Alex Santana
ECE365
Project Phase 3

Application 1:

This application takes a number and puts it through a random function to calculate a floating-point value. The magic function, magicFunc(n), is equal to magicFunc(n - 1) - magicFunc(n - 2)/4. The initial value is set in the data section on the code. This first program will use a smaller cache size. The initial value is set to 15.

Code:

```
# ECE365 Project Phase 3 Application 1

# Alex Santana

.data

num:    .word   15      # index


.text

main:

        lw      $a0, num

        jal     magicFunc


        li      $v0, 2          # print float


        mtc1    $zero, $f6      # clear

        cvt.s.w $f6, $f6        # convert


        add.s   $f12, $f6, $f30 # print number return

        syscall


        #END PROGRAM----

        li      $v0, 10

        syscall

        #---------------
```

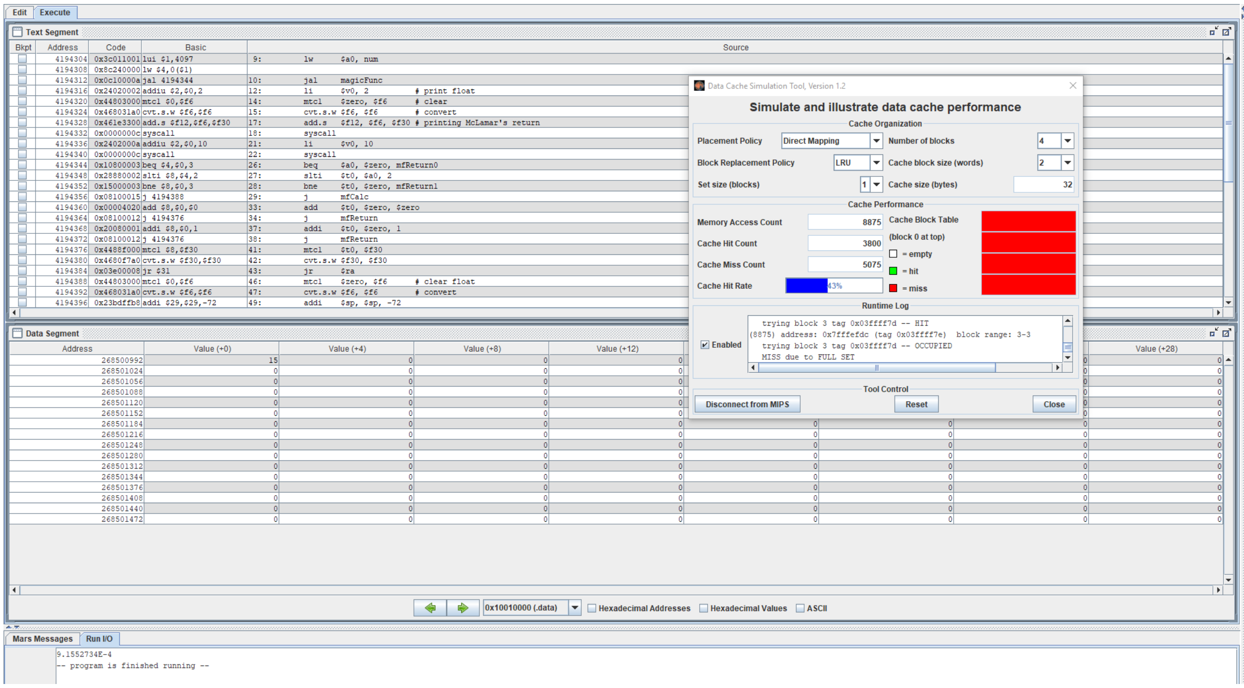
Alex Santana
ECE365
Project Phase 3

```
magicFunc:
        beq     $a0, $zero, mfReturn0
        slti    $t0, $a0, 2
        bne     $t0, $zero, mfReturn1
        j       mfCalc


mfReturn0:
        add     $t0, $zero, $zero
        j       mfReturn

mfReturn1:
        addi    $t0, $zero, 1
        j       mfReturn

mfReturn:
        mtc1    $t0, $f30
        cvt.s.w $f30, $f30
        jr      $ra

mfCalc:
        mtc1    $zero, $f6          # clear float
        cvt.s.w $f6, $f6            # convert

        addi    $sp, $sp, -72
        sw      $ra, 0($sp)
        sw      $a0, 4($sp)
```

```
swc1    $f0, 8($sp)
swc1    $f2, 40($sp)

addi    $a0, $a0, -1
jal     magicFunc
add.s   $f0, $f6, $f30

lw      $a0, 4($sp)
addi    $a0, $a0, -2
jal     magicFunc
add.s   $f2, $f6, $f30

addi    $t0, $zero, 4
mtc1    $t0, $f4
cvt.s.w $f4, $f4        # convert
div.s   $f2, $f2, $f4

sub.s   $f0, $f0, $f2
add.s   $f30, $f6, $f0

lw      $ra, 0($sp)
lw      $a0, 4($sp)
lwc1    $f0, 8($sp)
lwc1    $f2, 40($sp)
addi    $sp, $sp, 72
jr      $ra
```

Alex Santana
ECE365
Project Phase 3

Direct Mapping



Number of blocks: 4

Cache block size: 2

Memory access count: 8875

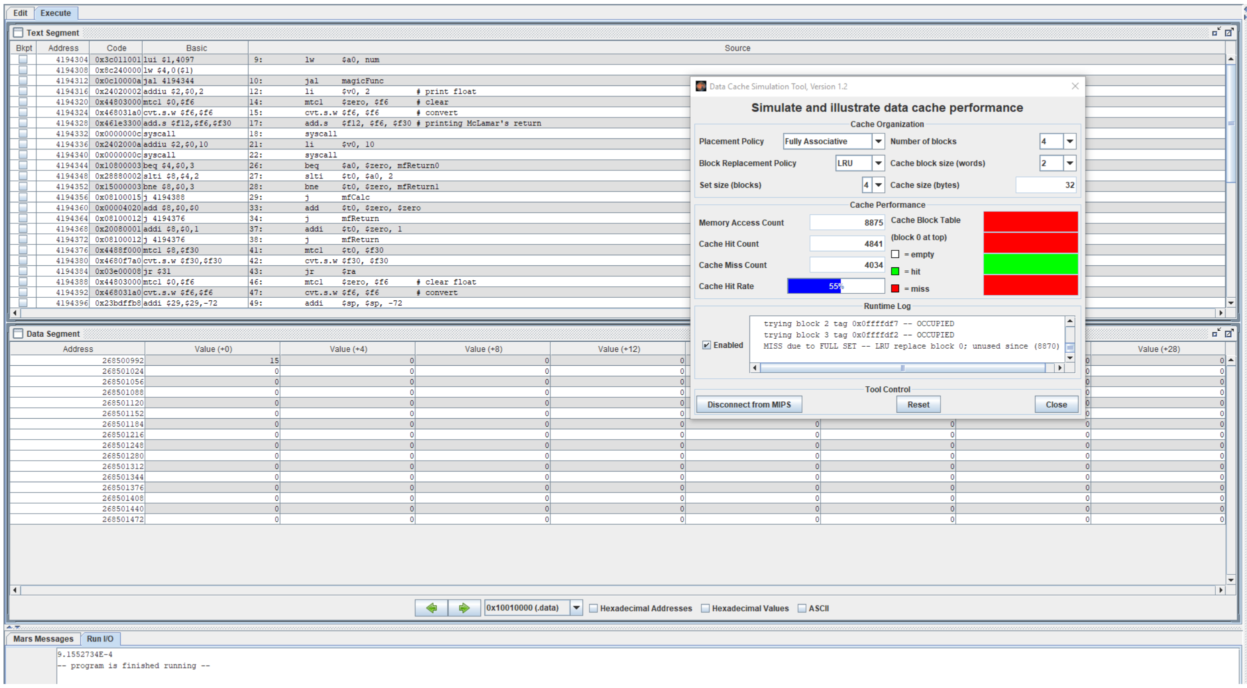
Cache hit count: 3800

Cache miss count: 5075

Cache hit ratio: 43%

Alex Santana
ECE365
Project Phase 3

Fully Associative



Number of blocks: 4

Cache block size: 2

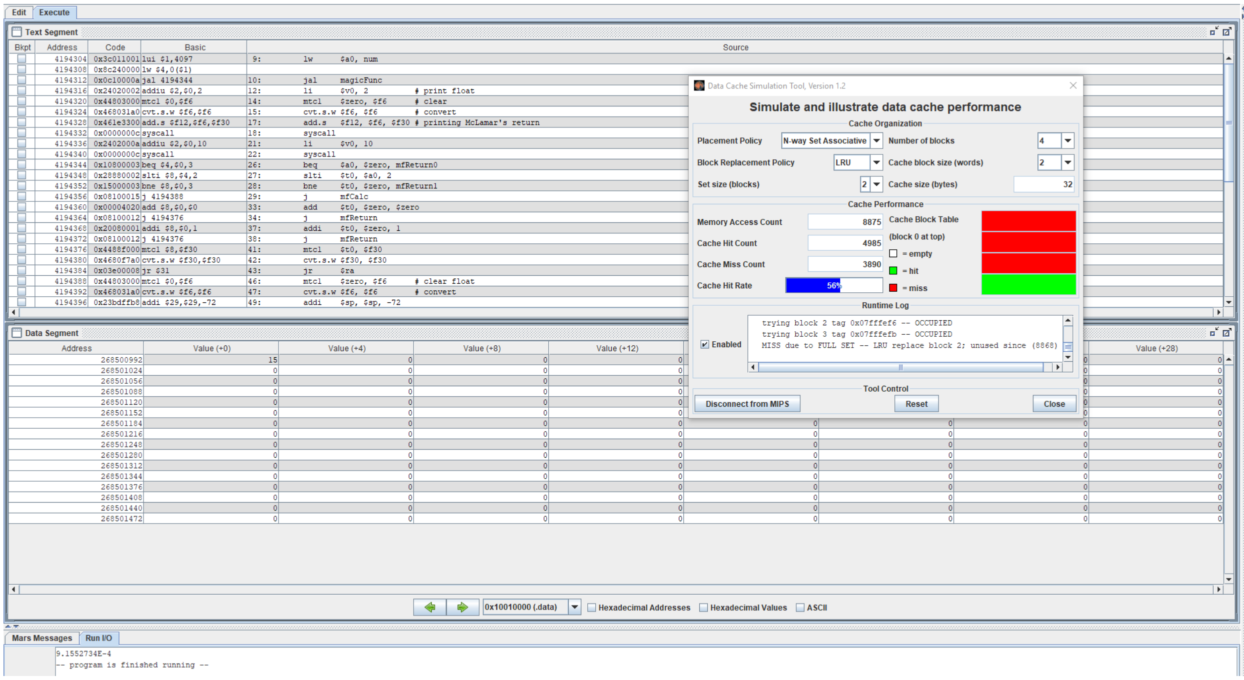
Memory access count: 8875

Cache hit count: 4841

Cache miss count: 4034

Cache hit ratio: 55%

Set Associative



Number of blocks: 4

Cache block size: 2

Memory access count: 8875

Cache hit count: 4985

Cache miss count: 3890

Cache hit ratio: 56%

Conclusion

This program help visualize how mapping affects performance. In this program the number 15 was put into a function. This allowed the program to return the same value after stepping through the same steps through all 3 mapping techniques. Because the function only did arithmetic, I set the blocks and size to a smaller amount so that it may use the cache more rigorously. This helped depict the different in hit ration among the set techniques and direct. Direct was obviously returned the lowest hit ratio. The difference between fully and set differed by only a percent as the instructions were not as complex. Even with these restrictions, the performance increase may still be seen amount the different techniques.

Alex Santana
ECE365
Project Phase 3

Application 2:

This application accesses an array and returns information such as steps size and rep count in order to better visualize cache operation and performance. Because of the random nature of the program, hit ratio was not constant but stayed in a range among different mappings.

Code:

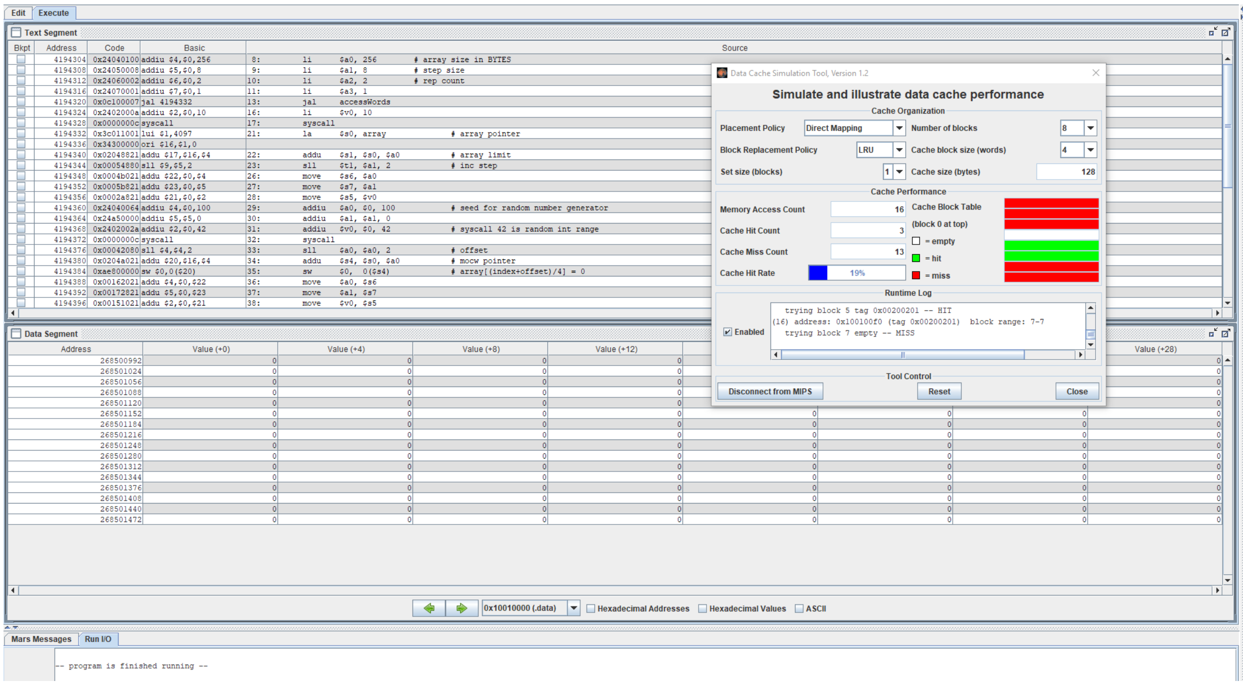
```
# ECE365 Project Phase 3 Application 2

# Alex Santana

.data

arr:    .space  2048            # max array size in BYTES


.text

main:

        li      $a0, 256        # array size in BYTES

        li      $a1, 8          # step size

        li      $a2, 2          # rep count

        li      $a3, 1


        jal     wordAccess


        #END PROGRAM----

        li      $v0, 10

        syscall

        #---------------


wordAccess:

        la      $s0, arr        # array pointer

        addu    $s1, $s0, $a0           # array limit
```

```
	sll	$t1, $a1, 2		# inc step

wordLP:
	move	$s6, $a0
	move	$s7, $a1
	move	$s5, $v0
	addiu	$a0, $0, 100		# seed for random number generator
	addiu	$a1, $a1, 0
	addiu	$v0, $0, 42		# syscall 42 is random int range
	syscall
	sll	$a0, $a0, 2		# offset
	addu	$s4, $s0, $a0		# move pointer
	sw	$0, 0($s4)		# array[(index+offset)/4] = 0
	move	$a0, $s6
	move	$a1, $s7
	move	$v0, $s5

wordCheck:
	addu	$s0, $s0, $t1		# increment ptr
	blt	$s0, $s1, wordLP
	addi	$a2, $a2, -1
	bgtz	$a2, wordAccess
	jr	$ra

byteAccess:
	la	$s0, arr	# array pointer
	addu	$s1, $s0, $a0		# array limit
byteLP:
```

```
        beq     $a3, $0, byteZero

        lbu     $t0, 0($s0)             # inc index
        addi    $t0, $t0, 1
        sb      $t0, 0($s0)
        j       byteCheck

byteZero:
        sb      $0, 0($s0)              # reset index

byteCheck:
        addu    $s0, $s0, $a1           # inc pointer
        blt     $s0, $s1, byteLP

        addi    $a2, $a2, -1
        bgtz    $a2, byteAccess
        jr      $ra
```

Alex Santana
ECE365
Project Phase 3

Direct Mapping



Number of blocks: 8

Cache block size: 4

Memory access count: 16

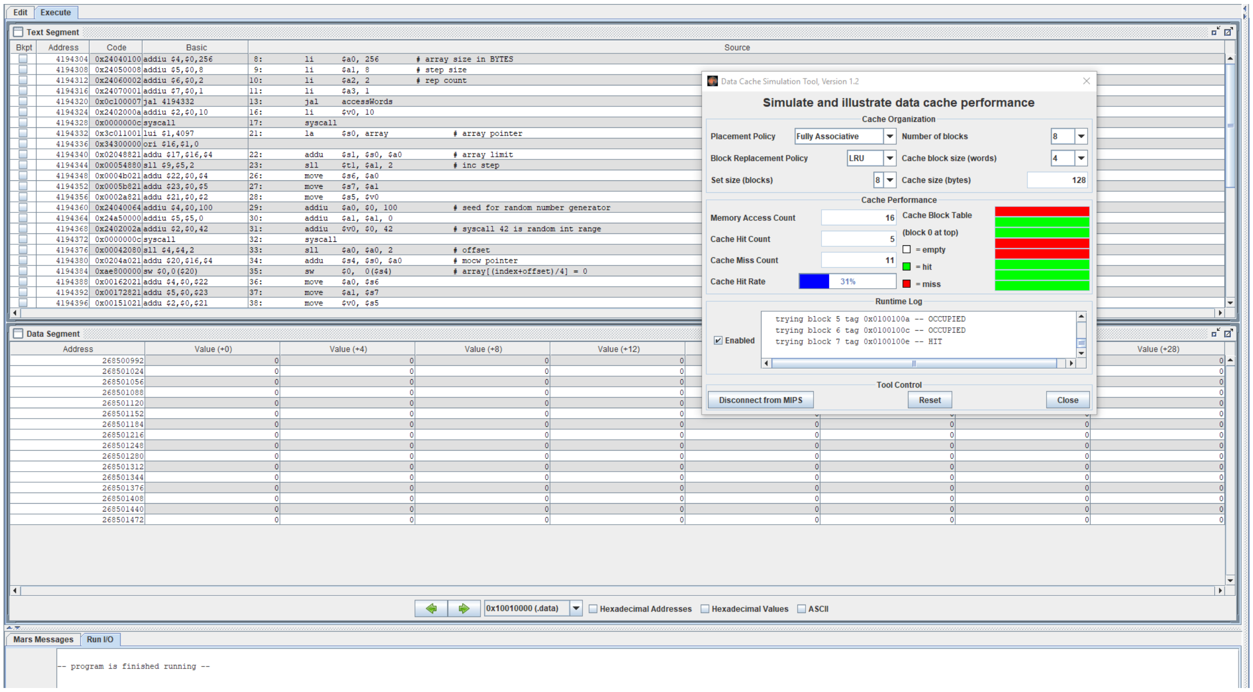
Cache hit count: 3

Cache miss count: 13

Cache hit ratio: 19%

Alex Santana
ECE365
Project Phase 3

Fully Associative



Number of blocks: 8

Cache block size: 4

Memory access count: 16

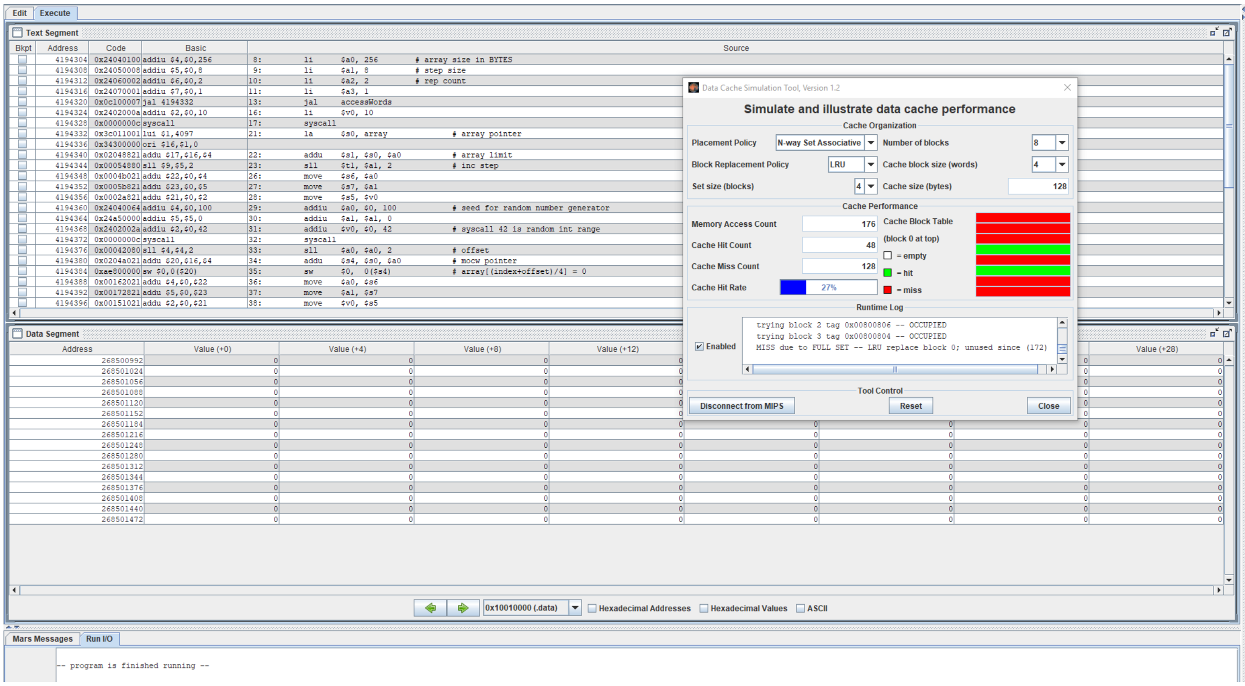
Cache hit count: 5

Cache miss count: 11

Cache hit ratio: 31%

Alex Santana
ECE365
Project Phase 3

Set Associative



Number of blocks: 8

Cache block size: 4

Memory access count: 176

Cache hit count: 48

Cache miss count: 128

Cache hit ratio: 27%

Conclusion

I noticed the hit rate was non-deterministic. I believe this was due to the random nature of the program, but I was able to still conclude that improvement was attained when moving away from direct mapping. Because of the small number of blocks and size, direct mapping resulted in a hit ratio of under 20% as the rewriting of the black in the small case cause misses repeatedly. Upon changing mapping, =I noticed an increase in hit count and ratio as I was testing different number of sets. Fully associative and set associative mappings gave similar results but not as close as the previous program.