

```
1: #####
2: # Ethan West, CS 2318-253, Assignment 2 Part 1 Program B #
3: #####
4: # MIPS assembly program to carry out the following tasks, in order:
5: # - Allocate a global array (i.e., space in the data segment) enough
6: #   for storing 4 integers and initialize the array (from 1st to 4th
7: #   element) with 888, 111, 333 and 222 at the same time (i.e., DON'T
8: #   first allocate uninitialized space for array and later write code
9: #   to put the values into array).
10: #
11: # - Display a labeled output about the array's initial contents
12: #   (in the order from 1st to 4th element).
13: #
14: # - IMPORTANT (for the purpose of this exercise):
15: #   - You are to load the values of the array elements from memory
16: #     and use those values to generate the labeled output. (You are
17: #     not to simply display a hard-coded string(s) and values
18: #     showing what the contents of the array should look like.)
19: #
20: # - Re-order the values in the array so that the contents of the array
21: #   in memory (from 1st to 4th element) eventually becomes 333, 222, 888
22: #   and 111, using the following operations in the order listed (to not
23: #   defeat the goals of this exercise, you must NOT change the specified
24: #   operations and order, even if doing so will accomplish the same
25: #   effect more efficiently):
26: #
27: #   - Swap the contents in memory of the 1st and 3rd elements of the
28: #     initial array.
29: #
30: #   - NOTE: Contents of the array in memory (from 1st to 4th
31: #     element) after this first swapping operation should be 333,
32: #     111, 888 and 222.
33: #   - Swap the contents in memory of the 2nd and 4th elements of the
34: #     array that results after the preceding first swap.
35: #
36: #   - NOTE: Contents of the array in memory (from 1st to 4th
37: #     element) after this second swapping operation should
38: #     now and finally be 333, 222, 888 and 111.
39: #
40: # - IMPORTANT (for the purpose of this exercise):
41: #   - When performing each of the three swap operations above, you can
42: #     re-use (where expedient) the array's base address in register
43: #     (loaded when performing the display of the array's initial
44: #     contents) but you MUST re-load the values of the associated
45: #     array elements fresh from memory (i.e., assuming no knowledge
46: #     that certain values might have already existed in some registers
47: #     due to prior operations).
48: #
49: # - Display a labeled output about the array's contents (in the order
50: #   from 4th to 1st element) after the 2 swapping operations above.
51: #
```

```

52: #      - NOTE: The contents of the array's elements are to appear (when
53: #      displayed) in the order from 4th to 1st element and not from 1st
54: #      to 4th element.
55: #
56: #      - (IN CASE YOU WONDER: Order in which the eventual contents of the
57: #      array should appear in this output --> 111, 888, 222 and 333.)
58: #
59: #      - IMPORTANT (for the purpose of this exercise):
60: #      - When displaying the after-swap labeled output, you can re-use
61: #      the array's base address in register (loaded when performing
62: #      prior operations) but you MUST re-load the values of the array
63: #      elements fresh from memory (i.e., assuming no knowledge that
64: #      certain values might have already existed in some registers
65: #      due to prior operations).
66: #
67: #      - CAUTION:
68: #      - Too many past students regretted having points taken off for
69: #      not labeling output.
70: #
71: ##### data segment #####
72:
73:      .data
74: intArr:      .word 888, 111, 333, 222      # global int array of size 4 initialized
75:              # t0 888, 111, 333, 222 (from 1 - 4)
76: indexZero:   .asciiz "[0]: "              # String helpers for use in labeling
77: indexOne:    .asciiz "[1]: "
78: indexTwo:    .asciiz "[2]: "
79: indexThree:  .asciiz "[3]: "
80:
81:      .text
82:      .globl main
83: main:
84:      la $t0, intArr      # $t0 has adress of intArr
85:
86:      # BEGIN_(Printing of initial order of array)
87:      lw $t1, 0($t0)      # $t1 has adress of intArr[0]
88:      li $v0, 4
89:      la $a0, indexZero   # Print "[0]: " lable
90:      syscall
91:      li $v0, 1
92:      move $a0, $t1       # Print intArr[0]
93:      syscall
94:
95:      li $v0, 11          # new line
96:      li $a0, '\n'
97:      syscall
98:
99:      lw $t2, 4($t0)      # $t2 has adress of intArr[1]
100:     li $v0, 4
101:     la $a0, indexOne     # Print "[1]: " lable
102:     syscall

```

```
103:         li $v0, 1
104:         move $a0, $t2           # Print intArr[1]
105:         syscall
106:
107:         li $v0, 11              # new line
108:         li $a0, '\n'
109:         syscall
110:
111:         lw $t3, 8($t0)          # $t3 has adress of intArr[2]
112:         li $v0, 4
113:         la $a0, indexTwo        # Print "[2]: " lable
114:         syscall
115:         li $v0, 1
116:         move $a0, $t3           # Print intArr[2]
117:         syscall
118:
119:         li $v0, 11              # new line
120:         li $a0, '\n'
121:         syscall
122:
123:         lw $t4, 12($t0)         # $t4 has adress of intArr[3]
124:         li $v0, 4
125:         la $a0, indexThree      # Print "[3]: " lable
126:         syscall
127:         li $v0, 1
128:         move $a0, $t4           # Print intArr[3]
129:         syscall
130:         # END_(Printing of initial order of array)
131:
132:         li $v0, 11 # new line
133:         li $a0, '\n'
134:         syscall
135:
136:         # BEGIN_(Swap 1st and 3rd elements)
137:         sw $t1, 8($t0)
138:         sw $t3, 0($t0)
139:         # END_(Swap 1st and 3rd elements)
140:
141:         # BEGIN_(Swap 2nd and 4th elements)
142:         sw $t4, 4($t0)
143:         sw $t2, 12($t0)
144:         # END_(Swap 2nd and 4th elements)
145:
146:         li $v0, 11 # new line
147:         li $a0, '\n'
148:         syscall
149:
150:         # BEGIN_(Printing of final order of array)
151:         lw $t4, 12($t0)         # $t4 has adress of intArr[3]
152:         li $v0, 4
153:         la $a0, indexThree      # Print "[3]: " lable
```

```
154:          syscall
155:          li $v0, 1
156:          move $a0, $t4          # Print intArr[3]
157:          syscall
158:
159:          li $v0, 11  # new line
160:          li $a0, '\n'
161:          syscall
162:
163:          lw $t3, 8($t0)          # $t3 has adress of intArr[2]
164:          li $v0, 4
165:          la $a0, indexTwo        # Print "[2]: " lable
166:          syscall
167:          li $v0, 1
168:          move $a0, $t3          # Print intArr[2]
169:          syscall
170:
171:          li $v0, 11  # new line
172:          li $a0, '\n'
173:          syscall
174:
175:          lw $t2, 4($t0)          # $t2 has adress of intArr[1]
176:          li $v0, 4
177:          la $a0, indexOne        # Print "[1]: " lable
178:          syscall
179:          li $v0, 1
180:          move $a0, $t2          # Print intArr[1]
181:          syscall
182:
183:          li $v0, 11          # new line
184:          li $a0, '\n'
185:          syscall
186:
187:          lw $t1, 0($t0)          # $t1 has adress of intArr[0]
188:          li $v0, 4
189:          la $a0, indexZero       # Print "[0]: " lable
190:          syscall
191:          li $v0, 1
192:          move $a0, $t1          # Print intArr[0]
193:          syscall
194:
195:          li $v0, 11          # new line
196:          li $a0, '\n'
197:          syscall
198:          # END_(Printing of final order of array)
199:
200:          li $v0, 10          # graceful exit
201:          syscall
202: #####
203:
204:
```

205:

206: