Homework 1 Report

Author: Zhengqi Yang

Date: 10/27/2019

**Part 1:**

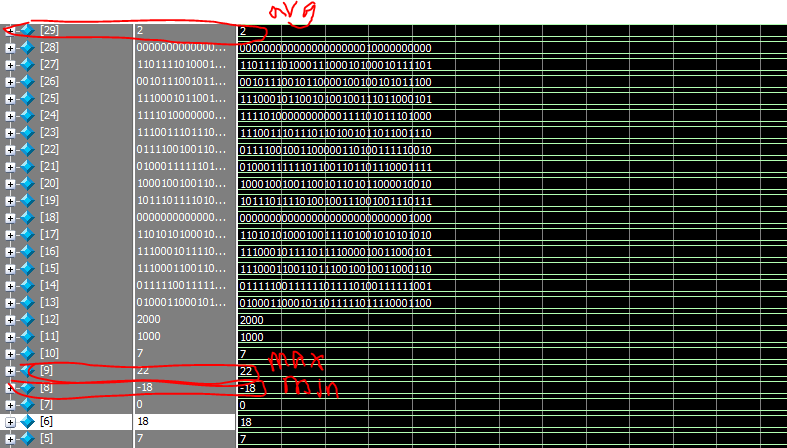
I have 7 inputs for the Min/Max/Avg Calculation problem. It consists of both positive and negative numbers in order to prove the algorithm working without loss of generality. The maximum number is 22 and the minimum number is -18. The average of these numbers should be 2 in decimal. The following figure proved the algorithm working correctly. The minimum value was stored in [8]. The maximum value was stored in [9]. The average value was stored in [29]. As this figure shows, this algorithm yields correct results. All the results were displayed in decimal.

Figure 1: ModelSim Verification.

The following is the assembly code. The highlighted part was modified.

main:

ori $a0, $zero, 7 #Number of values in array

ori $a1, $zero, 1000 #Input address

ori $a2, $zero, 2000 #Output address

ori $t0, $zero, 8 #Make t0 8

sw $t0, 0($a1) #store 8 into memory

ori $t0, $zero, 11

sw $t0, 4($a1)

ori $t0, $zero, -11

sw $t0, 8($a1)

ori $t0, $zero, 6

sw $t0, 12($a1)

ori $t0, $zero, -18

sw $t0, 16($a1)

ori $t0, $zero, 22

sw $t0, 20($a1)

ori $t0, $zero, 0

sw $t0, 24($a1)

jal $ra, min

jal $ra, max

jal $ra, avg

halt:

jal $ra, halt

min:

lw $s0, 0($a1) #s0 min

addi $t3, $a1, 0 #t3 current address

addi $t0, $zero, 1

minloop:

addi $t0, $t0, 1 #t0 loop count

addi $t3, $t3, 4 #current address

lw $t1, 0($t3) #load current value into t1

slt $t2, $s0, $t1 #if statement in t2 s0 < t1

bne $zero, $t2, minendif #if s0 less than t1 jump

ori $s0, $t1, 0 #else statement

minendif:

bne $t0, $a0, minloop #looping branch

sw $s0, 0($a2) #Save min

jr $ra #return

max:

lw $s1, 0($a1) #$s1 max

addi $t6, $a1, 0

addi $t0, $zero, 1

maxloop:

addi $t0, $t0, 1 #t0 loop count

addi $t6, $t6, 4 #current address

lw $t4, 0($t6)

slt $t5, $s1, $t4 #if statement in t5 s1 < t4

beq $zero, $t5, maxendif

ori $s1, $t4, 0

maxendif:

bne $t0, $a0, maxloop

sw $s1, 0($a2) #save max

jr $ra #return

avg:

lw $s2, 0($a1)

addi $t3, $a1, 0

addi $t0, $zero, 1

addi $t1, $s2, 0 #initial sum

avgloop:

addi $t0, $t0, 1 #t0 loop count

addi $t3, $t3, 4 #current address

lw $t2, 0($t3) #load current value into t2

add $t1, $t2, $t1

div $t4, $t1, $t0

beq $t4, $t4, avgend

avgend:

bne $t0, $a0, avgloop

sw $t4, 0($a2)

jr $ra #return

**Part 2:**

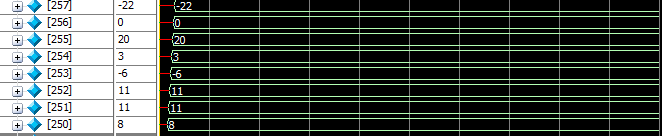
I set up 8 numbers to test the correctness of bubble sort code. It consists of positive, negative, and repeated numbers in order to prove the algorithm working without loss of generality. The waveforms were displayed in decimal. As we can see in Figure 2, it shows the initial sequence of these numbers as inputs. By applying the bubble sort code, the sorted outputs sequence was shown in Figure 3 displayed in decimal. Clearly, this is a correctly sorted sequence of input numbers. Hence, we can say the bubble sort code working correctly.

Figure 2: Initial sequence of input numbers.

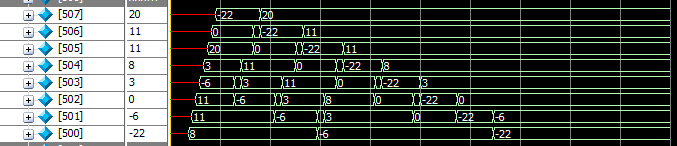


Figure 3: The sorted sequence of these numbers.

The following is the modified part of bubble sort code.

main:

ori $a0, $zero, 8 #Number of values in array

ori $a1, $zero, 1000 #Input address

ori $a2, $zero, 2000 #Output address

ori $t0, $zero, 8 #Make t0 8

sw $t0, 0($a1) #store 8 into memory

ori $t0, $zero, 11

sw $t0, 4($a1)

ori $t0, $zero, 11

sw $t0, 8($a1)

ori $t0, $zero, -6

sw $t0, 12($a1)

ori $t0, $zero, 3

sw $t0, 16($a1)

ori $t0, $zero, 20

sw $t0, 20($a1)

ori $t0, $zero, 0

sw $t0, 24($a1)

ori $t0, $zero, -22

sw $t0, 28($a1)

Appendix

I added BEQ function to the assembler code as following.

def beq\_call(command, line\_index):

imm = get\_immediate\_for\_jumps(command[3], line\_index)

imm = conv\_2\_bin(13, imm)

rs1 = conv\_2\_bin(5, registers.index(command[1]))

rs2 = conv\_2\_bin(5, registers.index(command[2]))

call\_str = imm[0] + imm[2:8] + rs2 + rs1 + '000' + imm[8:12] + imm[1] + '1100011'

call\_str = conv\_2\_hex(call\_str)

return call\_str