schen29_366_p1_report

Thursday, September 19, 2019 8:43 PM

Project 1 Report Questions

a. **Which level does your program achieve? How many hours have you spend in total on this project? What was the most difficult (or time-consuming) part of this project?**

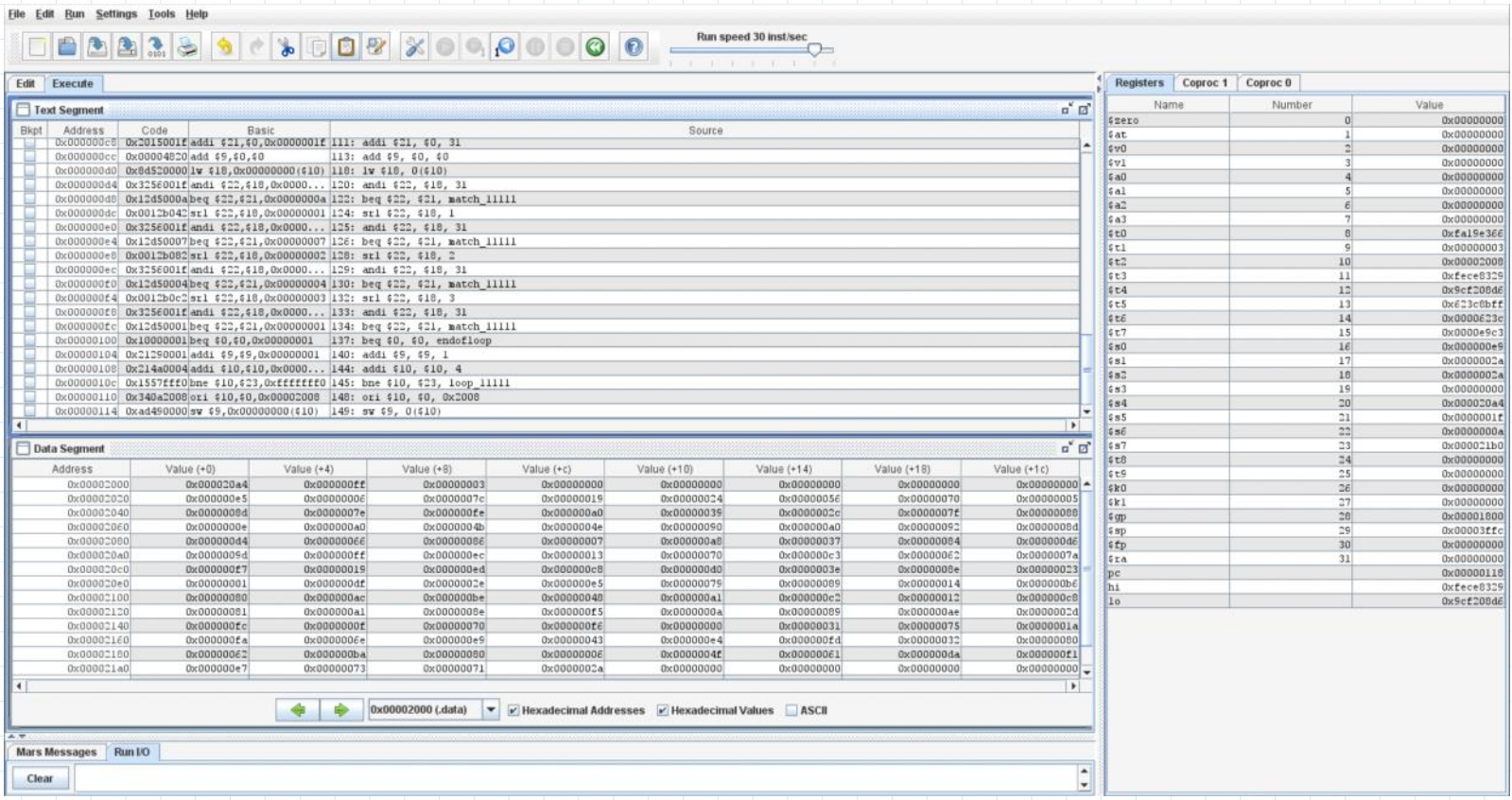
The program achieves (A) + (B) (i), (ii). In total, I spent about 5 hours on this project. The most difficult part of the project was forming the pseudocode that fit the required task the program was to do, since it involved creative thinking in addition to prior knowledge of how information is handled within the simulated MIPS processor.

b. **What is the total (dynamic) instruction count of your program? If extra credit will be given to achieve low dynamical instruction count, how would you optimize your program?**

There were 4,994 instructions run during the execution of this program. In the future, I would attempt to reduce the number of instructions required to move and compare data, such as in the pattern matching section with multiple shifts and comparison branches.

c. **Overall, do you think this multiply-fold is a good Hash Function? Provide some reasons or analysis based on data evidences.**

This function does not appear to be a good function for generating data in this manner. There appears to be a lot of granular information lost in the folding process, and the resulting hash table also appears to have frequent collisions due to the small size of data elements.




```
# Author: Sheng Chen
# For: UIC ECE 366 Fall 2019 Project 1
# "Multiply and Fold" Hash Function
# Calculate C = H(A, B)for a given B = 0xFA19E366,
# for all the A values ranging from 1 to 100,
# and write these 100 results of C in memory starting at 0x2020.

# initialize B
lui $8, 0xFA19
ori $8, $8, 0xE366
# initialize A
addi $9, $0, 1
# initialize value of A at which main function loop should stop
addi $23, $0, 101
# initialize memory address for storing C
ori $10, $0, 0x2020
# loop for hash function data generation, stop branching when $9 == 101
loop:
# A * B
mult $9, $8
# A1 = hi XOR lo
mfhi $11
mflo $12
xor $13, $11, $12
# A1 * B
mult $13, $8
mfhi $11
mflo $12
# A2 = hi XOR lo
xor $13, $11, $12
# A2 * B
mult $13, $8
mfhi $11
mflo $12
# A3 = hi XOR lo
xor $13, $11, $12
# A3 * B
mult $13, $8
```

```
mfhi $11
mflo $12
# A4 = hi XOR lo
xor $13, $11, $12
# A4 * B
mult $13, $8
mfhi $11
mflo $12
# A5 = hi XOR lo
xor $13, $11, $12
# A5[31:16]
srl $14, $13, 16
# A5[15:0]
andi $15, $13, 0x0000FFFF
# C = A5[31:16] XOR A5[15:0]
xor $15, $14, $15
# C[15:8]
srl $16, $15, 8
# C[7:0]
andi $17, $15, 0x000000FF
# C = C[15:8] XOR C[7:0]
xor $17, $16, $17
# store C in memory
sw $17, 0($10)
# increment memory address to next word
addi $10, $10, 4
# increment A by 1
addi $9, $9, 1
# loop again with next value of A
bne $9, $23, loop

# among the 100 calculated values,
# search for the largest C and write its address in 0x2000,
# and its value in 0x2004.
# reset address to load from 0x2020 onwards
ori $10, $0, 0x2020
# set target stop address for end of loop_biggest iteration
ori $23, $0, 0x21B0
# initialize first value to be compared
lw $18, 0($10)

# loop for comparisons, stop branching when when stop address is reached
loop_biggest:
# increment memory address to next word
addi $10, $10, 4
lw $19, 0($10)
# set comparison flag if new bigger number is loaded
slt $9, $18, $19
# branch to end of loop if not set
beq $9, $0, not_bigger
# if flag set, record address of new biggest value and move biggest value for next comparison
add $20, $0, $10
add $18, $0, $19
not_bigger:
# loop again with next address
bne $10, $23, loop_biggest

# store address of largest value in 0x2000
ori $10, $0, 0x2000
sw $20, 0($10)
# store largest value in 0x2004
ori $10, $0, 0x2004
sw $18, 0($10)

# check if each C in the array contains "11111" anywhere in the 8 bit binary (Y / N)
# the total number of "Y" should be written into memory location 0x2008.
# reset address read head to 0x2020, target stop address in $23 is already 0x21B0
ori $10, $0, 0x2020
# initialize register containing pattern
addi $21, $0, 31
# reset $9 for use as counter for values that match
add $9, $0, $0

# loop for comparisons, stop branching when when stop address is reached
loop_11111:
# initialize value to be compared
lw $18, 0($10)
# first comparison: check if bits 4:0 of value is 0b11111
andi $22, $18, 31
# skip increment if not equal
beq $22, $21, match_11111
# second comparison: shift value right to compare for bits 5:1
srl $22, $18, 1
andi $22, $18, 31
beq $22, $21, match_11111
# third comparison: shift value right to compare for bits 6:2
srl $22, $18, 2
andi $22, $18, 31
beq $22, $21, match_11111
# fourth comparison: shift value right to compare for bits 7:3
srl $22, $18, 3
andi $22, $18, 31
beq $22, $21, match_11111
# no match, go to end of loop
# (this is supposed to be a j instruction but the 15-instruction limit was reached)
beq $0, $0, endofloop
# pattern match increment
match_11111:
addi $9, $9, 1
# next loop around
endofloop:
# increment memory address to next word
addi $10, $10, 4
bne $10, $23, loop_11111

# store match count in 0x2008
ori $10, $0, 0x2008
sw $9, 0($10)
```