```
1: ###############################################################################
2: # Name:    Ethan West
3: # Class:   CS2318-253 (Assembly Language, Spring 2024)
4: # Subject: Assignment 3 Part 1
5: # Date:    04/25/2024
6: ###############################################################################
7: # MIPS assembly language translation of a given C++ program that, except for the
8: # main function, involves "trivial" functions each of which:
9: # - is a leaf function
10: # - does not require local storage (on the stack)
11: # NOTES:
12: # - "does not require local storage" means each (leaf) function
13: #   -- does not need memory on the stack for local variables (including arrays)
14: #   -- WILL NOT use any callee-saved registers ($s0 through $s7)
15: # - meant as an exercise for familiarizing w/ the
16: #   -- basics of MIPS' function-call mechanism
17: #   -- how-to's of pass-by-value & pass-by-address when doing functions in MIPS
18: # - does NOT adhere to more-broadly-applicable function-call convention (which
19: #   is needed when doing functions in general, not just "trivial" functions)
20: # - main (being the only non-"trivial" function & an unavoidable one) will in
21: #   fact violate more-broadly-applicable function-call conventions
22: #   -- due to this, each of the functions that main calls MUST TAKE ANOMALOUS
23: #      CARE not to "clobber" the contents of registers that main uses & expects
24: #      to be preserved across calls
25: #   -- experiencing the pains and appreciating the undesirability of having to
26: #      deal with the ANOMALOUS SITUATION (due to the non-observance of any
27: #      function-call convention that governs caller-callee relationship) should
28: #      help in understanding why some function-call convention must be defined
29: #      and observed
30: ###############################################################################

31: # Algorithm used:
32: # Given C++ program (Assign03P1.cpp)
33: ###############################################################################

34: # Sample test run:
35: ##################
36: #
37: # vals to do? 4
38: # enter an int: 1
39: # enter an int: 2
40: # enter an int: 3
41: # enter an int: 4
42: # original:
43: # 1 2 3 4
44: # backward:
45: # 4 3 2 1
46: # do more? y
47: # vals to do? 0
48: # 0 is bad, make it 1
49: # enter an int: 5
```

```
50: # original:
51: # 5
52: # backward:
53: # 5
54: # do more? y
55: # vals to do? 8
56: # 8 is bad, make it 7
57: # enter an int: 7
58: # enter an int: 6
59: # enter an int: 5
60: # enter an int: 4
61: # enter an int: 3
62: # enter an int: 2
63: # enter an int: 1
64: # original:
65: # 7 6 5 4 3 2 1
66: # backward:
67: # 1 2 3 4 5 6 7
68: # do more? n
69: # -- program is finished running --
70: ################################################################################

71: # int GetOneIntByVal(const char vtdPrompt[]);
72: # void GetOneIntByAddr(int* intVarToPutInPtr,const char prompt[]);
73: # void GetOneCharByAddr(char* charVarToPutInPtr, const char prompt[]);
74: # void ValidateInt(int* givenIntPtr, int minInt, int maxInt, const char msg[]);
75: # void SwapTwoInts(int* intPtr1, int* intPtr2);
76: # void ShowIntArray(const int array[], int size, const char label[]);
77: #
78: #int main()
79: #{
80:         .text
81:         .globl main
82: main:
83: #   int intArr[7];
84: #   int valsToDo;
85: #   char reply;
86: #   char vtdPrompt[] = "vals to do? ";
87: #   char entIntPrompt[] = "enter some int: ";
88: #   char adjMsg[] = " is bad, make it ";
89: #   char origLab[] = "original:\n";
90: #   char backLab[] = "backward:\n";
91: #   char dmPrompt[] = "do more? ";
92: #   int i, j;
93: ##################
94: # Register Usage:
95: ##################
96: # $t0: register holder for a value
97: # $t1: i
98: # $t2: j
99: ##################
```

```
100:        addiu $sp, $sp, -114
101:        j StrInitCode   # clutter-reduction jump (string initialization)
102: endStrInit:
103: #   do
104: #   {
105: begWBodyM1:
106:        li $a0, '\n'
107:        li $v0, 11
108:        syscall # '\n' to offset effects of syscall #12 drawback
109: #      valsToDo = GetOneIntByVal(vtdPrompt);
110:
111: ####################(3)####################
112:        addi $a0, $sp, 73
113:        jal GetOneIntByVal
114:        sw $v0, 54($sp)
115:
116:
117:
118:
119: #      ValidateInt(&valsToDo, 1, 7, adjMsg);
120:
121: ####################(4)####################
122:        addi $a0, $sp, 54
123:        li $a1, 1
124:        li $a2, 7
125:        addi $a3, $sp, 0
126:
127:
128:
129:
130:        jal ValidateInt
131: #      for (i = valsToDo; i > 0; --i)
132:
133: ####################(1)####################
134:        lw $t1, 54($sp)
135:
136:        j FTestM1
137: begFBodyM1:
138: #         if (i % 2) // i is odd
139:        andi $t0, $t1, 0x00000001
140:        beqz $t0, ElseI1
141: #         intArr[valsToDo - i] = GetOneIntByVal(entIntPrompt);
142:
143: ####################(8)####################
144:        addi $a0, $sp, 39
145:        jal GetOneIntByVal
146:        #addi $t0, $sp, 54
147:        lw $t0, 54($sp)
148:        sub $t0, $t0, $t1
149:        sll $t0, $t0, 2
150:
```

```
151:        #t0 vals-i*4
152:        addi $a0, $sp, 86
153:        add $t0, $a0, $t0
154:        sw $v0, 0($t0)
155:
156:        j endI1
157: #         else // i is even
158: ElseI1:
159: #            GetOneIntByAddr(intArr + valsToDo - i, entIntPrompt);
160:
161: ####################(7)####################
162:        lw $a0, 54($sp)
163:        sub $a0, $a0, $t1
164:        sll $a0, $a0, 2
165:        add $a0, $a0, $sp
166:        addi $a0, $a0, 86
167:        addi $a1, $sp, 39
168:        jal GetOneIntByAddr
169:
170:
171: endI1:
172:        addi $t1, $t1, -1
173: FTestM1:
174:        bgtz $t1, begFBodyM1
175: #      ShowIntArray(intArr, valsToDo, origLab);
176:
177: ####################(3)####################
178:        addi $a0, $sp, 86
179:        lw $a1, 54($sp)
180:        addi $a2, $sp, 62
181:
182:
183:
184:        jal ShowIntArray
185:
186: #      for (i = 0, j = valsToDo - 1; i < j; ++i, --j)
187: ####################(3)####################
188:        li $t1, 0
189:        lw $t0, 54($sp)
190:        addi $t2, $t0, -1
191:
192:        j FTestM2
193: begFBodyM2:
194: #         SwapTwoInts(intArr + i, intArr + j);
195:
196: ####################(5)####################
197:        addi $t0, $sp, 86
198:        sll $a0, $t1, 2
199:        add $a0, $t0, $a0
200:        sll $a1, $t2, 2
201:        add $a1, $t0, $a1
```

```
202:
203:
204:        jal SwapTwoInts
205:
206:        addi $t1, $t1, 1
207:        addi $t2, $t2, -1
208: FTestM2:
209:        blt $t1, $t2, begFBodyM2
210: #      ShowIntArray(intArr, valsToDo, backLab);
211:
212: ####################(3)####################
213:        addi $a0, $sp, 86
214:        lw $a1, 54($sp)
215:        addi $a2, $sp, 28
216:
217:
218:        jal ShowIntArray
219:
220: #      GetOneCharByAddr(&reply, dmPrompt);
221:
222: ####################(2)####################
223:        addi $a0, $sp, 58
224:        addi $a1, $sp, 18
225:
226:
227:        jal GetOneCharByAddr
228: #   }
229: #   while (reply != 'n' && reply != 'N');
230:
231: ####################(1)####################
232:        lw $v1, 58($sp)
233:
234:        li $t0, 'n'
235:        beq $v1, $t0, endWhileM1
236:        li $t0, 'N'
237:        bne $v1, $t0, begWBodyM1
238: endWhileM1:    # extra helper label added
239:
240: #   return 0;
241: #}
242:        addiu $sp, $sp, 114
243:        li $v0, 10
244:        syscall
245:
246: ################################################################################

247: #int GetOneIntByVal(const char prompt[])
248: #{
249: GetOneIntByVal:
250: #   int oneInt;
251: #   cout << prompt;
```

```
252:        li $v0, 4
253:        syscall
254: #   cin >> oneInt;
255:        li $v0, 5
256:        syscall
257: #   return oneInt;
258: #}
259:        jr $ra
260:
261: ################################################################################

262: #void GetOneIntByAddr(int* intVarToPutInPtr, const char prompt[])
263: #{
264: GetOneIntByAddr:
265: #   cout << prompt;
266:        move $t0, $a0   # $t0 has saved copy of $a0 as received
267:        move $a0, $a1
268:        li $v0, 4
269:        syscall
270: #   cin >> *intVarToPutInPtr;
271:        li $v0, 5
272:        syscall
273:        sw $v0, 0($t0)
274: #}
275:        jr $ra
276:
277: ################################################################################

278: #void ValidateInt(int* givenIntPtr, int minInt, int maxInt, const char msg[])
279: #{
280: ValidateInt:
281: ##################
282: # Register Usage:
283: ##################
284: # $t0: copy of arg1 ($a0) as received
285: # $v1: value loaded from mem (*givenIntPtr)
286: ##################
287:        move $t0, $a0   # $t0 has saved copy of $a0 as received
288: #   if (*givenIntPtr < minInt)
289: #   {
290:        lw $v1, 0($t0)  # $v1 has *givenIntPtr
291:        bge $v1, $a1, ElseVI1
292: #      cout << *givenIntPtr << msg << minInt << endl;
293:        move $a0, $v1
294:        li $v0, 1
295:        syscall
296:        move $a0, $a3
297:        li $v0, 4
298:        syscall
299:        move $a0, $a1
300:        li $v0, 1
```

```
301:        syscall
302:        li $a0, '\n'
303:        li $v0, 11
304:        syscall
305: #      *givenIntPtr = minInt;
306:        sw $a1, 0($t0)
307:        j endIfVI1
308: #   }
309: #   else
310: #   {
311: ElseVI1:
312: #      if (*givenIntPtr > maxInt)
313: #      {
314:        ble $v1, $a2, endIfVI2
315: #         cout << *givenIntPtr << msg << maxInt << endl;
316:        move $a0, $v1
317:        li $v0, 1
318:        syscall
319:        move $a0, $a3
320:        li $v0, 4
321:        syscall
322:        move $a0, $a2
323:        li $v0, 1
324:        syscall
325:        li $a0, '\n'
326:        li $v0, 11
327:        syscall
328: #         *givenIntPtr = maxInt;
329:        sw $a2, 0($t0)
330: #      }
331: endIfVI2:
332: #   }
333: endIfVI1:
334: #}
335:        jr $ra
336:
337: ################################################################################

338: #void ShowIntArray(const int array[], int size, const char label[])
339: #{
340: ShowIntArray:
341: ##################
342: # Register Usage:
343: ##################
344: # $t0: copy of arg1 ($a0) as received
345: # $a3: k
346: # $v1: value loaded from mem (*givenIntPtr)
347: ##################
348:        move $t0, $a0   # $t0 has saved copy of $a0 as received
349: #   cout << label;
350:        move $a0, $a2
```

```
351:        li $v0, 4
352:        syscall
353: #   int k = size;
354:        move $a3, $a1
355:        j WTestSIA
356: #   while (k > 0)
357: #   {
358: begWBodySIA:
359: #      cout << array[size - k] << ' ';
360:        sub $v1, $a1, $a3   # $v1 gets (size - k)
361:        sll $v1, $v1, 2 # $v1 now has 4*(size - k)
362:        add $v1, $v1, $t0   # $v1 now has &array[size - k]
363:        lw $a0, 0($v1)  # $a0 has array[size - k]
364:        li $v0, 1
365:        syscall
366:        li $a0, ' '
367:        li $v0, 11
368:        syscall
369: #      --k;
370:        addi $a3, $a3, -1
371: #   }
372: WTestSIA:
373:        bgtz $a3, begWBodySIA
374: #   cout << endl;
375:        li $a0, '\n'
376:        li $v0, 11
377:        syscall
378: #}
379:        jr $ra
380:
381: ################################################################################

382: #void SwapTwoInts(int* intPtr1, int* intPtr2)
383: #{
384: SwapTwoInts:
385: ##################
386: # Register Usage:
387: ##################
388: # $t0: holder for saving pntr of 1st arg ($a0) recieved
389: ##################
390: #   int temp = *intPtr1;
391: #  *intPtr1 = *intPtr2;
392: #   *intPtr2 = temp;
393:
394: ####################(4)####################
395:        lw $t0, 0($a0)
396:        lw $t3, 0($a1)
397:        sw $t3, 0($a0)
398:        sw $t0, 0($a1)
399:
400:
```

```
401:
402:
403: #
404:         jr $ra
405:
406: ################################################################################

407: #void GetOneCharByAddr(char* charVarToPutInPtr, const char prompt[])
408: #{
409: GetOneCharByAddr:
410: ##################
411: # Register Usage:
412: ##################
413: # (fill in where applicable)
414: ##################
415: #   cout << prompt;
416: #   cin >> *charVarToPutInPtr;
417:
418: ####################(7)####################
419:         move $t0, $a0
420:         move $a0, $a1
421:         li $v0, 4
422:         syscall
423:         li $v0, 12
424:         syscall
425:         sw $v0, 0($t0)
426: #}
427:         jr $ra
428:
429: ################################################################################

430: StrInitCode:
431: ##################
432: # "bulky & boring" string-initializing code move off of main stage
433: ################################################################################

434:         li $t0, 'd'
435:         sb $t0, 18($sp)
436:         li $t0, 'o'
437:         sb $t0, 19($sp)
438:         li $t0, ' '
439:         sb $t0, 20($sp)
440:         li $t0, 'm'
441:         sb $t0, 21($sp)
442:         li $t0, 'o'
443:         sb $t0, 22($sp)
444:         li $t0, 'r'
445:         sb $t0, 23($sp)
446:         li $t0, 'e'
447:         sb $t0, 24($sp)
448:         li $t0, '?'
```