.macro System\_out\_print\_str(%str)

.data

string: .asciiz %str

.text

#Print the string

li $v0, 4

la $a0, string

syscall

.end\_macro

############################################################

.macro System\_exit

li $v0, 10 #Exit to OS

syscall

.end\_macro

############################################################

# "nextInt($t0)" - this macro reads an integer from the keyboard and store it

# in a given register

.macro nextInt(%reg)

li $v0, 5 #OS Function 5 read an integer from the keyboard

syscall #the integer typed overwrites $v0

move %reg, $v0 #Store entered integer in %reg to send back

.end\_macro

#############################################################

#Prints the value of a given register sent in as a parameter

.macro System\_out\_print\_reg(%reg)

li $v0, 1 #Print integer syscall

move $a0, %reg

syscall

.end\_macro

##########################################################

#Reads a character into a given register

.macro nextChar(%reg)

li $v0, 12 #Function 12 read a char into $v0

syscall

move %reg, $v0

.end\_macro

##########################################################

#Simulate a high level languages retrieving of an array element

# int %reg = %array [%index]

#This assumes that %index is register holding the index of the desired

#element

.macro getArrayElement(%reg, %array, %index, %DATASIZE)

#Put the value stored at array in $t0

la $t1, %array #Get the address of array and put it in $t1

#Translate the given index ($t2) to go to the actual location of the

#desired element.

#Process: multiply the given index by 4 to get the distance to "step"

#away (offset) from the base address (address of "array")

move $t2, %index

mul $t2, $t2, %DATASIZE # $s0 = $s0 \* 4 this gets the offset amount

add $t1, $t1, $t2 # Add the offset to the base address to get

# address of desired element

li $t3, %DATASIZE # Move the size (1,2 or 4) to $t3 for the sake of comparison using beq

beq $t3, 4, wordsize #if (%SIZE == 4 use lw), if (%SIZE == 2 use lh), if (%SIZE == 1 use lb)

beq $t3, 2, halfsize

beq $t3, 1, bytesize

b badsize

wordsize: lw %reg, ($t1) #Get the actual word at array

b endif

halfsize: lh %reg, ($t1) #Get the actual "half" at array

b endif

bytesize: lb %reg, ($t1) #Get the actual "byte" at array

b endif

badsize: System\_out\_print\_str("Invalid data size!")

endif:

.end\_macro

##########################################################

.macro charAt(%string, %index)

#Get character at index and leave in $v0

getArrayElement($v0, %string, %index, 1) #$v0 = %string[%index]; The "1" mean byte-sized data

.end\_macro

##########################################################

#Prints the value of a given register sent in as a parameter

.macro System\_out\_print\_char(%reg)

li $v0, 11 #Print char syscall

move $a0, %reg

syscall

.end\_macro

#############################################################

.macro toUpperCase(%string) #TO BE CONTINUED...

#This macro will convert a given string to all uppercase letters.

li $t5, 1

charAt(%string, $t5) #$v0 = %string[$t5]

move $t6, $v0 #Copy the character from $v0 to $t6

#if ($t6 is a lowercase letter) (asciiValue >= 97 && asciiValue <= 122)

#

# \*then make it uppercase (Do this by subtract 32 from the value of the lowercase letter).

#

#else

# \*leave it as is.

bgt $t6,122, notLower

blt $t6,97, notLower

isLower: sub $t6, $t6, 32 #$t6 = $t6 - 32

notLower:

System\_out\_print\_char($t6)

.end\_macro