5%	Hit: 77.998 % Miss: 22.002 %	Hit: 80.070 % Miss: 19.930 %	Hit: 80.683 % Miss: 19.317 %	Hit: 80.537 % Miss: 19.463 %	Hit: 77.882 % Miss: 22.118 %	Hit: 79.924 % Miss: 20.076 %	Hit: 80.712 % Miss: 19.288 %	Hit: 80.391 % Miss: 19.609 %	Hit: 77.210 % Miss: 22.790 %	Hit: 80.537 % Miss: 19.463 %

Wave_Burst 2 Processors (Po and P1):

	2 Way	2 Way	2 Way	2 Way	4 Way	4 Way	4 Way	4 Way	8 Way	8 Way	8 Way	8 Way
	LRU	FIFO	RAN D	LFU	LRU	FIFO	RAN D	LFU	LRU	FIFO	RAN D	LFU
8 Blocks	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %	Hit: 95.444 % Miss: 4.556 %
16 Blocks	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %	Hit: 97.518 % Miss: 2.482 %
32 Blocks	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %	Hit: 98.548 % Miss: 1.452 %
64 Blocks	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %	Hit: 99.062 % Miss: 0.938 %

a. First use one processor *P*0 and then use 2 processors *P*0 *and P*1.

In the first two tables, one processor was used. In the second two tables, two processors were in use. As seen by the table, the number of processors seemed to have no effect on the wave_burst.prg. However, for the wave.prg, the table with two processors had a lower hit rate and a higher miss rate.

b. Analyze the effectiveness of different block replacement techniques by listing down the miss rate in each case.

The effectiveness of the different block replacement techniques are listed in each of their respective tables. The hit and miss rate was recorded for each. To find these values, the SMPCache

Simulator was used. After inputting the correct information for each table, the hit and miss rate were calculated by the simulator and recorded into their respective cells on the table.

c. What other block replacement technique can be used and is proved to be the ideal? Explain.

Another block replacement technique that can be used is the Least Frequently Used (LFU) method. This caching algorithm takes the least frequently used block in the cache and removes it whenever the cache becomes overflowed. This method is ideal because it actually gets rid of the least common block in the cache rather than getting rid of a random block in the cache or getting rid of a block in the first in first out method.

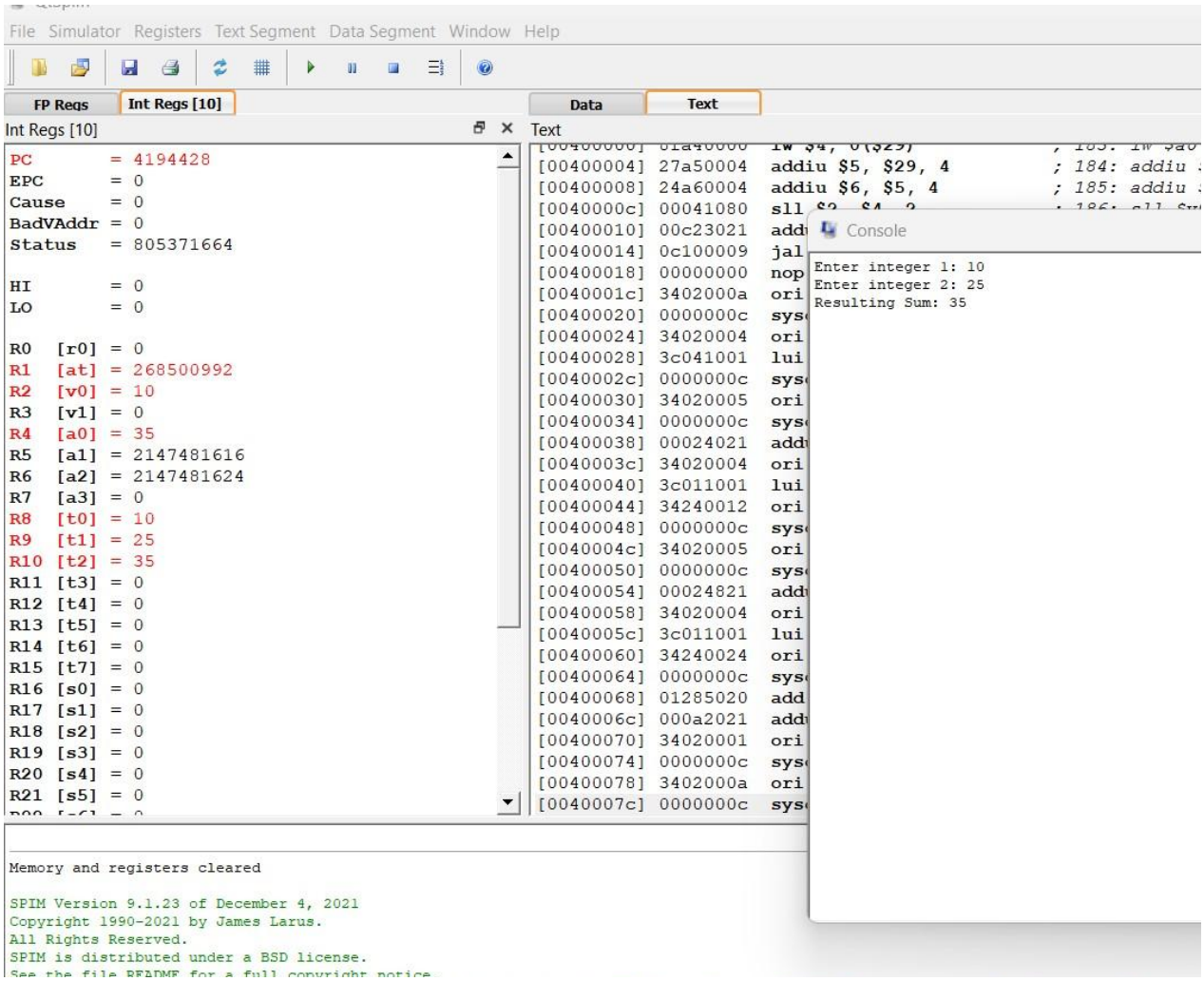
Part 2: MIPS Code

Program 1: Print the sum of two integers specified at runtime by the user

```
# Data Declarations
#=======================================================

# Different text lines to print
num1:          .asciiz "Enter integer 1: "  # Asking user to enter first int
num2:          .asciiz "Enter integer 2: "  # Asking user to enter second int
sum:           .asciiz "Resulting Sum: "     # Printing resultant sum

#=======================================================

        .globl main

        .text

#=======================================================
# Main Code / Text Section

main:

# First Integer

        li $v0, 4              # print_string
        la $a0, num1   # input into a0
        syscall
        li $v0, 5              # read_integer
        syscall
        move $t0, $v0  # move into register t0

# Second Integer

        li $v0, 4              # print_string
        la $a0, num2   # input into a0
        syscall

        li $v0, 5              # read_integer
        syscall

        move $t1, $v0  # move input to register $t1
        li $v0, 4              # print_string
        la $a0, sum            # puts sum in a0
        syscall

        add $t2, $t1, $t0      # add integers
        move $a0, $t2  # move sum to a0 to be printed
        li $v0, 1              # command to print the sum
        syscall

        li $v0, 10             # Ends program
        syscall
```

*The code will also be provided in a separate submitted document.

Execution of Program1:



Program 2: Print the larger of two numbers specified at runtime by the user.

```
# Colt Uhlenhopp & Jack Utzerath
# Program 2 CLC 1
# 2/3/2023
# This Program prints the larger number inputed by the user in the console


# Data Declarations
# ============================================================
          .data

first:    .asciiz "Enter first integer: "
second:   .asciiz "Enter second integer: "
n:                .asciiz "\n"
larger: .asciiz "The larger int is: "

# ============================================================
# Main Code Section

          .globl main
          .text

main:
          li $v0, 4           # code for printing string
          la $a0, first       # loads address of string to be printed in a0
          syscall

          li $v0, 5           # code for read_int
          syscall
          move $t0, $v0       # moves the user input into t0

          #
          li $v0, 4           # code for printing string
          la $a0, second      # loads address of string to be printed in a0
          syscall

          li $v0, 5           # code for read_int
          syscall
          move $t1, $v0       # moves user input into t1

          bgt $t0, $t1, t0big # if t0 is bigger than t1, go to t0big method
          move $t2, $t1                 # if not, move t1 into t2
          b endif                       # go to endif method

#================================================================
# t0big Method

t0big:
          move $t2, $t0       # moves t0 into t2

#================================================================
# endif Method

endif:
          li $v0, 4           # code for printing string
          la $a0, larger      # load address of string to be printed
          syscall

          move $a0, $t2       # moves t2 into a0 to be printed
          li $v0, 1           # code for printing integer
          syscall

          li $v0, 4           # code for printing string
          la $a0, n           # loads address of string to be printed
          syscall

end:
          li $v0, 10                    # code for exit of program
          syscall
```

Execution of Program2: