**Check for undercase character**

#Expects a character at a0

#will return 0 in v0 if it is an undercase char

#compare chacaracter to code of a, store result in v0 (use slt) a0 < a if true char is out of bounds

#if v0 != 0 go to exit

#compare code of z to character, store result in v0 (use slt) z < a0 if true char is out of bounds

#exit

**Get Command Code**

#gets memory location of first character in a0

#return code of instruction in v0

#code is the sum of all the characters' ascii codes

#return location of last checked character in v1

#if reached a non-lowercase character before a whitespace returns error code of 0

#store a0 in s0

#put 0 into s1

#while label

#load byte using s0 into a0

#check if it is undercase

#if not go to exit

#add a0 to s1

#increment s0

#go to while label

#exit label

#check to see if a0 is whitespace

#if white space go to exit2

#put 0 in v0

#put s0 in v1

#jump back

#exit2 label

#put s1 in v0

#put s0 in v1

#jump back

**getCmdLine**

#get pointer to first non blank character in a0

#call getCmdCode

#move v0 to s0 #s0 now has cmd code

#move v1 to s1 #s1 now has pointer to whitespace proceeding the instruction

#move s0 to a0

#move s1 to a1

#call getOpCode

#store v0 in s2 #start building the commandline using register s2

#store v1 in s3 #s3 now determines which format to use to get the rest of the command line

#R format

#find next $

#load first two bytes after $ into a0 and a1

#call registerCode procedure

#shift result left x amount

#or result with s2

#find next $

#load first two bytes after $ into a0 and a1

#call registerCode procedure

#shift result left y amount

#or result with s2

#find next $

#load first two bytes after $ into a0 and a1

#call registerCode procedure

#shift result left z amount

#or result with s2

#done

#I format

#J format

#Parenthesis format

#??

**getOpCode**

#recieves cmd code in a0, pointer to first blank char after command in a1

#returns opcode and func code (if necessary) in the correct bit placements

#returns format code in v1

# 0 = R 1 = I 2 = J 3 = parentheses instruction ...?

#use series of beq to compare a0 to various codes for the instructions

#will need to load instruction codes into registers for comparison

#example:

#addi $t0, $zer0, 297 #(code for add)

#beq $a0, $t0, getOpCode\_add

#getOpCode\_add:

#addi $v0, $zero, 0

#sll $v0, $v0, 26 #always do this to get the opcode in the correct position

#ori $v0, $v0, 0x20 #used ori with 0x20 to get the func code in with the opcode

#addi $v1, $zero, 0 $since add has R format

#j getOpCode\_exit

#getOpCode\_exit:

#jr $ra

#if code == 330

#check character with offset of -3 from a1, this would correspond to the s, l, or o in sub, lui, and ori

**loadFromStack**

#loads saved temporaries from the stack, hopefully $ra was among them if needed

loadFromStack:

lw $s0, 0($sp)

lw $s1, 4($sp)

lw $s2, 8($sp)

lw $s3, 12($sp)

lw $s4, 16($sp)

lw $s5, 20($sp)

lw $s6, 24($sp)

lw $s7, 28($sp)

add $sp, $sp, 32

jr $ra

**regToCode**

#Expectation:

#after reading a $ put next two chars into a0, a1

#using this method $s4fja is just as legitimate as $s4

#$z is just as legitimate as $zero due to checking

#similarly for other uniquely named registers

#this is a feature :)

#returns decimal version of the register through $v0

#if optimization is wanted we can compare directly to the decimal values of characters

#this was decided against originally because I thought loading things into registers

#would make the code more legible

regToCode:

#move $ra to $s0, jump back to $s0 at end, use ra for jumping w/in procedure

add $s0, $ra, $zero

#Load character codes into temp registers for comparison

#char code loaded #char

li $t0, 122 #z

li $t1, 97 #a

li $t2, 118 #v

li $t3, 116 #t

li $t4, 115 #s

li $t5, 107 #k

li $t6, 103 #g

li $t7, 102 #f

li $t8, 114 #r

beq $a0, $t0, regToCode\_firstZ #first character is z

beq $a0, $t1, regToCode\_firstA #first character is a

beq $a0, $t2, regToCode\_firstV #first character is v

beq $a0, $t3, regToCode\_firstT #first character is t

beq $a0, $t4, regToCode\_firstS #first character is s

beq $a0, $t5, regToCode\_firstK #first character is k

beq $a0, $t6, regToCode\_firstG #first character is g

beq $a0, $t7, regToCode\_firstF #first character is f

beq $a0, $t8, regToCode\_firstR #first character is r

j regToCode\_ERROR #first character is invalid

regToCode\_firstZ: #Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z

#only possible register is $zero

li $v0, 0

j regToCode\_Exit

regToCode\_firstA: #A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A

beq $a1, $t3, regToCode\_firstA\_at #second character is t

jal regToCode\_loadNumCodes

beq $a1, $t0, regToCode\_firstA\_a0 #second char is 0

beq $a1, $t1, regToCode\_firstA\_a1 #second char is 1

beq $a1, $t2, regToCode\_firstA\_a2 #second char is 2

beq $a1, $t3, regToCode\_firstA\_a3 #second char is 3

j regToCode\_ERROR #second char is invalid

regToCode\_firstA\_at:

li $v0, 1

j regToCode\_exit

regToCode\_firstA\_a0:

li $v0, 4

j regToCode\_exit

regToCode\_firstA\_a1:

li $v0, 5

j regToCode\_exit

regToCode\_firstA\_a2:

li $v0, 6

j regToCode\_exit

regToCode\_firstA\_a3:

li $v0, 7

j regToCode\_exit

regToCode\_firstV: #V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V V

jal regToCode\_loadNumCodes

beq $a1, $t0, regToCode\_firstV\_v0 #second char 0

beq $a1, $t1, regToCode\_firstV\_v1 #second char 1

j regToCode\_ERROR

regToCode\_firstV\_v0:

li $v0, 2

j regToCode\_exit

regToCode\_firstV\_v1:

li $v0, 3

j regToCode\_exit

regToCode\_firstT: #T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T T

jal regToCode\_loadNumCodes

beq $a1, $t0, regToCode\_firstT\_t0 #second char 0

beq $a1, $t1, regToCode\_firstT\_t1 #second char 1

beq $a1, $t2, regToCode\_firstT\_t2 #second char 2

beq $a1, $t3, regToCode\_firstT\_t3 #second char 3

beq $a1, $t4, regToCode\_firstT\_t4 #second char 4

beq $a1, $t5, regToCode\_firstT\_t5 #second char 5

beq $a1, $t6, regToCode\_firstT\_t6 #second char 6

beq $a1, $t7, regToCode\_firstT\_t7 #second char 7

beq $a1, $t8, regToCode\_firstT\_t8 #second char 8

beq $a1, $t9, regToCode\_firstT\_t9 #second char 9

j regToCode\_ERROR #second char invalid

regToCode\_firstT\_t0:

li $v0, 8

j regToCode\_exit

regToCode\_firstT\_t1:

li $v0, 9

j regToCode\_exit

regToCode\_firstT\_t2:

li $v0, 10

j regToCode\_exit

regToCode\_firstT\_t3:

li $v0, 11

j regToCode\_exit

regToCode\_firstT\_t4:

li $v0, 12

j regToCode\_exit

regToCode\_firstT\_t5:

li $v0, 13

j regToCode\_exit

regToCode\_firstT\_t6:

li $v0, 14

j regToCode\_exit

regToCode\_firstT\_t7:

li $v0, 15

j regToCode\_exit

regToCode\_firstT\_t8:

li $v0, 24

j regToCode\_exit

regToCode\_firstT\_t9:

li $v0, 25

j regToCode\_exit

regToCode\_firstS: #S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S

beq $a1, 112, regToCode\_firstS\_sp #second char p 112 is the decimal code for 'p'

jal regToCode\_loadNumCodes:

beq $a1, $t0, regToCode\_firstS\_s0 #second char 0

beq $a1, $t1, regToCode\_firstS\_s1 #second char 1

beq $a1, $t2, regToCode\_firstS\_s2 #second char 2

beq $a1, $t3, regToCode\_firstS\_s3 #second char 3

beq $a1, $t4, regToCode\_firstS\_s4 #second char 4

beq $a1, $t5, regToCode\_firstS\_s5 #second char 5

beq $a1, $t6, regToCode\_firstS\_s6 #second char 6

beq $a1, $t7, regToCode\_firstS\_s7 #second char 7

j regToCode\_ERROR

regToCode\_firstS\_sp:

li $v0, 29

j regToCode\_exit

regToCode\_firstS\_s0:

li $v0, 16

j regToCode\_exit

regToCode\_firstS\_s1:

li $v0, 17

j regToCode\_exit

regToCode\_firstS\_s2:

li $v0, 18

j regToCode\_exit

regToCode\_firstS\_s3:

li $v0, 19

j regToCode\_exit

regToCode\_firstS\_s4:

li $v0, 20

j regToCode\_exit

regToCode\_firstS\_s5:

li $v0, 21

j regToCode\_exit

regToCode\_firstS\_s6:

li $v0, 22

j regToCode\_exit

regToCode\_firstS\_s7:

li $v0, 23

j regToCode\_exit

regToCode\_firstK: #K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K K

jal regToCode\_loadNumCodes

beq $a1, $t0, regToCode\_firstK\_k0 #second char 0

beq $a1, $t1, regToCode\_firstK\_k1 #second char 1

j regToCode\_ERROR #second char invalid

regToCode\_firstK\_k0:

li $v0, 26

j regToCode\_exit

regToCode\_firstK\_k1:

li $v0, 27

j regToCode\_exit

regToCode\_firstG: #G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G G

li $v0, 28 #only $gp starts with g

j regToCode\_exit

regToCode\_firstF: #F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F

li $v0, 30 #only $fp starts with f

j regToCode\_exit

regToCode\_firstR: #R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R

li $v0, 31 #only $ra starts with r

j regToCode\_exit

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

regToCode\_loadNumCodes:

#char code loaded #char

li $t0, 48 #0

li $t1, 49 #1

li $t2, 50 #2

li $t3, 51 #3

li $t4, 52 #4

li $t5, 53 #5

li $t6, 54 #6

li $t7, 55 #7

li $t8, 56 #8

li $t9, 57 #9

#jump back

jr $ra

regToCode\_ERROR:

li $v0, 0xFFFFFFFF #ERROR ERROR ERROR ERROR ERROR

regToCode\_exit:

jr $s0

**saveToStack**

#saves saved temporaries to the stack, does nothing for ra, if valuable handle it outside of this procedure

saveToStack:

add $sp, $sp, -32

sw $s0, 0($sp)

sw $s1, 4($sp)

sw $s2, 8($sp)

sw $s3, 12($sp)

sw $s4, 16($sp)

sw $s5, 20($sp)

sw $s6, 24($sp)

sw $s7, 28($sp)

jr $ra