```
##################################################################################
# Name:    Ethan West
# Class:   CS2318-253 (Assembly Language, Spring 2024)
# Subject: Assignment 3 Part 1
# Date:    04/25/2024
##################################################################################
# MIPS assembly language translation of a given C++ program that, except for the
# main function, involves "trivial" functions each of which:
# - is a leaf function
# - does not require local storage (on the stack)
# NOTES:
# - "does not require local storage" means each (leaf) function
#   -- does not need memory on the stack for local variables (including arrays)
#   -- WILL NOT use any callee-saved registers ($s0 through $s7)
# - meant as an exercise for familiarizing w/ the
#   -- basics of MIPS' function-call mechanism
#   -- how-to's of pass-by-value & pass-by-address when doing functions in MIPS
# - does NOT adhere to more-broadly-applicable function-call convention (which
#   is needed when doing functions in general, not just "trivial" functions)
# - main (being the only non-"trivial" function & an unavoidable one) will in
#   fact violate more-broadly-applicable function-call conventions
#   -- due to this, each of the functions that main calls MUST TAKE ANOMALOUS
#      CARE not to "clobber" the contents of registers that main uses & expects
#      to be preserved across calls
#   -- experiencing the pains and appreciating the undesirability of having to
#      deal with the ANOMALOUS SITUATION (due to the non-observance of any
#      function-call convention that governs caller-callee relationship) should
#      help in understanding why some function-call convention must be defined
#      and observed
##################################################################################

# Algorithm used:
# Given C++ program (Assign03P1.cpp)
##################################################################################

# Sample test run:
##################
#
# vals to do? 4
# enter an int: 1
# enter an int: 2
# enter an int: 3
# enter an int: 4
# original:
# 1 2 3 4
# backward:
# 4 3 2 1
# do more? y
# vals to do? 0
# 0 is bad, make it 1
# enter an int: 5
# original:
# 5
# backward:
# 5
# do more? y
# vals to do? 8
# 8 is bad, make it 7
# enter an int: 7
# enter an int: 6
# enter an int: 5
# enter an int: 4
# enter an int: 3
# enter an int: 2
# enter an int: 1
# original:
# 7 6 5 4 3 2 1
# backward:
# 1 2 3 4 5 6 7
# do more? n
# -- program is finished running --
##################################################################################

# int GetOneIntByVal(const char vtdPrompt[]);
# void GetOneIntByAddr(int* intVarToPutInPtr,const char prompt[]);
# void GetOneCharByAddr(char* charVarToPutInPtr, const char prompt[]);
# void ValidateInt(int* givenIntPtr, int minInt, int maxInt, const char msg[]);
# void SwapTwoInts(int* intPtr1, int* intPtr2);
# void ShowIntArray(const int array[], int size, const char label[]);
#
#int main()
#{
        .text
        .globl main
main:
#   int intArr[7];
#   int valsToDo;
#   char reply;
#   char vtdPrompt[] = "vals to do? ";
#   char entIntPrompt[] = "enter some int: ";
#   char adjMsg[] = " is bad, make it ";
#   char origLab[] = "original:\n";
#   char backLab[] = "backward:\n";
#   char dmPrompt[] = "do more? ";
#   int i, j;
#################
# Register Usage:
#################
# $t0: register holder for a value
# $t1: i
# $t2: j
#################
```

```
       addiu $sp, $sp, -114
       j StrInitCode   # clutter-reduction jump (string initialization)
endStrInit:
#   do
#   {
begWBodyM1:
       li $a0, '\n'
       li $v0, 11
       syscall # '\n' to offset effects of syscall #12 drawback
#      valsToDo = GetOneIntByVal(vtdPrompt);

####################(3)####################
       addi $a0, $sp, 73
       jal GetOneIntByVal
       sw $v0, 54($sp)




#      ValidateInt(&valsToDo, 1, 7, adjMsg);

####################(4)####################
       addi $a0, $sp, 54
       li $a1, 1
       li $a2, 7
       addi $a3, $sp, 0




       jal ValidateInt
#      for (i = valsToDo; i > 0; --i)

####################(1)####################
       lw $t1, 54($sp)

       j FTestM1
begFBodyM1:
#         if (i % 2) // i is odd
       andi $t0, $t1, 0x00000001
       beqz $t0, ElseI1
#         intArr[valsToDo - i] = GetOneIntByVal(entIntPrompt);

####################(8)####################
       addi $a0, $sp, 39
       jal GetOneIntByVal
       #addi $t0, $sp, 54
       lw $t0, 54($sp)
       sub $t0, $t0, $t1
       sll $t0, $t0, 2

```

```
        #t0 vals-i*4
        addi $a0, $sp, 86
        add $t0, $a0, $t0
        sw $v0, 0($t0)

        j endI1
#          else // i is even
ElseI1:
#             GetOneIntByAddr(intArr + valsToDo - i, entIntPrompt);

####################(7)####################
        lw $a0, 54($sp)
        sub $a0, $a0, $t1
        sll $a0, $a0, 2
        add $a0, $a0, $sp
        addi $a0, $a0, 86
        addi $a1, $sp, 39
        jal GetOneIntByAddr


endI1:
        addi $t1, $t1, -1
FTestM1:
        bgtz $t1, begFBodyM1
#       ShowIntArray(intArr, valsToDo, origLab);

####################(3)####################
        addi $a0, $sp, 86
        lw $a1, 54($sp)
        addi $a2, $sp, 62



        jal ShowIntArray

#       for (i = 0, j = valsToDo - 1; i < j; ++i, --j)
####################(3)####################
        li $t1, 0
        lw $t0, 54($sp)
        addi $t2, $t0, -1

        j FTestM2
begFBodyM2:
#          SwapTwoInts(intArr + i, intArr + j);

####################(5)####################
        addi $t0, $sp, 86
        sll $a0, $t1, 2
        add $a0, $t0, $a0
        sll $a1, $t2, 2
        add $a1, $t0, $a1
```

```


       jal SwapTwoInts

       addi $t1, $t1, 1
       addi $t2, $t2, -1
FTestM2:
       blt $t1, $t2, begFBodyM2
#      ShowIntArray(intArr, valsToDo, backLab);

####################(3)####################
       addi $a0, $sp, 86
       lw $a1, 54($sp)
       addi $a2, $sp, 28


       jal ShowIntArray

#      GetOneCharByAddr(&reply, dmPrompt);

####################(2)####################
       addi $a0, $sp, 58
       addi $a1, $sp, 18


       jal GetOneCharByAddr
#   }
#   while (reply != 'n' && reply != 'N');

####################(1)####################
       lw $v1, 58($sp)

       li $t0, 'n'
       beq $v1, $t0, endWhileM1
       li $t0, 'N'
       bne $v1, $t0, begWBodyM1
endWhileM1:    # extra helper label added

#   return 0;
#}
       addiu $sp, $sp, 114
       li $v0, 10
       syscall

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

#int GetOneIntByVal(const char prompt[])
#{
GetOneIntByVal:
#   int oneInt;
#   cout << prompt;
```

```
        li $v0, 4
        syscall
#   cin >> oneInt;
        li $v0, 5
        syscall
#   return oneInt;
#}
        jr $ra

##################################################################################
#void GetOneIntByAddr(int* intVarToPutInPtr, const char prompt[])
#{
GetOneIntByAddr:
#   cout << prompt;
        move $t0, $a0   # $t0 has saved copy of $a0 as received
        move $a0, $a1
        li $v0, 4
        syscall
#   cin >> *intVarToPutInPtr;
        li $v0, 5
        syscall
        sw $v0, 0($t0)
#}
        jr $ra

##################################################################################
#void ValidateInt(int* givenIntPtr, int minInt, int maxInt, const char msg[])
#{
ValidateInt:
#################
# Register Usage:
#################
# $t0: copy of arg1 ($a0) as received
# $v1: value loaded from mem (*givenIntPtr)
#################
        move $t0, $a0   # $t0 has saved copy of $a0 as received
#   if (*givenIntPtr < minInt)
#   {
        lw $v1, 0($t0)  # $v1 has *givenIntPtr
        bge $v1, $a1, ElseVI1
#       cout << *givenIntPtr << msg << minInt << endl;
        move $a0, $v1
        li $v0, 1
        syscall
        move $a0, $a3
        li $v0, 4
        syscall
        move $a0, $a1
        li $v0, 1
```

```
              syscall
              li $a0, '\n'
              li $v0, 11
              syscall
#             *givenIntPtr = minInt;
              sw $a1, 0($t0)
              j endIfVI1
#      }
#      else
#      {
ElseVI1:
#             if (*givenIntPtr > maxInt)
#             {
              ble $v1, $a2, endIfVI2
#                cout << *givenIntPtr << msg << maxInt << endl;
              move $a0, $v1
              li $v0, 1
              syscall
              move $a0, $a3
              li $v0, 4
              syscall
              move $a0, $a2
              li $v0, 1
              syscall
              li $a0, '\n'
              li $v0, 11
              syscall
#                *givenIntPtr = maxInt;
              sw $a2, 0($t0)
#             }
endIfVI2:
#      }
endIfVI1:
#}
              jr $ra

################################################################################
#void ShowIntArray(const int array[], int size, const char label[])
#{
ShowIntArray:
#################
# Register Usage:
#################
# $t0: copy of arg1 ($a0) as received
# $a3: k
# $v1: value loaded from mem (*givenIntPtr)
#################
              move $t0, $a0    # $t0 has saved copy of $a0 as received
#      cout << label;
              move $a0, $a2
```

```
        li $v0, 4
        syscall
#   int k = size;
        move $a3, $a1
        j WTestSIA
#   while (k > 0)
#   {
begWBodySIA:
#       cout << array[size - k] << ' ';
        sub $v1, $a1, $a3   # $v1 gets (size - k)
        sll $v1, $v1, 2 # $v1 now has 4*(size - k)
        add $v1, $v1, $t0   # $v1 now has &array[size - k]
        lw $a0, 0($v1)  # $a0 has array[size - k]
        li $v0, 1
        syscall
        li $a0, ' '
        li $v0, 11
        syscall
#       --k;
        addi $a3, $a3, -1
#   }
WTestSIA:
        bgtz $a3, begWBodySIA
#   cout << endl;
        li $a0, '\n'
        li $v0, 11
        syscall
#}
        jr $ra

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

#void SwapTwoInts(int* intPtr1, int* intPtr2)
#{
SwapTwoInts:
#################
# Register Usage:
#################
# $t0: holder for saving pntr of 1st arg ($a0) recieved
#################
#   int temp = *intPtr1;
#  *intPtr1 = *intPtr2;
#   *intPtr2 = temp;

###################(4)###################
        lw $t0, 0($a0)
        lw $t3, 0($a1)
        sw $t3, 0($a0)
        sw $t0, 0($a1)


```

```


#
       jr $ra

##################################################################################
#void GetOneCharByAddr(char* charVarToPutInPtr, const char prompt[])
#{
GetOneCharByAddr:
#################
# Register Usage:
#################
# (fill in where applicable)
#################
#   cout << prompt;
#   cin >> *charVarToPutInPtr;

####################(7)####################
       move $t0, $a0
       move $a0, $a1
       li $v0, 4
       syscall
       li $v0, 12
       syscall
       sw $v0, 0($t0)
#}
       jr $ra

##################################################################################
StrInitCode:
#################
# "bulky & boring" string-initializing code move off of main stage
##################################################################################
       li $t0, 'd'
       sb $t0, 18($sp)
       li $t0, 'o'
       sb $t0, 19($sp)
       li $t0, ' '
       sb $t0, 20($sp)
       li $t0, 'm'
       sb $t0, 21($sp)
       li $t0, 'o'
       sb $t0, 22($sp)
       li $t0, 'r'
       sb $t0, 23($sp)
       li $t0, 'e'
       sb $t0, 24($sp)
       li $t0, '?'
```