```
1:  .data
2:
3:      # Bitmap Frame Buffer
4:      frameBuffer: .space 0x80000
5:
6:      # Delcaring a newline
7:      newline: .asciiz "\n"
8:
9:
10:     # Declaing GameIntro
11:     GameIntro1: .asciiz "\n\n******* Hello Hello, This Is A Math Game Where You Need T
o Find The Root Of The Polynomial Equations!! *******\n"
12:         GameIntro2: .asciiz "\tRules Of The Game:\n"
13:         GameIntro3: .asciiz "\t-> You Have To Solve All Of The 7 Questions In Order To
 Win The Prize! (Two Attempts For Each Question)\n"
14:         GameIntro4: .asciiz "\t-> Questions' Level Are As Follow:\n"
15:         GameIntro5: .asciiz "\t\t. Q's 1 - 3 Are Level Easy. (Prize: “Bronze Medal”)\n
"
16:         GameIntro6: .asciiz "\t\t. Q's 4 - 6 Are Level Medium. (Prize: “Silver Medal”)
\n"
17:         GameIntro7: .asciiz "\t\t. Q 7 is Level Hard. (Prize: “Gold Medal”)\n\n"
18:         GameIntro8: .asciiz "\t-> You Can Opt Out Whenever You Win One Of The Medals,
Or Continue Unit you Win Gold Medal\n"
19:         GameIntro9: .asciiz "\n\t\tGOOD LUCK!!!!!\n\n"
20:
21:
22:         # Declaring Game Map
23:         # gameMaps1()
24:         gameMaps1.1: .asciiz "                                                      EASY
                                    \n"
25:         gameMaps1.2: .asciiz "                 --------------                --------
-----                --------------\n"
26:
27:     # gameMaps2()
28:     gameMaps2.1: .asciiz "                  --------------                -----------
-                --------------         \\\n"
29:         gameMaps2.2: .asciiz "
                                          |\n"
30:         gameMaps2.3: .asciiz "
                                    “Bronze Medal”\n"
31:         gameMaps2.4: .asciiz "                                                  MEDIU
M                                         |\n"
32:         gameMaps2.5: .asciiz "                 --------------                --------
-----                --------------         /\n"
33:
34:         # gameMaps3()
35:     gameMaps3.1: .asciiz "        /        --------------                -----------
-                --------------    \n"
36:     gameMaps3.2: .asciiz "       |\n"
37:         gameMaps3.3: .asciiz "“Silver Medal”\n"
38:         gameMaps3.4: .asciiz "       |                                          HARD
\n"
39:         gameMaps3.5: .asciiz "        \\                                        -------
------              \n"
40:
```

```
41:         # gameMap()
42:         gameMap.1: .asciiz "         -------|  Question 1  |--------------|  Question
2  |--------------|  Question 3  |--------\n"
43:     gameMap.2: .asciiz "         -------|  Question 6  |--------------|  Question 5  |
--------------|  Question 4  |--------\n"
44:     gameMap.3: .asciiz "         ------------------------------------|  Question 7  |
------------------------------ “Gold Medal”\n"
45:     gameMap.4: .asciiz "                                                --------------
              \n"
46:
47:     # Game Maps Points
48:         # gameMap1(10 points)
49:         gameMap1.1: .asciiz "         -------|  "
50:         gameMap1.2: .asciiz " points   |--------------|  Question 2  |--------------|
 Question 3  |--------\n"
51:     gameMap1.3: .asciiz "         -------|  Question 6  |--------------|  Question 5
|--------------|  Question 4  |--------\n"
52:     gameMap1.4: .asciiz "         ------------------------------------|  Question 7
|------------------------------ “Gold Medal”\n"
53:     gameMap1.5: .asciiz "                                                --------------
               \n"
54:
55:     # gameMap2(10 points, 20 points)
56:     gameMap2.1: .asciiz "         -------|  "
57:         gameMap2.2: .asciiz " points   |--------------|  "
58:         gameMap2.3: .asciiz " points   |--------------|  Question 3  |--------\n"
59:         gameMap2.4: .asciiz "         -------|  Question 6  |--------------|  Question
 5  |--------------|  Question 4  |--------\n"
60:         gameMap2.5: .asciiz "         ------------------------------------|  Question
 7  |------------------------------ “Gold Medal”\n"
61:         gameMap2.6: .asciiz "                                                 ----------
----               \n"
62:
63:         # gameMap3(10 points, 20 points, 30 points)
64:         gameMap3.1: .asciiz "         -------|  "
65:         gameMap3.2: .asciiz " points   |--------------|  "
66:         gameMap3.3: .asciiz " points   |--------------|  "
67:         gameMap3.4: .asciiz " points   |--------\n"
68:         gameMap3.5: .asciiz "         -------|  Question 6  |--------------|  Question
 5  |--------------|  Question 4  |--------\n"
69:         gameMap3.6: .asciiz "         ------------------------------------|  Question
 7  |------------------------------ “Gold Medal”\n"
70:         gameMap3.7: .asciiz "                                                 ----------
----               \n"
71:
72:         # gameMap4(10 points, 20 points, 30 points, 40 points)
73:         gameMap4.1: .asciiz "         -------|  "
74:         gameMap4.2: .asciiz " points   |--------------|  "
75:         gameMap4.3: .asciiz " points   |--------------|  "
76:         gameMap4.4: .asciiz " points   |--------\n"
77:         gameMap4.5: .asciiz "         -------|  Question 6  |--------------|  Question
 5  |--------------|  "
78:         gameMap4.6: .asciiz " points   |--------\n"
79:         gameMap4.7: .asciiz "         ------------------------------------|  Question
 7  |------------------------------ “Gold Medal”\n"
```

```
80:         gameMap4.8: .asciiz "                                                  ----------
----               \n"
81:
82:         # gameMap5(10 points, 20 points, 30 points, 40 points, 50 points)
83:         gameMap5.1: .asciiz "         -------|  "
84:         gameMap5.2: .asciiz " points   |--------------|  "
85:         gameMap5.3: .asciiz " points   |--------------|  "
86:         gameMap5.4: .asciiz " points   |--------\n"
87:         gameMap5.5: .asciiz "         -------|  Question 6  |--------------|  "
88:         gameMap5.6: .asciiz " points   |--------------|  "
89:         gameMap5.7: .asciiz " points   |--------\n"
90:         gameMap5.8: .asciiz "         ------------------------------------|  Question
 7  |------------------------------ “Gold Medal”\n"
91:         gameMap5.9: .asciiz "                                                  ----------
----               \n"
92:
93:         # gameMap6(10 points, 20 points, 30 points, 40 points, 50 points, 60 points)
94:         gameMap6.1: .asciiz "         -------|  "
95:         gameMap6.2: .asciiz " points   |--------------|  "
96:         gameMap6.3: .asciiz " points   |--------------|  "
97:         gameMap6.4: .asciiz " points   |--------\n"
98:         gameMap6.5: .asciiz "         -------|  "
99:         gameMap6.6: .asciiz " points   |--------------|  "
100:         gameMap6.7: .asciiz " points   |--------------|  "
101:         gameMap6.8: .asciiz " points   |--------\n"
102:         gameMap6.9: .asciiz "          ------------------------------------|  Questio
n 7  |------------------------------ “Gold Medal”\n"
103:         gameMap6.10:.asciiz "                                                  ---------
-----               \n"
104:
105:         # gameMap7(10 points, 20 points, 30 points, 40 points, 50 points, 60 points,
100 points)
106:         gameMap7.1: .asciiz "         -------|  "
107:         gameMap7.2: .asciiz " points   |--------------|  "
108:         gameMap7.3: .asciiz " points   |--------------|  "
109:         gameMap7.4: .asciiz " points   |--------\n"
110:         gameMap7.5: .asciiz "         -------|  "
111:         gameMap7.6: .asciiz " points   |--------------|  "
112:         gameMap7.7: .asciiz " points   |--------------|  "
113:         gameMap7.8: .asciiz " points   |--------\n"
114:         gameMap7.9: .asciiz "          ------------------------------------|  "
115:         gameMap7.10:.asciiz " points  |------------------------------ “Gold Medal”\
n"
116:         gameMap7.11:.asciiz "                                                  ---------
-----               \n"
117:
118:
119:         # Gold Medal Prize
120:         GoldMedal1.1: .asciiz "\n\n"
121:     GoldMedal1.2: .asciiz "     CONGRATULATIONS YOU'VE WON THE GOLD MEDAL\n"
122:     GoldMedal1.3: .asciiz "           ******     /|     ******             \n"
123:     GoldMedal1.4: .asciiz "          ***   *    / |     *   ***           \n"
124:     GoldMedal1.5: .asciiz "             ** *      |     * **              \n"
125:     GoldMedal1.6: .asciiz "               **      |     **                \n"
126:     GoldMedal1.7: .asciiz "                *    -----   *                 \n"
```

```
127:
128:    # Silver Medal Prize
129:    SilverMedal2.1: .asciiz "\n\n"
130:    SilverMedal2.2: .asciiz "     CONGRATULATIONS YOU'VE WON THE SILVER MEDAL\n"
131:    SilverMedal2.3: .asciiz "           ******    -----   ******            \n"
132:    SilverMedal2.4: .asciiz "          ***   *        |   *   ***           \n"
133:    SilverMedal2.5: .asciiz "             ** *    -----   * **              \n"
134:    SilverMedal2.6: .asciiz "               **    |       **                \n"
135:    SilverMedal2.7: .asciiz "                *    -----   *                 \n"
136:
137:    # Bronze Medal Prize
138:    BronzeMedal3.1: .asciiz "\n\n"
139:    BronzeMedal3.2: .asciiz "     CONGRATULATIONS YOU'VE WON THE BRONZE MEDAL\n"
140:    BronzeMedal3.3: .asciiz "           ******    -----   ******            \n"
141:    BronzeMedal3.4: .asciiz "          ***   *        |   *   ***           \n"
142:    BronzeMedal3.5: .asciiz "             ** *    -----   * **              \n"
143:    BronzeMedal3.6: .asciiz "               **        |   **                \n"
144:    BronzeMedal3.7: .asciiz "                *    -----   *                 \n"
145:
146:    # First part of prizes
147:    Medal.1: .asciiz "               ****************               \n"
148:    Medal.2: .asciiz "               ****************               \n"
149:    Medal.3: .asciiz "               ****************               \n"
150:    Medal.4: .asciiz "                *            *                \n"
151:
152:    # Second part of prizes
153:    Medals.1: .asciiz "                 *          *                 \n"
154:    Medals.2: .asciiz "                  *        *                  \n"
155:    Medals.3: .asciiz "                   ********                   \n"
156:    Medals.4: .asciiz "                  **        **                 \n"
157:    Medals.5: .asciiz "                ****************               \n"
158:
159:
160:        # Game Map Points
161:        # MapPoints[3][3]       # 2D array
162:        MapPoints: .word 10, 20, 30 # MapPoints[0][0] = 10, MapPoints[0][1] = 20, MapP
oints[0][2] = 30
163:                   .word 40, 50, 60    # MapPoints[1][0] = 40, MapPoints[1][1] = 50,
MapPoints[1][2] = 60
164:                   .word 100       # MapPoints[2][0] = 100
165:
166:         # Question 1
167:         # ArrayQ1[7][3]       Nx +- D = A       # 2D array
168:        ArrayQ1:    .word 4, 36, 9       # ArrayQ1[0][0], ArrayQ1[0][1], ArrayQ1[0][2]
169:                    .word 5, 10, -2      # ArrayQ1[1][0], ArrayQ1[1][1], ArrayQ1[1][2]
170:                    .word 6, 42, 7       # ArrayQ1[2][0], ArrayQ1[2][1], ArrayQ1[2][2]
171:                    .word 7, 42, -6      # ArrayQ1[3][0], ArrayQ1[3][1], ArrayQ1[3][2]
172:                    .word 8, 24, 3       # ArrayQ1[4][0], ArrayQ1[4][1], ArrayQ1[4][2]
173:                    .word 9, 72, -8      # ArrayQ1[5][0], ArrayQ1[5][1], ArrayQ1[5][2]
174:                    .word 10, 40, 4      # ArrayQ1[6][0], ArrayQ1[6][1], ArrayQ1[6][2]
175:
176:
177:    # Question 2
178:    # ArrayQ2[6][3]     x^2 - D = A, A      # 2D array
179:        ArrayQ2:    .word 25, 5, -5         # ArrayQ2[0][0], ArrayQ2[0][1], ArrayQ2[0]
```

```
[2]
180:                    .word 36, 6, -6         # ArrayQ2[1][0], ArrayQ2[1][1], ArrayQ2[1]
[2]
181:                    .word 49, 7, -7         # ArrayQ2[2][0], ArrayQ2[2][1], ArrayQ2[2]
[2]
182:                    .word 64, 8, -8         # ArrayQ2[3][0], ArrayQ2[3][1], ArrayQ2[3]
[2]
183:                    .word 81, 9, -9         # ArrayQ2[4][0], ArrayQ2[4][1], ArrayQ2[4]
[2]
184:                    .word 100, 10, -10      # ArrayQ2[5][0], ArrayQ2[5][1], ArrayQ2[4]
[2]
185:
186:
187:    # Question 3
188:    # ArrayQ3[5][3]     x^2 -+ D = A, A     # 2D array
189:        ArrayQ3:    .word 10, 10, 0     # ArrayQ3[0][0], ArrayQ3[0][1], ArrayQ3[0][2]
190:                    .word 13, -13, 0    # ArrayQ3[1][0], ArrayQ3[1][1], ArrayQ3[1][2]
191:                    .word 16, 16, 0     # ArrayQ3[2][0], ArrayQ3[2][1], ArrayQ3[2][2]
192:                    .word 19, -19, 0    # ArrayQ3[3][0], ArrayQ3[3][1], ArrayQ3[3][2]
193:                    .word 20, 20, 0     # ArrayQ3[4][0], ArrayQ3[4][1], ArrayQ3[4][2]
194:
195:
196:         # Question 4
197:    # ArrayQ4[4][3]     x^2-x- D = A, A         # 2D array
198:        ArrayQ4:        .word 42, -6, 7         # ArrayQ4[0][0], ArrayQ4[0][1], ArrayQ
4[0][2]
199:                        .word 56, -7, 8         # ArrayQ4[1][0], ArrayQ4[1][1], ArrayQ
4[1][2]
200:                        .word 72, -8, 9         # ArrayQ4[2][0], ArrayQ4[2][1], ArrayQ
4[2][2]
201:                        .word 90, -9, 10        # ArrayQ4[3][0], ArrayQ4[3][1], ArrayQ
4[3][2]
202:
203:
204:         # Question 5
205:         # ArrayQ5[3][3]          2x^2+-4x-D = A, A     # 2D array
206:        ArrayQ5:        .word 48, 4, -6     # ArrayQ5[0][0], ArrayQ5[0][1], ArrayQ5[0]
[2]
207:                        .word 96, -6, 8     # ArrayQ5[1][0], ArrayQ5[1][1], ArrayQ5[1]
[2]
208:                        .word 160, 8, -10   # ArrayQ5[2][0], ArrayQ5[2][1], ArrayQ5[2]
[2]
209:
210:
211:         # Question 6
212:         # ArrayQ6Int[]      x^3 - Dx^2 + Dx - D    # 1D array
213:    ArrayQ6Int:     .space 12           # ArrayQ6Int[0], ArrayQ6Int[1], ArrayQ6Int[2]
214:    # ArrayQ6Ans[2][3]    A, A, A           # 2D array
215:        ArrayQ6Ans:     .word 1, 2, 3           # ArrayQ6Ans[0][0], ArrayQ6Ans[0][1],
ArrayQ6Ans[0][2]
216:                        .word -1, -2, -3        # ArrayQ6Ans[1][0], ArrayQ6Ans[1][1],
ArrayQ6Ans[1][2]
217:
218:
219:         # Question 7   Dx^4 - Dx^3 + Dx^2 - Dx + D
```

```
220:        ArrayQ7Int: .space 20          # 1D array (Integers) ArrayQ7Int[0], ArrayQ7In
t[1], ArrayQ7Int[2], ArrayQ7Int[3], ArrayQ7Int[4]
221:        ArrayQ7Ans1: .space 16         # 1D array (Answers)  ArrayQ7Ans1[0], ArrayQ7A
ns1[1], ArrayQ7Ans1[2], ArrayQ7Ans1[3]
222:
223:
224:         # Value of x for Question 1 (Answer)
225:    valueX_Q1:  .word 0
226:
227:    # Print Question 1
228:    Question1: .asciiz "\nQuestion 1: \n"
229:
230:    # Asking the user to enter either 0 or 1
231:    enterZeroOne: .asciiz "Enter 0 if you want to have a visual look at the map of the
 game\n"
232:    enterOne: .asciiz "Else, if You're Ready, Enter 1 To Start The Game\n"
233:    notZeroOne: .asciiz "Number Entered was Neither 1 or 0. Please Try Again\n"
234:    enter_One: .asciiz "Enter 1 To Start The Game\n"
235:    notOne: .asciiz "Number Entered was NOT 1. Please Try Again\n"
236:
237:    # Version 1 Display
238:    firstVer1:  .asciiz "x - "
239:    firstVer2:  .asciiz " = 0\n"
240:
241:    # Version 2 Display
242:    secondVer1: .asciiz "x + "
243:
244:    # Ask for roots
245:    roots: .asciiz "Enter the Roots for x: "
246:
247:    # Answer correct/incorrect
248:    correct: .asciiz "\nCORRECT!!\n"
249:    incorrect: .asciiz "\nWRONG, Please Try Again : "
250:
251:    # Lost the game
252:    lost: .asciiz "\nWRONG, YOU LOST! :(\n"
253:
254:    # Question 1 correct
255:    goodJob: .asciiz "\nGood Job! :o\n"
256:
257:    # Map or Continue or Exit
258:    zeroMap:     .asciiz "Enter 0 to see the map\n"
259:    oneContinue: .asciiz "Enter 1 to continue\n"
260:    numExit: .asciiz "Enter a number to Exit\n"
261:
262:    # Value of x1 and x2 for question 2 (Answers)
263:    valueX1_Q2:     .word 0
264:    valueX2_Q2:     .word 0
265:
266:    # Print Question 2
267:    Question2: .asciiz "\nQuestion 2: \n"
268:
269:    # Question 2 Display
270:    firstVer2.1:  .asciiz "x^2 - "
271:    firstVer2.2:  .asciiz " = 0\n"
```

```
272:
273:    # Value of x1 and x2 for question 3 (Answers)
274:    valueX1_Q3:     .word 0
275:    valueX2_Q3:     .word 0
276:
277:    # Print Question 3
278:    Question3: .asciiz "\nQuestion 3: \n"
279:
280:    # Version 1 Display
281:    firstVer3.1:  .asciiz "(x^2 - "
282:    firstVer3.2:  .asciiz "x) = 0\n"
283:
284:    # Version 2 Display
285:    secondVer3.1: .asciiz "(x^2 + "
286:
287:    # Correct Answer
288:    awesomeJob: .asciiz "\nAwesome Job! :p\n"
289:
290:    # Value of x1 and x2 for question 4 (Answers)
291:    valueX1_Q4:     .word 0
292:    valueX2_Q4:     .word 0
293:
294:    # Print Question 4
295:    Question4: .asciiz "\nQuestion 4: \n"
296:
297:    # Question 4 Display
298:    firstVer4.1:  .asciiz "x^2 - x - "
299:    firstVer4.2:  .asciiz " = 0\n"
300:
301:    # Correct Answer
302:    prizeOne.1: .asciiz "\nCongratulations! You've unlocked the Bronze Medal! :)\n\n"
303:    prizeOne.2: .asciiz "Would you like to keep playing or opt out with the Bronze Med
al?\n"
304:    prizeOne.3: .asciiz "Enter 0 to opt out\n"
305:    prizeOne.4: .asciiz "Enter 1 to continue\n"
306:
307:    prizeOneIfZero: .asciiz "\nGoodbye. . . ;)\n\n"
308:    prizeOneIfOne: .asciiz "\nQuestions Are A Little Bit Harder In This Level ;)\n"
309:
310:    # Value of x1 and x2 for question 5 (Answers)
311:    valueX1_Q5:     .word 0
312:    valueX2_Q5:     .word 0
313:
314:    # Print Question 5
315:    Question5: .asciiz "\nQuestion 5: \n"
316:
317:    # Version 1 Display
318:    firstVer5.1:  .asciiz "2x^2 + 2x - "
319:    firstVer5.2:  .asciiz " = 0\n"
320:
321:    # Version 2 Display
322:    secondVer5.1: .asciiz "2x^2 - 2x - "
323:
324:    # Correct Answer
325:    GoodJob: .asciiz "\nGood Job! :)\n"
```

```
326:
327:    # Value of x1, x2 and x3 for question 6 (Answers)
328:    valueX1_Q6:     .word 0
329:    valueX2_Q6:     .word 0
330:    valueX3_Q6:     .word 0
331:    valueX1_Q6_:    .word 0
332:    valueX2_Q6_:    .word 0
333:    valueX3_Q6_:    .word 0
334:
335:    # Print Question 6
336:    Question6: .asciiz "\nQuestion 6: \n"
337:
338:    # Version 1 Display
339:    firstVer6.1:  .asciiz "x^3 - "
340:    firstVer6.2:  .asciiz "x^2 + "
341:    firstVer6.3:  .asciiz "x - "
342:    firstVer6.4:  .asciiz " = 0\n"
343:
344:    # Version 2 Display
345:    secondVer6.1:  .asciiz "x^3 + "
346:    secondVer6.2:  .asciiz "x^2 + "
347:    secondVer6.3:  .asciiz "x + "
348:
349:    # Correct Answer
350:    IncredibleJob: .asciiz "\nIncredible Job! :O\n"
351:
352:    # Value of x1, x2, x3, x4 for question 7 (Answers)
353:    valueX1_Q7:     .word 0
354:    valueX2_Q7:     .word 0
355:    valueX3_Q7:     .word 0
356:    valueX4_Q7:     .word 0
357:
358:    # Print Question 7
359:    Question7: .asciiz "\nQuestion 7: \n"
360:
361:    # Version 1 Display
362:    firstVer7.1:  .asciiz "x^4 - "
363:    firstVer7.2:  .asciiz "x^3 + "
364:    firstVer7.3:  .asciiz "x^2 - "
365:    firstVer7.4:  .asciiz "x + "
366:    firstVer7.5:  .asciiz " = 0\n"
367:
368:    # Correct Answer
369:    prizeTwo.1: .asciiz "\nOutStanding! Congratulations! You've unlocked the Silver Me
dal! :)\n\n"
370:    prizeTwo.2: .asciiz "Would you like to keep playing or opt out with the Silver Med
al?\n"
371:    prizeTwo.3: .asciiz "Enter 0 to opt out\n"
372:    prizeTwo.4: .asciiz "Enter 1 to continue\n"
373:
374:    prizeThree: .asciiz "\nCONGRATULATIONS! You Have Solved All Of The Questions and W
ON THE GOLD MEDAL\n"
375:
376:    prizeTwoIfZero: .asciiz "\nGoodbye. . . ;)\n\n"
377:    prizeTwoIfOne:  .asciiz "\nLooks Like Your Goal is To Win This Game ;)\nHate To Te
```

```
ll Ya, But This Last Question is the HARDEST\n\n"
378:
379:
380:    # Generating Sounds
381:    sound5: .byte 95      # Sound number for 95
382:    duration5: .byte 200  # Duration of 200 milliseconds
383:    instrument5: .byte 7  # Sound instrument number 7
384:    volume5: .byte 127    # Maximum volume 127
385:
386:    beep6: .byte 45       # Sound number for 45
387:    duration6: .byte 500  # Duration of 500 milliseconds
388:    instrument6: .byte 30  # Sound instrument number 30
389:    volume6: .byte 100    # Maximum volume 100
390:
391:    beep: .byte 60        # Sound number for 60
392:    duration: .byte 500   # Duration of 500 milliseconds
393:    instrument: .byte 55  # Sound instrument number 55
394:    volume: .byte 100    # Maximum volume 100
395:
396:    beep1: .byte 70        # Sound number for 7-
397:    duration1: .byte 500   # Duration of 500 milliseconds
398:    instrument1: .byte 55  # Sound instrument number 55
399:    volume1: .byte 100    # Maximum volume 100
400:
401:    beep2: .byte 55        # Sound number for 55
402:    duration2: .byte 500   # Duration of 500 milliseconds
403:    instrument2: .byte 55  # Sound instrument number 55
404:    volume2: .byte 100    # Maximum volume 100
405:
406:    beep9: .byte 60       # Sound number for 60
407:    duration9: .byte 500  # Duration of 500 milliseconds
408:    instrument9: .byte 55  # Sound instrument number 55
409:    volume9: .byte 100    # Maximum volume 100
410:
411:    beep7: .byte 70       # Sound number for 70
412:    duration7: .byte 500  # Duration of 500 milliseconds
413:    instrument7: .byte 55  # Sound instrument number 55
414:    volume7: .byte 100    # Maximum volume 100
415:
416:    beep8: .byte 55       # Sound number for 55
417:    duration8: .byte 500  # Duration of 500 milliseconds
418:    instrument8: .byte 55  # Sound instrument number 55
419:    volume8: .byte 100    # Maximum volume 100
420:
421:    beep3: .byte 95       # Sound number for 95
422:    duration3: .byte 200  # Duration of 200 milliseconds
423:    instrument3: .byte 105  # Sound instrument number 105
424:    volume3: .byte 100    # Maximum volume 100
425:
426:
427: .text
428:    # Pseudocode:
429:    # int main() {
430:    #   while(enter == 0){
431:    #   GameMap();
```

```
    # }
    #   while(enter == 1){
    #   PolynomialQuestionOne(ArrayQ1);
    #   continue;
    # }
    #   while(enter == 0){
    #   gameMap1(MapPoints[0][0]);
    # }
    #   while(enter == 1){
    #   PolynomialQuestionTwo(ArrayQ2);
    #   continue;
    # }
    #   while(enter == 0){
    #   gameMap2(MapPoints[0][0], MapPoints[0][1]);
    # }
    #   while(enter == 1){
    #   PolynomialQuestionThree(ArrayQ3);
    #   continue;
    # }
    #   while(enter == 0){
    #   BronzeMedal();
    #   exit(0);
    # }
    #   while(enter == 1){
    #   continue;
    # }
    #   while(enter == 0){
    #   gameMap3(MapPoints[0][0], MapPoints[0][1], MapPoints[0][2]);
    # }
    #   while(enter == 1){
    #   PolynomialQuestionFour(ArrayQ4);
    #   continue;
    # }
    #   while(enter == 0){
    #   gameMap4(MapPoints[0][0], MapPoints[0][1], MapPoints[0][2], MapPoints[1][0]);
    # }
    #   while(enter == 1){
    #   PolynomialQuestionFive(ArrayQ5);
    #   continue;
    # }
    #   while(enter == 0){
    #   gameMap5(MapPoints[0][0], MapPoints[0][1], MapPoints[0][2], MapPoints[1][0], M
apPoints[1][1]);
    # }
    #   while(enter == 1){
    #   PolynomialQuestionSix(ArrayQ6Int, ArrayQ6Ans);
    #   continue;
    # }
    #   while(enter == 0){
    #   SilverMedal();
    #   exit(0);
    # }
    #   while(enter == 1){
    #   continue;
    # }
```

```
486:    #   while(enter == 0){
487:    #   gameMap6(MapPoints[0][0], MapPoints[0][1], MapPoints[0][2], MapPoints[1][0], M
apPoints[1][1], MapPoints[1][2]);
488:    # }
489:    #   while(enter == 1){
490:    #   PolynomialQuestionSeven(ArrayQ7Int, ArrayQ7Ans1);
491:    #   continue;
492:    # }
493:    #   GoldMedal();
494:    #   while(enter == 0){
495:    #   gameMap7(MapPoints[0][0], MapPoints[0][1], MapPoints[0][2], MapPoints[1][0], M
apPoints[1][1], MapPoints[1][2], MapPoints[2][0]);
496:    # }
497:    #   return 0;
498:    .globl main
499:    main:
500:
501:        jal printGameIntro
502:
503:
504:        # Question 1 random number between 0-6
505:        addi $v0, $zero, 30     # Syscall 30: System Time syscall
506:        syscall                    # $a0 will contain the 32 LS bits of the system tim
e
507:        add $t0, $zero, $a0     # add/move value held in register $a0 to $t0 using add

508:
509:        addi $v0, $zero, 40     # syscall code for seeding random number generator
510:        add $a0, $zero, $zero      # Initialize/Set RNG ID to 0
511:        add $a1, $zero, $t0        # Set Random seed to value held in register $t0
512:        syscall             # Perform
513:
514:        addi $v0, $zero, 42        # Syscall 42: generate a Random int range
515:        add $a0, $zero, $zero      # Set RNG ID to 0
516:        addi $a1, $zero, 7         # Set upper bound to 7 (exclusive) (7 different que
stions for question 1)
517:        syscall                    # Generate a random number and put it in $a0
518:        add $s3, $zero, $a0        # Move/add/Copy the random number to register $s3
519:
520:
521:        # Question 2 random number between 0-5
522:        addi $v0, $zero, 30     # Syscall 30: System Time syscall
523:        syscall                    # $a0 will contain the 32 LS bits of the system tim
e
524:        add $t0, $zero, $a0     # add/move value held in register $a0 to $t0 using add

525:
526:        addi $v0, $zero, 40     # syscall code for seeding random number generator
527:        add $a0, $zero, $zero      # Initialize/Set RNG ID to 0
528:        add $a1, $zero, $t0        # Set Random seed to value held in register $t0
529:        syscall             # Perform
530:
531:        addi $v0, $zero, 42        # Syscall 42: generate a Random int range
532:        add $a0, $zero, $zero      # Set RNG ID to 0
533:        addi $a1, $zero, 6         # Set upper bound to 6 (exclusive) (6 different que
```

```
stions for question 2)
534:        syscall                    # Generate a random number and put it in $a0
535:        add $s6, $zero, $a0        # Move/add/Copy the random number to register $s3
536:
537:
538:        # Question 3 random number between 0-4
539:        addi $v0, $zero, 30     # Syscall 30: System Time syscall
540:        syscall                    # $a0 will contain the 32 LS bits of the system tim
e
541:        add $t0, $zero, $a0     # add/move value held in register $a0 to $t0 using add

542:
543:        addi $v0, $zero, 40     # syscall code for seeding random number generator
544:        add $a0, $zero, $zero      # Initialize/Set RNG ID to 0
545:        add $a1, $zero, $t0        # Set Random seed to value held in register $t0
546:        syscall             # Perform
547:
548:        addi $v0, $zero, 42        # Syscall 42: generate a Random int range
549:        add $a0, $zero, $zero      # Set RNG ID to 0
550:        addi $a1, $zero, 5         # Set upper bound to 5 (exclusive) (5 different que
stions for question 3)
551:        syscall                    # Generate a random number and put it in $a0
552:        add $s7, $zero, $a0        # Move/add/Copy the random number to register $s7
553:
554:
555:        # Question 4 random number between 0-3
556:        addi $v0, $zero, 30     # Syscall 30: System Time syscall
557:        syscall                    # $a0 will contain the 32 LS bits of the system tim
e
558:        add $t0, $zero, $a0     # add/move value held in register $a0 to $t0 using add

559:
560:        addi $v0, $zero, 40     # syscall code for seeding random number generator
561:        add $a0, $zero, $zero      # Initialize/Set RNG ID to 0
562:        add $a1, $zero, $t0        # Set Random seed to value held in register $t0
563:        syscall             # Perform
564:
565:        addi $v0, $zero, 42        # Syscall 42: generate a Random int range
566:        add $a0, $zero, $zero      # Set RNG ID to 0
567:        addi $a1, $zero, 4         # Set upper bound to 4 (exclusive) (4 different que
stions for question 4)
568:        syscall                    # Generate a random number and put it in $a0
569:        add $t7, $zero, $a0        # Move/add/Copy the random number to register $t7
570:
571:
572:        # Question 5 random number between 0-2
573:        addi $v0, $zero, 30     # Syscall 30: System Time syscall
574:        syscall                    # $a0 will contain the 32 LS bits of the system tim
e
575:        add $t0, $zero, $a0     # add/move value held in register $a0 to $t0 using add

576:
577:        addi $v0, $zero, 40     # syscall code for seeding random number generator
578:        add $a0, $zero, $zero      # Initialize/Set RNG ID to 0
579:        add $a1, $zero, $t0        # Set Random seed to value held in register $t0
```

```
580:        syscall             # Perform
581:
582:        addi $v0, $zero, 42        # Syscall 42: generate a Random int range
583:        add $a0, $zero, $zero      # Set RNG ID to 0
584:        addi $a1, $zero, 3         # Set upper bound to 3 (exclusive) (3 different que
stions for question 5)
585:        syscall                    # Generate a random number and put it in $a0
586:        add $t8, $zero, $a0        # Move/add/Copy the random number to register $t8
587:
588:
589:        # Question 6 random number between 0-1
590:        addi $v0, $zero, 30     # Syscall 30: System Time syscall
591:        syscall                    # $a0 will contain the 32 LS bits of the system tim
e
592:        add $t0, $zero, $a0     # add/move value held in register $a0 to $t0 using add

593:
594:        addi $v0, $zero, 40     # syscall code for seeding random number generator
595:        add $a0, $zero, $zero      # Initialize/Set RNG ID to 0
596:        add $a1, $zero, $t0        # Set Random seed to value held in register $t0
597:        syscall             # Perform
598:
599:        addi $v0, $zero, 42        # Syscall 42: generate a Random int range
600:        add $a0, $zero, $zero      # Set RNG ID to 0
601:        addi $a1, $zero, 2         # Set upper bound to 2 (exclusive) (2 different que
stions for question 6)
602:        syscall                    # Generate a random number and put it in $a0
603:        add $a3, $zero, $a0        # Move/add/Copy the random number to register $a3
604:
605:
606:        # ADDING VALUES TO ArrayQ6Int QUESTION 6: Creating the values to add to 1D Arr
ayQ6Int Question 6
607:        addi $s0, $zero, 6  # initialize/add 6 to register $s0
608:        addi $s1, $zero, 11 # initialize/add 11 to register $s1
609:        addi $s2, $zero, 6  # initialize/add 6 to register $s2
610:
611:        # Creating a 0 index by initializing index 0 = $t0
612:        addi $t0, $zero, 0  # initialize index 0 by adding 0 to register $t0
613:
614:        # Storing values into 1D ArrayQ6Int Question 6 using sw (store word)
615:        sw $s0, ArrayQ6Int($t0)    # Storing value held in register $s0, at index 0 in
 ArrayQ6Int Question 6
616:        addi $t0, $t0, 4    # Updating register $t0 to index 4 by adding 4 to it
617:        sw $s1, ArrayQ6Int($t0)    # Storing value held in register $s1, at index 4 in
 ArrayQ6Int Question 6
618:        addi $t0, $t0, 4    # Updating register $t0 to index 8 by adding 4 to it
619:        sw $s2, ArrayQ6Int($t0)    # Storing value held in register $s2, at index 8 in
 ArrayQ6Int Question 6
620:
621:
622:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
623:        la $a0, enterZeroOne    # Load enterZeroOne message into $a0 using la (load Ad
dress)
624:        syscall     # Print out the output on screen
```

```
625:
626:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
627:        la $a0, enterOne    # Load enterOne message into $a0 using la (load Address)
628:        syscall     # Print out the output on screen
629:
630:
631:        # Game Map/Displaying Question 1
632:    # While loop for Question 1
633:    While_Q1:
634:        li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
635:        syscall     # Print out the output on screen
636:
637:        move $s0, $v0   # move value held in $v0 to register $s0
638:
639:        addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
640:        beq $s0, $t0, if_Zero_Q1    # If value held in register $s0 is equal to value
held in $t0 (0) go to if_Zero_Q1, else keep going
641:
642:        addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
643:        beq $s0, $t1, if_One_Q1     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q1, else keep going
644:
645:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
646:        la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
647:        syscall     # Print out the output on screen
648:
649:        j While_Q1  # jump back to While_Q1 until value is equal to 0 or 1
650:
651:    if_Zero_Q1:
652:        jal gameMap # Printing Game Map by calling child function gameMap
653:
654:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
655:        la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
656:        syscall     # Print out the output on screen
657:    inner_While_Q1:
658:        li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
659:        syscall     # Print out the output on screen
660:
661:        move $s1, $v0   # move value held in $v0 to register $s1
662:
663:        addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
664:        beq $s1, $t2, if_One_Q1 # If value held in register $s1 is equal to value held
 in $t2 ( 1 ) go to if_One_Q1, else keep going
665:
666:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
667:        la $a0, notOne  # Load notOne message into $a0 using la (load Address)
668:        syscall     # Print out the output on screen
669:
670:        j inner_While_Q1    # jump back to inner_While_Q1 until value is equal to 1
```

```
671:
672:    if_One_Q1:
673:        # Question 1
674:        li $a0, 20  # left x-coordinate is 20
675:        li $a1, 70  # width is 70
676:        li $a2, 20  # top y-coordinate is 20
677:        li $a3, 50  # height is 50
678:        jal rectangleRed # Jump to rectangleRed
679:
680:        # Horizontal Line 1
681:        li $a0, 90  # left x-coordinate is 90
682:        li $a1, 130 # width is 130
683:        li $a2, 45  # top y-coordinate is 45
684:        li $a3, 50  # height is 50
685:        jal Hline   # Jump to Hline
686:
687:        # Number 1
688:        li $a0, 67  # left x-coordinate is 67
689:        li $a1, 50  # width is 50
690:        li $a2, 35  # top y-coordinate is 35
691:        li $a3, 20  # height is 20
692:        jal Vline    # Jump to Vline
693:
694:        # Q for Question
695:        li $a0, 35  # left x-coordinate is 35
696:        li $a1, 20  # width is 20
697:        li $a2, 35  # top y-coordinate is 35
698:        li $a3, 50  # height is 50
699:        jal Hline    # Jump to Hline
700:
701:        # Q for Question
702:        li $a0, 55  # left x-coordinate is 55
703:        li $a1, 40  # width is 40
704:        li $a2, 35  # top y-coordinate is 35
705:        li $a3, 20  # height is 20
706:        jal Vline    # Jump to Vline
707:
708:        # Q for Question
709:        li $a0, 35  # left x-coordinate is 35
710:        li $a1, 40  # width is 40
711:        li $a2, 35  # top y-coordinate is 35
712:        li $a3, 20  # height is 20
713:        jal Vline    # Jump to Vline
714:
715:        # Q for Question
716:        li $a0, 47  # left x-coordinate is 47
717:        li $a1, 15  # width is 15
718:        li $a2, 47  # top y-coordinate is 47
719:        li $a3, 50  # height is 50
720:        jal Hline    # Jump to Hline
721:
722:        # Q for Question
723:        li $a0, 35  # left x-coordinate is 35
724:        li $a1, 20  # width is 20
725:        li $a2, 54  # top y-coordinate is 54
```

```
726:        li $a3, 50  # height is 50
727:        jal Hline    # Jump to Hline
728:
729:        jal PolynomialQuestionOne   # Jump to PolynomialQuestionOne and perform questi
on 1
730:
731:        j exit_While_Q1         # jump exit_While_Q1
732:
733:    exit_While_Q1:
734:
735:
736:        # MapPoints[0][0] -> Prints 10
737:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
738:        la $a0, goodJob # Load zeroMap message into $a0 using la (load Address)
739:        syscall     # Print out the output on screen
740:
741:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
742:        la $a0, zeroMap # Load zeroMap message into $a0 using la (load Address)
743:        syscall     # Print out the output on screen
744:
745:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
746:        la $a0, oneContinue # Load zeroMap message into $a0 using la (load Address)
747:        syscall     # Print out the output on screen
748:
749:
750:    # While loop for Question 2
751:    While_Q2:
752:        li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
753:        syscall     # Print out the output on screen
754:
755:        move $s0, $v0   # move value held in $v0 to register $s0
756:
757:        addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
758:        beq $s0, $t0, if_Zero_Q2    # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q2, else keep going
759:
760:        addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
761:        beq $s0, $t1, if_One_Q2     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q2, else keep going
762:
763:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
764:        la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
765:        syscall     # Print out the output on screen
766:
767:        j While_Q2  # jump back to While_Q2 until value is equal to 0 or 1
768:
769:    if_Zero_Q2:
770:        jal gameMap1    # Printing Game Map1 by calling child function gameMap1
771:
772:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
```

```
ter $v0
773:        la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
774:        syscall     # Print out the output on screen
775:
776:    inner_While_Q2:
777:        li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
778:        syscall     # Print out the output on screen
779:
780:        move $s1, $v0   # move value held in $v0 to register $s1
781:
782:        addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
783:        beq $s1, $t2, if_One_Q2 # If value held in register $s1 is equal to value held
 in $t2 ( 1 ) go to if_One_Q1, else keep going
784:
785:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
786:        la $a0, notOne  # Load notOne message into $a0 using la (load Address)
787:        syscall     # Print out the output on screen
788:
789:        j inner_While_Q2    # jump back to inner_While_Q2 until value is equal to 1
790:
791:    if_One_Q2:
792:        # Question 2
793:        li $a0, 220 # left x-coordinate is 220
794:        li $a1, 70  # width is 70
795:        li $a2, 20  # top y-coordinate is 20
796:        li $a3, 50  # height is 50
797:        jal rectangleGreen # Jump to rectangleGreen
798:
799:        # Horizontal Line 2
800:        li $a0, 290 # left x-coordinate is 290
801:        li $a1, 132 # width is 132
802:        li $a2, 45  # top y-coordinate is 45
803:        li $a3, 50  # height is 50
804:        jal Hline   # Jump to Hline
805:
806:        # Number 2
807:        li $a0, 260 # left x-coordinate is 260
808:        li $a1, 20  # width is 20
809:        li $a2, 33  # top y-coordinate is 33
810:        li $a3, 50  # height is 50
811:        jal Hline    # Jump to Hline
812:
813:        # Number 2
814:        li $a0, 280 # left x-coordinate is 280
815:        li $a1, 40  # width is 40
816:        li $a2, 33  # top y-coordinate is 33
817:        li $a3, 13  # height is 13
818:        jal Vline    # Jump to Vline
819:
820:        # Number 2
821:        li $a0, 260 # left x-coordinate is 260
822:        li $a1, 20  # width is 20
823:        li $a2, 45  # top y-coordinate is 45
```

```
824:        li $a3, 50  # height is 50
825:        jal Hline    # Jump to Hline
826:
827:        # Number 2
828:        li $a0, 260 # left x-coordinate is 260
829:        li $a1, 40  # width is 40
830:        li $a2, 45  # top y-coordinate is 45
831:        li $a3, 13  # height is 13
832:        jal Vline    # Jump to Vline
833:
834:        # Number 2
835:        li $a0, 260 # left x-coordinate is 260
836:        li $a1, 20  # width is 20
837:        li $a2, 57  # top y-coordinate is 57
838:        li $a3, 50  # height is 50
839:        jal Hline   # Jump to Hline
840:
841:        # Q for Question
842:        li $a0, 229 # left x-coordinate is 229
843:        li $a1, 20  # width is 20
844:        li $a2, 35  # top y-coordinate is 35
845:        li $a3, 50  # height is 50
846:        jal Hline    # Jump to Hline
847:
848:        # Q for Question
849:        li $a0, 249 # left x-coordinate is 249
850:        li $a1, 40  # width is 40
851:        li $a2, 35  # top y-coordinate is 35
852:        li $a3, 20  # height is 20
853:        jal Vline    # Jump to Vline
854:
855:        # Q for Question
856:        li $a0, 229 # left x-coordinate is 229
857:        li $a1, 40  # width is 40
858:        li $a2, 35  # top y-coordinate is 35
859:        li $a3, 20  # height is 20
860:        jal Vline    # Jump to Vline
861:
862:        # Q for Question
863:        li $a0, 241 # left x-coordinate is 241
864:        li $a1, 15  # width is 15
865:        li $a2, 47  # top y-coordinate is 47
866:        li $a3, 50  # height is 50
867:        jal Hline   # Jump to Hline
868:
869:        # Q for Question
870:        li $a0, 229 # left x-coordinate is 229
871:        li $a1, 20  # width is 20
872:        li $a2, 54  # top y-coordinate is 54
873:        li $a3, 50  # height is 50
874:        jal Hline   # Jump to Hline
875:
876:        jal PolynomialQuestionTwo   # Jump to PolynomialQuestionTwo and perform questi
on 2
877:
```

```
878:        j exit_While_Q2         # jump exit_While_Q2
879:
880:    exit_While_Q2:
881:
882:
883:        # MapPoints[0][1] -> 20 Points
884:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
885:        la $a0, awesomeJob  # Load awesomeJob message into $a0 using la (load Address)
886:        syscall     # Print out the output on screen
887:
888:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
889:        la $a0, zeroMap     # Load zeroMap message into $a0 using la (load Address)
890:        syscall     # Print out the output on screen
891:
892:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
893:        la $a0, oneContinue # Load zeroMap message into $a0 using la (load Address)
894:        syscall     # Print out the output on screen
895:
896:
897:    # While loop for Question 3
898:    While_Q3:
899:        li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
900:        syscall     # Print out the output on screen
901:
902:        move $s0, $v0   # move value held in $v0 to register $s0
903:
904:        addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
905:        beq $s0, $t0, if_Zero_Q3    # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q3, else keep going
906:
907:        addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
908:        beq $s0, $t1, if_One_Q3     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q3, else keep going
909:
910:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
911:        la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
912:        syscall     # Print out the output on screen
913:
914:        j While_Q3  # jump back to While_Q3 until value is equal to 0 or 1
915:
916:    if_Zero_Q3:
917:        jal gameMap2    # Printing Game Map2 by calling child function gameMap2
918:
919:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
920:        la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
921:        syscall     # Print out the output on screen
922:
923:    inner_While_Q3:
924:        li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
```

```
lue 5 into register $v0 (scanf)
925:        syscall     # Print out the output on screen
926:
927:        move $s1, $v0   # move value held in $v0 to register $s1
928:
929:        addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
930:        beq $s1, $t2, if_One_Q3 # If value held in register $s1 is equal to value held
 in $t2 ( 1 ) go to if_One_Q1, else keep going
931:
932:        li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
933:        la $a0, notOne  # Load notOne message into $a0 using la (load Address)
934:        syscall     # Print out the output on screen
935:
936:        j inner_While_Q3    # jump back to inner_While_Q3 until value is equal to 1
937:
938:    if_One_Q3:
939:        # Question 3
940:        li $a0, 422 # left x-coordinate is 422
941:        li $a1, 70  # width is 70
942:        li $a2, 20  # top y-coordinate is 20
943:        li $a3, 50  # height is 50
944:        jal rectangleYellow # Jump to rectangleYellow
945:
946:        # Vertical line 1
947:        li $a0, 457 # left x-coordinate is 457
948:        li $a1, 30  # width is 30
949:        li $a2, 70  # top y-coordinate is 70
950:        li $a3, 35  # height is 35
951:        jal Vline    # Jump to Vline
952:
953:        # Number 3
954:        li $a0, 463 # left x-coordinate is 463
955:        li $a1, 20  # width is 20
956:        li $a2, 33  # top y-coordinate is 33
957:        li $a3, 50  # height is 50
958:        jal Hline    # Jump to Hline
959:
960:        # Number 3
961:        li $a0, 483 # left x-coordinate is 483
962:        li $a1, 40  # width is 40
963:        li $a2, 33  # top y-coordinate is 33
964:        li $a3, 13  # height is 13
965:        jal Vline   # Jump to Vline
966:
967:        # Number 3
968:        li $a0, 463 # left x-coordinate is 463
969:        li $a1, 20  # width is 20
970:        li $a2, 45  # top y-coordinate is 45
971:        li $a3, 50  # height is 50
972:        jal Hline   # Jump to Hline
973:
974:        # Number 3
975:        li $a0, 483 # left x-coordinate is 483
976:        li $a1, 40  # width is 40
```

```
977:        li $a2, 45  # top y-coordinate is 45
978:        li $a3, 13  # height is 13
979:        jal Vline   # Jump to Vline
980:
981:        # Number 3
982:        li $a0, 463 # left x-coordinate is 463
983:        li $a1, 20  # width is 20
984:        li $a2, 57  # top y-coordinate is 57
985:        li $a3, 50  # height is 50
986:        jal Hline   # Jump to Hline
987:
988:        # Q for Question
989:        li $a0, 431 # left x-coordinate is 431
990:        li $a1, 20  # width is 20
991:        li $a2, 35  # top y-coordinate is 35
992:        li $a3, 50  # height is 50
993:        jal Hline   # Jump to Hline
994:
995:        # Q for Question
996:        li $a0, 451 # left x-coordinate is 451
997:        li $a1, 40  # width is 40
998:        li $a2, 35  # top y-coordinate is 35
999:        li $a3, 20  # height is 20
1000:       jal Vline   # Jump to Vline
1001:
1002:       # Q for Question
1003:       li $a0, 431 # left x-coordinate is 431
1004:       li $a1, 40  # width is 40
1005:       li $a2, 35  # top y-coordinate is 35
1006:       li $a3, 20  # height is 17
1007:       jal Vline   # Jump to Vline
1008:
1009:       # Q for Question
1010:       li $a0, 443 # left x-coordinate is 443
1011:       li $a1, 15  # width is 15
1012:       li $a2, 47  # top y-coordinate is 47
1013:       li $a3, 50  # height is 50
1014:       jal Hline   # Jump to Hline
1015:
1016:       # Q for Question
1017:       li $a0, 431 # left x-coordinate is 431
1018:       li $a1, 20  # width is 20
1019:       li $a2, 54  # top y-coordinate is 54
1020:       li $a3, 50  # height is 50
1021:       jal Hline   # Jump to Hline
1022:
1023:       jal PolynomialQuestionThree # Jump to PolynomialQuestionThree and perform ques
tion 3
1024:
1025:       j exit_While_Q3         # jump exit_While_Q3
1026:
1027:   exit_While_Q3:
1028:
1029:
1030:       # MapPoints[0][2] -> 30 Points
```

```
1031:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1032:       la $a0, prizeOne.1  # Load prizeOne.1 message into $a0 using la (load Address)
1033:       syscall     # Print out the output on screen
1034:
1035:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1036:       la $a0, prizeOne.2  # Load prizeOne.2 message into $a0 using la (load Address)
1037:       syscall     # Print out the output on screen
1038:
1039:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1040:       la $a0, prizeOne.3  # Load prizeOne.3 message into $a0 using la (load Address)
1041:       syscall     # Print out the output on screen
1042:
1043:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1044:       la $a0, prizeOne.4  # Load prizeOne.4 message into $a0 using la (load Address)
1045:       syscall     # Print out the output on screen
1046:
1047:
1048:       # While loop for prize or Question 4
1049:       While_Q4_:
1050:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1051:       syscall     # Print out the output on screen
1052:
1053:       move $s0, $v0   # move value held in $v0 to register $s0
1054:
1055:       addi $t0, $zero, 0      # Add 0 to register $t0 in order to do comparisons
1056:       beq $s0, $t0, if_Zero_Q4_   # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q4_, else keep going
1057:
1058:       addi $t1, $zero, 1      # Add 1 to register $t1 in order to do comparisons
1059:       beq $s0, $t1, if_One_Q4_    # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q4_, else keep going
1060:
1061:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1062:       la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
1063:       syscall     # Print out the output on screen
1064:
1065:       j While_Q4_ # jump back to While_Q4_ until value is equal to 0 or 1
1066:
1067:   if_Zero_Q4_:
1068:       jal BronzeMedal     # Printing Bronze Medal Prize by calling child function Br
onzeMedal
1069:
1070:       la $a0, beep9       # Load the address of "beep9" into $a0
1071:       la $a1, duration9   # Load the address of "duration9" into $a1
1072:       la $a2, instrument9 # Load the address of "instrument9" into $a2
1073:       la $a3, volume9     # Load the address of "volume9" into $a3
1074:
1075:       lb $a0, 0($a0)      # Load the value of "beep9" into $a0
1076:       lb $a1, 0($a1)      # Load the value of "duration9" into $a1
```

```
1077:       lb $a2, 0($a2)      # Load the value of "instrument9" into $a2
1078:       lb $a3, 0($a3)      # Load the value of "volume9" into $a3
1079:
1080:           li $v0, 31  # Use the "play note" system call
1081:           syscall     # Print out the output on screen
1082:
1083:
1084:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1085:       li $a0, 800 # wait for 800 millisecond
1086:       syscall     # Print out the output on screen
1087:
1088:
1089:       la $a0, beep7       # Load the address of "beep7" into $a0
1090:       la $a1, duration7   # Load the address of "duratin7" into $a1
1091:       la $a2, instrument7 # Load the address of "instrument7" into $a2
1092:       la $a3, volume7     # Load the address of "volume7" into $a3
1093:
1094:       lb $a0, 0($a0)      # Load the value of "beep7" into $a0
1095:       lb $a1, 0($a1)      # Load the value of "duration7" into $a1
1096:       lb $a2, 0($a2)      # Load the value of "instrument7" into $a2
1097:       lb $a3, 0($a3)      # Load the value of "volume7" into $a3
1098:
1099:       li $v0, 31  # Use the "play note" system call
1100:           syscall     # Print out the output on screen
1101:
1102:
1103:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1104:       li $a0, 300 # wait for 300 millisecond
1105:       syscall     # Print out the output on screen
1106:
1107:
1108:       la $a0, beep7       # Load the address of "beep7" into $a0
1109:       la $a1, duration7   # Load the address of "duratin7" into $a1
1110:       la $a2, instrument7 # Load the address of "instrument7" into $a2
1111:       la $a3, volume7     # Load the address of "volume7" into $a3
1112:
1113:       lb $a0, 0($a0)      # Load the value of "beep7" into $a0
1114:       lb $a1, 0($a1)      # Load the value of "duration7" into $a1
1115:       lb $a2, 0($a2)      # Load the value of "instrument7" into $a2
1116:       lb $a3, 0($a3)      # Load the value of "volume7" into $a3
1117:
1118:       li $v0, 31  # Use the "play note" system call
1119:           syscall     # Print out the output on screen
1120:
1121:
1122:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1123:       li $a0, 300 # wait for 300 millisecond
1124:       syscall     # Print out the output on screen
1125:
1126:
1127:           la $a0, beep7       # Load the address of "beep7" into $a0
1128:       la $a1, duration7   # Load the address of "duratin7" into $a1
1129:       la $a2, instrument7 # Load the address of "instrument7" into $a2
1130:       la $a3, volume7     # Load the address of "volume7" into $a3
1131:
```

```
1132:       lb $a0, 0($a0)      # Load the value of "beep1" into $a0
1133:       lb $a1, 0($a1)      # Load the value of "duration1" into $a1
1134:       lb $a2, 0($a2)      # Load the value of "instrument1" into $a2
1135:       lb $a3, 0($a3)      # Load the value of "volume1" into $a3
1136:
1137:       li $v0, 31  # Use the "play note" system call
1138:           syscall     # Print out the output on screen
1139:
1140:
1141:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1142:       li $a0, 300 # wait for 300 millisecond
1143:       syscall     # Print out the output on screen
1144:
1145:
1146:           la $a0, beep9       # Load the address of "beep9" into $a0
1147:       la $a1, duration9   # Load the address of "duration9" into $a1
1148:       la $a2, instrument9 # Load the address of "instrument9" into $a2
1149:       la $a3, volume9     # Load the address of "volume9" into $a3
1150:
1151:       lb $a0, 0($a0)      # Load the value of "beep9" into $a0
1152:       lb $a1, 0($a1)      # Load the value of "duration9" into $a1
1153:       lb $a2, 0($a2)      # Load the value of "instrument9" into $a2
1154:       lb $a3, 0($a3)      # Load the value of "volume9" into $a3
1155:
1156:           li $v0, 31  # Use the "play note" system call
1157:           syscall     # Print out the output on screen
1158:
1159:
1160:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1161:       li $a0, 500 # wait for 500 millisecond
1162:       syscall     # Print out the output on screen
1163:
1164:
1165:       la $a0, beep8       # Load the address of "beep8" into $a0
1166:       la $a1, duration8   # Load the address of "duration8" into $a1
1167:       la $a2, instrument8 # Load the address of "instrument8" into $a2
1168:       la $a3, volume8     # Load the address of "volume8" into $a3
1169:
1170:       lb $a0, 0($a0)      # Load the value of "beep8" into $a0
1171:       lb $a1, 0($a1)      # Load the value of "duration8" into $a1
1172:       lb $a2, 0($a2)      # Load the value of "instrument8" into $a2
1173:       lb $a3, 0($a3)      # Load the value of "volume8" into $a3
1174:
1175:           li $v0, 31  # Use the "play note" system call
1176:           syscall     # Print out the output on screen
1177:
1178:
1179:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1180:       li $a0, 500 # wait for 500 millisecond
1181:       syscall     # Print out the output on screen
1182:
1183:
1184:       la $a0, beep3       # Load the address of "beep3" into $a0
1185:       la $a1, duration3   # Load the address of "duration3" into $a1
1186:       la $a2, instrument3 # Load the address of "instrument3" into $a2
```

```
1187:       la $a3, volume3     # Load the address of "volume3" into $a3
1188:
1189:       lb $a0, 0($a0)      # Load the value of "beep3" into $a0
1190:       lb $a1, 0($a1)      # Load the value of "duration3" into $a1
1191:       lb $a2, 0($a2)      # Load the value of "instrument3" into $a2
1192:       lb $a3, 0($a3)      # Load the value of "volume3" into $a3
1193:
1194:           li $v0, 31  # Use the "play note" system call
1195:           syscall     # Print out the output on screen
1196:
1197:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1198:       la $a0, prizeOneIfZero  # Load prizeOneIfZero message into $a0 using la (load
Address)
1199:       syscall     # Print out the output on screen
1200:
1201:       j _exit_While_Q4_           # jump _exit_While_Q4_
1202:
1203:   if_One_Q4_:
1204:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1205:       la $a0, prizeOneIfOne   # Load prizeOneIfOne message into $a0 using la (load A
ddress)
1206:       syscall     # Print out the output on screen
1207:
1208:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1209:       la $a0, zeroMap # Load zeroMap message into $a0 using la (load Address)
1210:       syscall     # Print out the output on screen
1211:
1212:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1213:       la $a0, oneContinue # Load oneContinue message into $a0 using la (load Address
)
1214:       syscall     # Print out the output on screen
1215:
1216:       j exit_While_Q4_            # jump exit_While_Q4_
1217:
1218:   _exit_While_Q4_:
1219:       # Telling the system to stop executing the program
1220:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
1221:       syscall     # Print out the output on screen
1222:
1223:
1224:   exit_While_Q4_:
1225:
1226:
1227:   # While loop for Question 4
1228:   While_Q4:
1229:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1230:       syscall     # Print out the output on screen
1231:
1232:       move $s0, $v0   # move value held in $v0 to register $s0
```

```
1233:
1234:       addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
1235:       beq $s0, $t0, if_Zero_Q4    # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q4, else keep going
1236:
1237:       addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
1238:       beq $s0, $t1, if_One_Q4     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q4, else keep going
1239:
1240:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1241:       la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
1242:       syscall     # Print out the output on screen
1243:
1244:       j While_Q4  # jump back to While_Q4 until value is equal to 0 or 1
1245:
1246:   if_Zero_Q4:
1247:       jal gameMap3    # Printing Game Map3 by calling child function gameMap3
1248:
1249:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1250:       la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
1251:       syscall     # Print out the output on screen
1252:
1253:   inner_While_Q4:
1254:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1255:       syscall     # Print out the output on screen
1256:
1257:       move $s1, $v0   # move value held in $v0 to register $s1
1258:
1259:       addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
1260:       beq $s1, $t2, if_One_Q4 # If value held in register $s1 is equal to value held
 in $t2 ( 1 ) go to if_One_Q4, else keep going
1261:
1262:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1263:       la $a0, notOne  # Load notOne message into $a0 using la (load Address)
1264:       syscall     # Print out the output on screen
1265:
1266:       j inner_While_Q4    # jump back to inner_While_Q4 until value is equal to 1
1267:
1268:   if_One_Q4:
1269:       # Question 4
1270:       li $a0, 422 # left x-coordinate is 422
1271:       li $a1, 70  # width is 70
1272:       li $a2, 105 # top y-coordinate is 105
1273:       li $a3, 50  # height is 50
1274:       jal rectangleCyan # Jump to rectangleCyan
1275:
1276:       # Horizontal Line 4
1277:       li $a0, 290 # left x-coordinate is 290
1278:       li $a1, 132 # width is 132
1279:       li $a2, 130 # top y-coordinate is 130
1280:       li $a3, 50  # height is 50
```

```
1281:       jal Hline   # Jump to Hline
1282:
1283:       # Number 4
1284:       li $a0, 465 # left x-coordinate is 465
1285:       li $a1, 20  # width is 20
1286:       li $a2, 130 # top y-coordinate is 130
1287:       li $a3, 50  # height is 50
1288:       jal Hline   # Jump to Hline
1289:
1290:       # Number 4
1291:       li $a0, 485 # left x-coordinate is 485
1292:       li $a1, 40  # width is 40
1293:       li $a2, 115 # top y-coordinate is 115
1294:       li $a3, 20  # height is 20
1295:       jal Vline   # Jump to Vline
1296:
1297:       # Number 4
1298:       li $a0, 485 # left x-coordinate is 485
1299:       li $a1, 40  # width is 40
1300:       li $a2, 133 # top y-coordinate is 133
1301:       li $a3, 13  # height is 13
1302:       jal Vline   # Jump to Vline
1303:
1304:       # Number 4
1305:       li $a0, 465 # left x-coordinate is 465
1306:       li $a1, 40  # width is 40
1307:       li $a2, 115 # top y-coordinate is 115
1308:       li $a3, 16  # height is 16
1309:       jal Vline   # Jump to Vline
1310:
1311:       # Q for Question
1312:       li $a0, 432 # left x-coordinate is 432
1313:       li $a1, 20  # width is 20
1314:       li $a2, 120 # top y-coordinate is 120
1315:       li $a3, 50  # height is 50
1316:       jal Hline   # Jump to Hline
1317:
1318:       # Q for Question
1319:       li $a0, 451 # left x-coordinate is 451
1320:       li $a1, 40  # width is 40
1321:       li $a2, 121 # top y-coordinate is 121
1322:       li $a3, 20  # height is 20
1323:       jal Vline   # Jump to Vline
1324:
1325:       # Q for Question
1326:       li $a0, 431 # left x-coordinate is 431
1327:       li $a1, 40  # width is 40
1328:       li $a2, 120 # top y-coordinate is 120
1329:       li $a3, 20  # height is 20
1330:       jal Vline   # Jump to Vline
1331:
1332:       # Q for Question
1333:       li $a0, 443 # left x-coordinate is 443
1334:       li $a1, 15  # width is 15
1335:       li $a2, 133 # top y-coordinate is 133
```

```
1336:       li $a3, 50  # height is 50
1337:       jal Hline   # Jump to Hline
1338:
1339:       # Q for Question
1340:       li $a0, 431 # left x-coordinate is 431
1341:       li $a1, 20  # width is 20
1342:       li $a2, 140 # top y-coordinate is 140
1343:       li $a3, 50  # height is 50
1344:       jal Hline   # Jump to Hline
1345:
1346:       jal PolynomialQuestionFour  # Jump to PolynomialQuestionFour and perform quest
ion 4
1347:
1348:       j exit_While_Q4         # jump exit_While_Q4
1349:
1350:   exit_While_Q4:
1351:
1352:
1353:       #  MapPoints[1][0] -> 40 Points
1354:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1355:       la $a0, GoodJob # Load GoodJob message into $a0 using la (load Address)
1356:       syscall     # Print out the output on screen
1357:
1358:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1359:       la $a0, zeroMap     # Load zeroMap message into $a0 using la (load Address)
1360:       syscall     # Print out the output on screen
1361:
1362:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1363:       la $a0, oneContinue # Load zeroMap message into $a0 using la (load Address)
1364:       syscall     # Print out the output on screen
1365:
1366:
1367:   While_Q5:
1368:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1369:       syscall     # Print out the output on screen
1370:
1371:       move $s0, $v0   # move value held in $v0 to register $s0
1372:
1373:       addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
1374:       beq $s0, $t0, if_Zero_Q5    # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q5, else keep going
1375:
1376:       addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
1377:       beq $s0, $t1, if_One_Q5     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q5, else keep going
1378:
1379:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1380:       la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
1381:       syscall     # Print out the output on screen
1382:
```

```
1383:       j While_Q5  # jump back to While_Q5 until value is equal to 0 or 1
1384:
1385:   if_Zero_Q5:
1386:       jal gameMap4    # Printing Game Map4 by calling child function gameMap4
1387:
1388:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1389:       la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
1390:       syscall     # Print out the output on screen
1391:
1392:   inner_While_Q5:
1393:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1394:       syscall     # Print out the output on screen
1395:
1396:       move $s1, $v0   # move value held in $v0 to register $s1
1397:
1398:       addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
1399:       beq $s1, $t2, if_One_Q5 # If value held in register $s1 is equal to value held
 in $t2 ( 1 ) go to if_One_Q5, else keep going
1400:
1401:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1402:       la $a0, notOne  # Load notOne message into $a0 using la (load Address)
1403:       syscall     # Print out the output on screen
1404:
1405:       j inner_While_Q5    # jump back to inner_While_Q5 until value is equal to 1
1406:
1407:   if_One_Q5:
1408:               # Question 5
1409:       li $a0, 220 # left x-coordinate is 220
1410:       li $a1, 70  # width is 70
1411:       li $a2, 105 # top y-coordinate is 105
1412:       li $a3, 50  # height is 50
1413:       jal rectanglePurple # Jump to rectanglePurple
1414:
1415:       # Horizontal Line 3
1416:       li $a0, 90  # left x-coordinate is 90
1417:       li $a1, 130 # width is 130
1418:       li $a2, 130 # top y-coordinate is 130
1419:       li $a3, 50  # height is 50
1420:       jal Hline   # Jump to Hline
1421:
1422:       # Number 5
1423:       li $a0, 260 # left x-coordinate is 260
1424:       li $a1, 20  # width is 20
1425:       li $a2, 115 # top y-coordinate is 115
1426:       li $a3, 50  # height is 50
1427:       jal Hline   # Jump to Hline
1428:
1429:       # Number 5
1430:       li $a0, 260 # left x-coordinate is 260
1431:       li $a1, 40  # width is 40
1432:       li $a2, 115 # top y-coordinate is 115
1433:       li $a3, 13  # height is 13
```

```
1434:       jal Vline   # Jump to Vline
1435:
1436:       # Number 5
1437:       li $a0, 260 # left x-coordinate is 260
1438:       li $a1, 20  # width is 20
1439:       li $a2, 127 # top y-coordinate is 127
1440:       li $a3, 50  # height is 50
1441:       jal Hline   # Jump to Hline
1442:
1443:       # Number 5
1444:       li $a0, 280 # left x-coordinate is 280
1445:       li $a1, 40  # width is 40
1446:       li $a2, 127 # top y-coordinate is 127
1447:       li $a3, 13  # height is 13
1448:       jal Vline   # Jump to Vline
1449:
1450:       # Number 5
1451:       li $a0, 260 # left x-coordinate is 260
1452:       li $a1, 20  # width is 20
1453:       li $a2, 139 # top y-coordinate is 139
1454:       li $a3, 50  # height is 50
1455:       jal Hline   # Jump to Hline
1456:
1457:       # Q for Question
1458:       li $a0, 230 # left x-coordinate is 230
1459:       li $a1, 20  # width is 20
1460:       li $a2, 120 # top y-coordinate is 120
1461:       li $a3, 50  # height is 50
1462:       jal Hline   # Jump to Hline
1463:
1464:       # Q for Question
1465:       li $a0, 249 # left x-coordinate is 249
1466:       li $a1, 40  # width is 40
1467:       li $a2, 121 # top y-coordinate is 121
1468:       li $a3, 20  # height is 20
1469:       jal Vline   # Jump to Vline
1470:
1471:       # Q for Question
1472:       li $a0, 229 # left x-coordinate is 229
1473:       li $a1, 40  # width is 40
1474:       li $a2, 120 # top y-coordinate is 120
1475:       li $a3, 20  # height is 20
1476:       jal Vline   # Jump to Vline
1477:
1478:       # Q for Question
1479:       li $a0, 241 # left x-coordinate is 241
1480:       li $a1, 15  # width is 15
1481:       li $a2, 133 # top y-coordinate is 133
1482:       li $a3, 50  # height is 50
1483:       jal Hline   # Jump to Hline
1484:
1485:       # Q for Question
1486:       li $a0, 229 # left x-coordinate is 229
1487:       li $a1, 20  # width is 20
1488:       li $a2, 140 # top y-coordinate is 140
```

```
1489:       li $a3, 50  # height is 50
1490:       jal Hline   # Jump to Hline
1491:
1492:       jal PolynomialQuestionFive  # Jump to PolynomialQuestionFive and perform quest
ion 5
1493:
1494:       j exit_While_Q5         # jump exit_While_Q5
1495:
1496:   exit_While_Q5:
1497:
1498:
1499:                   #  MapPoints[1][1] -> 50 Points
1500:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1501:       la $a0, IncredibleJob   # Load IncredibleJob message into $a0 using la (load A
ddress)
1502:       syscall     # Print out the output on screen
1503:
1504:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1505:       la $a0, zeroMap     # Load zeroMap message into $a0 using la (load Address)
1506:       syscall     # Print out the output on screen
1507:
1508:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1509:       la $a0, oneContinue # Load zeroMap message into $a0 using la (load Address)
1510:       syscall     # Print out the output on screen
1511:
1512:
1513:   # While loop for Question 6
1514:   While_Q6:
1515:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1516:       syscall     # Print out the output on screen
1517:
1518:       move $s0, $v0   # move value held in $v0 to register $s0
1519:
1520:       addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
1521:       beq $s0, $t0, if_Zero_Q6    # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q6, else keep going
1522:
1523:       addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
1524:       beq $s0, $t1, if_One_Q6     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q6, else keep going
1525:
1526:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1527:       la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
1528:       syscall     # Print out the output on screen
1529:
1530:       j While_Q6  # jump back to While_Q6 until value is equal to 0 or 1
1531:
1532:   if_Zero_Q6:
1533:       jal gameMap5    # Printing Game Map5 by calling child function gameMap5
1534:
```

```
1535:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1536:       la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
1537:       syscall     # Print out the output on screen
1538:
1539:   inner_While_Q6:
1540:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1541:       syscall     # Print out the output on screen
1542:
1543:       move $s1, $v0   # move value held in $v0 to register $s1
1544:
1545:       addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
1546:       beq $s1, $t2, if_One_Q6     # If value held in register $s1 is equal to value
held in $t2 ( 1 ) go to if_One_Q6, else keep going
1547:
1548:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1549:       la $a0, notOne  # Load notOne message into $a0 using la (load Address)
1550:       syscall     # Print out the output on screen
1551:
1552:       j inner_While_Q6    # jump back to inner_While_Q6 until value is equal to 1
1553:
1554:   if_One_Q6:
1555:       # Question 6
1556:       li $a0, 20  # left x-coordinate is 20
1557:       li $a1, 70  # width is 70
1558:       li $a2, 105 # top y-coordinate is 105
1559:       li $a3, 50  # height is 50
1560:       jal rectangleBlue  # Jump to rectangleBlue
1561:
1562:       # Vertical Line 2
1563:       li $a0, 54  # left x-coordinate is 54
1564:       li $a1, 50  # width is 50
1565:       li $a2, 155 # top y-coordinate is 155
1566:       li $a3, 60  # height is 60
1567:       jal Vline   # Jump to Vline
1568:
1569:       # Number 6
1570:       li $a0, 61  # left x-coordinate is 61
1571:       li $a1, 20  # width is 20
1572:       li $a2, 115 # top y-coordinate is 115
1573:       li $a3, 50  # height is 50
1574:       jal Hline   # Jump to Hline
1575:
1576:       # Number 6
1577:       li $a0, 61  # left x-coordinate is 61
1578:       li $a1, 40  # width is 40
1579:       li $a2, 115 # top y-coordinate is 115
1580:       li $a3, 13  # height is 13
1581:       jal Vline   # Jump to Vline
1582:
1583:       # Number 6
1584:       li $a0, 61  # left x-coordinate is 61
1585:       li $a1, 20  # width is 20
```

```
1586:       li $a2, 127 # top y-coordinate is 127
1587:       li $a3, 50  # height is 50
1588:       jal Hline   # Jump to Hline
1589:
1590:       # Number 6
1591:       li $a0, 61  # left x-coordinate is 61
1592:       li $a1, 40  # width is 40
1593:       li $a2, 127 # top y-coordinate is 127
1594:       li $a3, 13  # height is 13
1595:       jal Vline   # Jump to Vline
1596:
1597:       # Number 6
1598:       li $a0, 81  # left x-coordinate is 81
1599:       li $a1, 40  # width is 40
1600:       li $a2, 127 # top y-coordinate is 127
1601:       li $a3, 13  # height is 13
1602:       jal Vline   # Jump to Vline
1603:
1604:       # Number 6
1605:       li $a0, 61  # left x-coordinate is 61
1606:       li $a1, 20  # width is 20
1607:       li $a2, 139 # top y-coordinate is 139
1608:       li $a3, 50  # height is 50
1609:       jal Hline   # Jump to Hline
1610:
1611:       # Q for Question
1612:       li $a0, 30  # left x-coordinate is 30
1613:       li $a1, 20  # width is 20
1614:       li $a2, 120 # top y-coordinate is 120
1615:       li $a3, 50  # height is 50
1616:       jal Hline   # Jump to Hline
1617:
1618:       # Q for Question
1619:       li $a0, 49  # left x-coordinate is 49
1620:       li $a1, 40  # width is 40
1621:       li $a2, 121 # top y-coordinate is 121
1622:       li $a3, 20  # height is 20
1623:       jal Vline   # Jump to Vline
1624:
1625:       # Q for Question
1626:       li $a0, 29  # left x-coordinate is 29
1627:       li $a1, 40  # width is 40
1628:       li $a2, 120 # top y-coordinate is 120
1629:       li $a3, 20  # height is 20
1630:       jal Vline   # Jump to Vline
1631:
1632:       # Q for Question
1633:       li $a0, 41  # left x-coordinate is 41
1634:       li $a1, 15  # width is 15
1635:       li $a2, 133 # top y-coordinate is 133
1636:       li $a3, 50  # height is 50
1637:       jal Hline   # Jump to Hline
1638:
1639:       # Q for Question
1640:       li $a0, 29  # left x-coordinate is 29
```

```
1641:       li $a1, 20  # width is 20
1642:       li $a2, 140 # top y-coordinate is 140
1643:       li $a3, 50  # height is 50
1644:       jal Hline   # Jump to Hline
1645:
1646:       jal PolynomialQuestionSix   # Jump to PolynomialQuestionSix and perform questi
on 6
1647:
1648:       j exit_While_Q6         # jump exit_While_Q6
1649:
1650:   exit_While_Q6:
1651:
1652:
1653:       # MapPoints[1][2] -> 60 Points
1654:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1655:       la $a0, prizeTwo.1  # Load prizeTwo.1 message into $a0 using la (load Address)
1656:       syscall     # Print out the output on screen
1657:
1658:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1659:       la $a0, prizeTwo.2  # Load prizeTwo.2 message into $a0 using la (load Address)
1660:       syscall     # Print out the output on screen
1661:
1662:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1663:       la $a0, prizeTwo.3  # Load prizeTwo.3 message into $a0 using la (load Address)
1664:       syscall     # Print out the output on screen
1665:
1666:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1667:       la $a0, prizeTwo.4  # Load prizeTwo.4 message into $a0 using la (load Address)
1668:       syscall     # Print out the output on screen
1669:
1670:
1671:                   # ADDING VALUES TO ArrayQ7Int QUESTION 7: Creating the values to add
 to 1D ArrayQ7Int Question 7
1672:       addi $s2, $zero, 3  # initialize/add 3 to register $s2
1673:       addi $s4, $zero, 39 # initialize/add 39 to register $s4
1674:       addi $s5, $zero, 168    # initialize/add 168 to register $s5
1675:           addi $s6, $zero, 276    # initialize/add 276 to register $s6
1676:       addi $s7, $zero, 144    # initialize/add 144 to register $s7
1677:
1678:       # Creating a 0 index by initializing index $t0 = 0
1679:       addi $t0, $zero, 0  # initialize index 0 by adding 0 to register $t0
1680:
1681:       # Storing values into 1D ArrayQ6Int Question 6 using sw (store word)
1682:       sw $s2, ArrayQ7Int($t0)    # Storing value held in register $s2, at index 0 in
 ArrayQ7Int Question 7
1683:       addi $t0, $t0, 4    # Updating register $t0 to index 4 by adding 4 to it
1684:       sw $s4, ArrayQ7Int($t0)    # Storing value held in register $s4, at index 4 in
 ArrayQ7Int Question 7
1685:       addi $t0, $t0, 4    # Updating register $t0 to index 8 by adding 4 to it
1686:       sw $s5, ArrayQ7Int($t0)    # Storing value held in register $s5, at index 8 in
 ArrayQ7Int Question 7
```

```
1687:       addi $t0, $t0, 4    # Updating register $t0 to index 12 by adding 4 to it
1688:       sw $s6, ArrayQ7Int($t0)    # Storing value held in register $s6, at index 12 i
n ArrayQ7Int Question 7
1689:       addi $t0, $t0, 4    # Updating register $t0 to index 16 by adding 4 to it
1690:       sw $s7, ArrayQ7Int($t0)    # Storing value held in register $s7, at index 16 i
n ArrayQ7Int Question 7
1691:
1692:       # ADDING VALUES TO ArrayQ7Ans1 QUESTION 7: Creating the values to add to 1D Ar
rayQ7Ans1 Question 7
1693:       addi $t4, $zero, 1  # initialize/add 1 to register $t4
1694:       addi $t5, $zero, 2  # initialize/add 2 to register $t5
1695:           addi $t6, $zero, 4  # initialize/add 4 to register $t6
1696:       addi $t7, $zero, 6  # initialize/add 6 to register $t7
1697:
1698:       # Creating a 0 index by initializing index $t0 = 0
1699:       addi $t3, $zero, 0  # initialize index 0 by adding 0 to register $t3
1700:
1701:       sw $t4, ArrayQ7Ans1($t3)    # Storing value held in register $t4, at index 4 i
n ArrayQ7Ans1 Question 7
1702:       addi $t3, $t3, 4    # Updating register $t0 to index 8 by adding 4 to it
1703:       sw $t5, ArrayQ7Ans1($t3)    # Storing value held in register $t5, at index 8 i
n ArrayQ7Ans1 Question 7
1704:       addi $t3, $t3, 4    # Updating register $t0 to index 12 by adding 4 to it
1705:       sw $t6, ArrayQ7Ans1($t3)    # Storing value held in register $t6, at index 12
in ArrayQ7Ans1 Question 7
1706:       addi $t3, $t3, 4    # Updating register $t0 to index 16 by adding 4 to it
1707:       sw $t7, ArrayQ7Ans1($t3)    # Storing value held in register $t7, at index 16
in ArrayQ7Ans1 Question 7
1708:
1709:
1710:       # While loop for prize or Question 7
1711:   While_Q7_:
1712:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1713:       syscall     # Print out the output on screen
1714:
1715:       move $s0, $v0   # move value held in $v0 to register $s0
1716:
1717:       addi $t0, $zero, 0      # Add 0 to register $t0 in order to do comparisons
1718:       beq $s0, $t0, if_Zero_Q7_   # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q7_, else keep going
1719:
1720:       addi $t1, $zero, 1      # Add 1 to register $t1 in order to do comparisons
1721:       beq $s0, $t1, if_One_Q7_    # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q7_, else keep going
1722:
1723:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1724:       la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
1725:       syscall     # Print out the output on screen
1726:
1727:       j While_Q7_ # jump back to While_Q7_ until value is equal to 0 or 1
1728:
1729:   if_Zero_Q7_:
1730:       jal SilverMedal     # Printing Silver Medal Prize by calling child function Si
```

```
lverMedal
1731:
1732:       la $a0, beep9       # Load the address of "beep9" into $a0
1733:       la $a1, duration9   # Load the address of "duration9" into $a1
1734:       la $a2, instrument9 # Load the address of "instrument9" into $a2
1735:       la $a3, volume9     # Load the address of "volume9" into $a3
1736:
1737:       lb $a0, 0($a0)      # Load the value of "beep9" into $a0
1738:       lb $a1, 0($a1)      # Load the value of "duration9" into $a1
1739:       lb $a2, 0($a2)      # Load the value of "instrument9" into $a2
1740:       lb $a3, 0($a3)      # Load the value of "volume9" into $a3
1741:
1742:           li $v0, 31  # Use the "play note" system call
1743:           syscall     # Print out the output on screen
1744:
1745:
1746:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1747:       li $a0, 800 # wait for 800 millisecond
1748:       syscall     # Print out the output on screen
1749:
1750:
1751:       la $a0, beep7       # Load the address of "beep7" into $a0
1752:       la $a1, duration7   # Load the address of "duratin7" into $a1
1753:       la $a2, instrument7 # Load the address of "instrument7" into $a2
1754:       la $a3, volume7     # Load the address of "volume7" into $a3
1755:
1756:       lb $a0, 0($a0)      # Load the value of "beep7" into $a0
1757:       lb $a1, 0($a1)      # Load the value of "duration7" into $a1
1758:       lb $a2, 0($a2)      # Load the value of "instrument7" into $a2
1759:       lb $a3, 0($a3)      # Load the value of "volume7" into $a3
1760:
1761:       li $v0, 31  # Use the "play note" system call
1762:           syscall     # Print out the output on screen
1763:
1764:
1765:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1766:       li $a0, 300 # wait for 300 millisecond
1767:       syscall     # Print out the output on screen
1768:
1769:
1770:       la $a0, beep7       # Load the address of "beep7" into $a0
1771:       la $a1, duration7   # Load the address of "duratin7" into $a1
1772:       la $a2, instrument7 # Load the address of "instrument7" into $a2
1773:       la $a3, volume7     # Load the address of "volume7" into $a3
1774:
1775:       lb $a0, 0($a0)      # Load the value of "beep7" into $a0
1776:       lb $a1, 0($a1)      # Load the value of "duration7" into $a1
1777:       lb $a2, 0($a2)      # Load the value of "instrument7" into $a2
1778:       lb $a3, 0($a3)      # Load the value of "volume7" into $a3
1779:
1780:       li $v0, 31  # Use the "play note" system call
1781:           syscall     # Print out the output on screen
1782:
1783:
1784:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
```

```
1785:       li $a0, 300 # wait for 300 millisecond
1786:       syscall     # Print out the output on screen
1787:
1788:
1789:           la $a0, beep7       # Load the address of "beep7" into $a0
1790:       la $a1, duration7   # Load the address of "duratin7" into $a1
1791:       la $a2, instrument7 # Load the address of "instrument7" into $a2
1792:       la $a3, volume7     # Load the address of "volume7" into $a3
1793:
1794:       lb $a0, 0($a0)      # Load the value of "beep1" into $a0
1795:       lb $a1, 0($a1)      # Load the value of "duration1" into $a1
1796:       lb $a2, 0($a2)      # Load the value of "instrument1" into $a2
1797:       lb $a3, 0($a3)      # Load the value of "volume1" into $a3
1798:
1799:       li $v0, 31  # Use the "play note" system call
1800:           syscall     # Print out the output on screen
1801:
1802:
1803:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1804:       li $a0, 300 # wait for 300 millisecond
1805:       syscall     # Print out the output on screen
1806:
1807:
1808:           la $a0, beep9       # Load the address of "beep9" into $a0
1809:       la $a1, duration9   # Load the address of "duration9" into $a1
1810:       la $a2, instrument9 # Load the address of "instrument9" into $a2
1811:       la $a3, volume9     # Load the address of "volume9" into $a3
1812:
1813:       lb $a0, 0($a0)      # Load the value of "beep9" into $a0
1814:       lb $a1, 0($a1)      # Load the value of "duration9" into $a1
1815:       lb $a2, 0($a2)      # Load the value of "instrument9" into $a2
1816:       lb $a3, 0($a3)      # Load the value of "volume9" into $a3
1817:
1818:           li $v0, 31  # Use the "play note" system call
1819:           syscall     # Print out the output on screen
1820:
1821:
1822:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1823:       li $a0, 500 # wait for 500 millisecond
1824:       syscall     # Print out the output on screen
1825:
1826:
1827:       la $a0, beep8       # Load the address of "beep8" into $a0
1828:       la $a1, duration8   # Load the address of "duration8" into $a1
1829:       la $a2, instrument8 # Load the address of "instrument8" into $a2
1830:       la $a3, volume8     # Load the address of "volume8" into $a3
1831:
1832:       lb $a0, 0($a0)      # Load the value of "beep8" into $a0
1833:       lb $a1, 0($a1)      # Load the value of "duration8" into $a1
1834:       lb $a2, 0($a2)      # Load the value of "instrument8" into $a2
1835:       lb $a3, 0($a3)      # Load the value of "volume8" into $a3
1836:
1837:           li $v0, 31  # Use the "play note" system call
1838:           syscall     # Print out the output on screen
1839:
```

```
1840:
1841:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
1842:       li $a0, 500 # wait for 500 millisecond
1843:       syscall     # Print out the output on screen
1844:
1845:
1846:       la $a0, beep3       # Load the address of "beep3" into $a0
1847:       la $a1, duration3   # Load the address of "duration3" into $a1
1848:       la $a2, instrument3 # Load the address of "instrument3" into $a2
1849:       la $a3, volume3     # Load the address of "volume3" into $a3
1850:
1851:       lb $a0, 0($a0)      # Load the value of "beep3" into $a0
1852:       lb $a1, 0($a1)      # Load the value of "duration3" into $a1
1853:       lb $a2, 0($a2)      # Load the value of "instrument3" into $a2
1854:       lb $a3, 0($a3)      # Load the value of "volume3" into $a3
1855:
1856:           li $v0, 31  # Use the "play note" system call
1857:           syscall     # Print out the output on screen
1858:
1859:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1860:       la $a0, prizeTwoIfZero  # Load prizeTwoIfZero message into $a0 using la (load
Address)
1861:       syscall     # Print out the output on screen
1862:
1863:       j _exit_While_Q7_           # jump _exit_While_Q7_
1864:
1865:   if_One_Q7_:
1866:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1867:       la $a0, prizeTwoIfOne   # Load prizeTwoIfOne message into $a0 using la (load A
ddress)
1868:       syscall     # Print out the output on screen
1869:
1870:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1871:       la $a0, zeroMap # Load zeroMap message into $a0 using la (load Address)
1872:       syscall     # Print out the output on screen
1873:
1874:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1875:       la $a0, oneContinue # Load oneContinue message into $a0 using la (load Address
)
1876:       syscall     # Print out the output on screen
1877:
1878:       j exit_While_Q7_            # jump exit_While_Q7_
1879:
1880:   _exit_While_Q7_:
1881:       # Telling the system to stop executing the program
1882:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
1883:       syscall     # Print out the output on screen
1884:
1885:
1886:   exit_While_Q7_:
```

```
1887:
1888:
1889:   # While loop for Question 7
1890:   While_Q7:
1891:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1892:       syscall     # Print out the output on screen
1893:
1894:       move $s0, $v0   # move value held in $v0 to register $s0
1895:
1896:       addi $t0, $zero, 0  # Add 0 to register $t0 in order to do comparisons
1897:       beq $s0, $t0, if_Zero_Q7    # If value held in register $t0 is equal to value
held in $t0 (0) go to if_Zero_Q7, else keep going
1898:
1899:       addi $t1, $zero, 1  # Add 1 to register $t1 in order to do comparisons
1900:       beq $s0, $t1, if_One_Q7     # If value held in register $s0 is equal to value
held in $t1 (1) go to if_One_Q7, else keep going
1901:
1902:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1903:       la $a0, notZeroOne  # Load notZeroOne message into $a0 using la (load Address)
1904:       syscall     # Print out the output on screen
1905:
1906:       j While_Q7  # jump back to While_Q7 until value is equal to 0 or 1
1907:
1908:   if_Zero_Q7:
1909:       jal gameMap6    # Printing Game Map6 by calling child function gameMap6
1910:
1911:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1912:       la $a0, enter_One   # Load enter_One message into $a0 using la (load Address)
1913:       syscall     # Print out the output on screen
1914:
1915:   inner_While_Q7:
1916:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
1917:       syscall     # Print out the output on screen
1918:
1919:       move $s1, $v0   # move value held in $v0 to register $s1
1920:
1921:       addi $t2, $zero, 1  # Add 1 to register $t2 in order to do comparisons
1922:       beq $s1, $t2, if_One_Q7 # If value held in register $s1 is equal to value held
 in $t2 ( 1 ) go to if_One_Q7, else keep going
1923:
1924:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
1925:       la $a0, notOne  # Load notOne message into $a0 using la (load Address)
1926:       syscall     # Print out the output on screen
1927:
1928:       j inner_While_Q7    # jump back to inner_While_Q4 until value is equal to 1
1929:
1930:   if_One_Q7:
1931:               # Horizontal Line 5
1932:       li $a0, 54  # left x-coordinate is 54
1933:       li $a1, 170 # width is 170
```

```
1934:       li $a2, 215 # top y-coordinate is 215
1935:       li $a3, 50  # height is 50
1936:       jal Hline   # Jump to Hline
1937:
1938:       # Question 7
1939:       li $a0, 220 # left x-coordinate is 220
1940:       li $a1, 70  # width is 70
1941:       li $a2, 190 # top y-coordinate is 190
1942:       li $a3, 50  # height is 50
1943:       jal rectangleWhite # Jump to rectangleWhite
1944:
1945:       # Number 7
1946:       li $a0, 258 # left x-coordinate is 258
1947:       li $a1, 18  # width is 18
1948:       li $a2, 205 # top y-coordinate is 205
1949:       li $a3, 50  # height is 50
1950:       jal HlineBlack  # Jump to HlineBlack
1951:
1952:       # Number 7
1953:       li $a0, 275 # left x-coordinate is 275
1954:       li $a1, 40  # width is 40
1955:       li $a2, 215 # top y-coordinate is 215
1956:       li $a3, 13  # height is 13
1957:       jal VlineBlack  # Jump to VlineBlack
1958:
1959:       # Number 7
1960:       li $a0, 275 # left x-coordinate is 275
1961:       li $a1, 40  # width is 40
1962:       li $a2, 205 # top y-coordinate is 205
1963:       li $a3, 13  # height is 13
1964:       jal VlineBlack  # Jump to VlineBlack
1965:
1966:       # Number 7
1967:       li $a0, 265 # left x-coordinate is 265
1968:       li $a1, 20  # width is 20
1969:       li $a2, 215 # top y-coordinate is 215
1970:       li $a3, 50  # height is 50
1971:       jal HlineBlack  # Jump to HlineBlack
1972:
1973:       # Q for Question
1974:       li $a0, 230 # left x-coordinate is 230
1975:       li $a1, 20  # width is 20
1976:       li $a2, 205 # top y-coordinate is 205
1977:       li $a3, 50  # height is 50
1978:       jal HlineBlack  # Jump to HlineBlack
1979:
1980:       # Q for Question
1981:       li $a0, 249 # left x-coordinate is 249
1982:       li $a1, 40  # width is 40
1983:       li $a2, 206 # top y-coordinate is 206
1984:       li $a3, 20  # height is 20
1985:       jal VlineBlack  # Jump to VlineBlack
1986:
1987:       # Q for Question
1988:       li $a0, 229 # left x-coordinate is 229
```

```
1989:       li $a1, 40  # width is 40
1990:       li $a2, 205 # top y-coordinate is 205
1991:       li $a3, 20  # height is 20
1992:       jal VlineBlack  # Jump to VlineBlack
1993:
1994:       # Q for Question
1995:       li $a0, 241 # left x-coordinate is 241
1996:       li $a1, 15  # width is 15
1997:       li $a2, 218 # top y-coordinate is 218
1998:       li $a3, 50  # height is 50
1999:       jal HlineBlack  # Jump to HlineBlack
2000:
2001:       # Q for Question
2002:       li $a0, 229 # left x-coordinate is 229
2003:       li $a1, 20  # width is 20
2004:       li $a2, 225 # top y-coordinate is 225
2005:       li $a3, 50  # height is 50
2006:       jal HlineBlack  # Jump to HlineBlack
2007:
2008:       jal PolynomialQuestionSeven # Jump to PolynomialQuestionSeven and perform ques
tion 7
2009:
2010:       j exit_While_Q7         # jump exit_While_Q7
2011:
2012:   exit_While_Q7:
2013:
2014:
2015:       # MapPoints[2][0] -> 70 Points
2016:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2017:       la $a0, prizeThree  # Load prizeThree message into $a0 using la (load Address)
2018:       syscall     # Print out the output on screen
2019:
2020:       jal GoldMedal
2021:
2022:       la $a0, beep9       # Load the address of "beep9" into $a0
2023:       la $a1, duration9   # Load the address of "duration9" into $a1
2024:       la $a2, instrument9 # Load the address of "instrument9" into $a2
2025:       la $a3, volume9     # Load the address of "volume9" into $a3
2026:
2027:       lb $a0, 0($a0)      # Load the value of "beep9" into $a0
2028:       lb $a1, 0($a1)      # Load the value of "duration9" into $a1
2029:       lb $a2, 0($a2)      # Load the value of "instrument9" into $a2
2030:       lb $a3, 0($a3)      # Load the value of "volume9" into $a3
2031:
2032:           li $v0, 31  # Use the "play note" system call
2033:           syscall     # Print out the output on screen
2034:
2035:
2036:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
2037:       li $a0, 800 # wait for 800 millisecond
2038:       syscall     # Print out the output on screen
2039:
2040:
2041:       la $a0, beep7       # Load the address of "beep7" into $a0
```

```
2042:       la $a1, duration7   # Load the address of "duratin7" into $a1
2043:       la $a2, instrument7 # Load the address of "instrument7" into $a2
2044:       la $a3, volume7     # Load the address of "volume7" into $a3
2045:
2046:       lb $a0, 0($a0)      # Load the value of "beep7" into $a0
2047:       lb $a1, 0($a1)      # Load the value of "duration7" into $a1
2048:       lb $a2, 0($a2)      # Load the value of "instrument7" into $a2
2049:       lb $a3, 0($a3)      # Load the value of "volume7" into $a3
2050:
2051:       li $v0, 31  # Use the "play note" system call
2052:           syscall     # Print out the output on screen
2053:
2054:
2055:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
2056:       li $a0, 300 # wait for 300 millisecond
2057:       syscall     # Print out the output on screen
2058:
2059:
2060:       la $a0, beep7       # Load the address of "beep7" into $a0
2061:       la $a1, duration7   # Load the address of "duratin7" into $a1
2062:       la $a2, instrument7 # Load the address of "instrument7" into $a2
2063:       la $a3, volume7     # Load the address of "volume7" into $a3
2064:
2065:       lb $a0, 0($a0)      # Load the value of "beep7" into $a0
2066:       lb $a1, 0($a1)      # Load the value of "duration7" into $a1
2067:       lb $a2, 0($a2)      # Load the value of "instrument7" into $a2
2068:       lb $a3, 0($a3)      # Load the value of "volume7" into $a3
2069:
2070:       li $v0, 31  # Use the "play note" system call
2071:           syscall     # Print out the output on screen
2072:
2073:
2074:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
2075:       li $a0, 300 # wait for 300 millisecond
2076:       syscall     # Print out the output on screen
2077:
2078:
2079:           la $a0, beep7       # Load the address of "beep7" into $a0
2080:       la $a1, duration7   # Load the address of "duratin7" into $a1
2081:       la $a2, instrument7 # Load the address of "instrument7" into $a2
2082:       la $a3, volume7     # Load the address of "volume7" into $a3
2083:
2084:       lb $a0, 0($a0)      # Load the value of "beep1" into $a0
2085:       lb $a1, 0($a1)      # Load the value of "duration1" into $a1
2086:       lb $a2, 0($a2)      # Load the value of "instrument1" into $a2
2087:       lb $a3, 0($a3)      # Load the value of "volume1" into $a3
2088:
2089:       li $v0, 31  # Use the "play note" system call
2090:           syscall     # Print out the output on screen
2091:
2092:
2093:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
2094:       li $a0, 300 # wait for 300 millisecond
2095:       syscall     # Print out the output on screen
2096:
```

```
2097:
2098:           la $a0, beep9       # Load the address of "beep9" into $a0
2099:       la $a1, duration9   # Load the address of "duration9" into $a1
2100:       la $a2, instrument9 # Load the address of "instrument9" into $a2
2101:       la $a3, volume9     # Load the address of "volume9" into $a3
2102:
2103:       lb $a0, 0($a0)      # Load the value of "beep9" into $a0
2104:       lb $a1, 0($a1)      # Load the value of "duration9" into $a1
2105:       lb $a2, 0($a2)      # Load the value of "instrument9" into $a2
2106:       lb $a3, 0($a3)      # Load the value of "volume9" into $a3
2107:
2108:           li $v0, 31  # Use the "play note" system call
2109:           syscall     # Print out the output on screen
2110:
2111:
2112:           li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
2113:       li $a0, 500 # wait for 500 millisecond
2114:       syscall     # Print out the output on screen
2115:
2116:
2117:       la $a0, beep8       # Load the address of "beep8" into $a0
2118:       la $a1, duration8   # Load the address of "duration8" into $a1
2119:       la $a2, instrument8 # Load the address of "instrument8" into $a2
2120:       la $a3, volume8     # Load the address of "volume8" into $a3
2121:
2122:       lb $a0, 0($a0)      # Load the value of "beep8" into $a0
2123:       lb $a1, 0($a1)      # Load the value of "duration8" into $a1
2124:       lb $a2, 0($a2)      # Load the value of "instrument8" into $a2
2125:       lb $a3, 0($a3)      # Load the value of "volume8" into $a3
2126:
2127:           li $v0, 31  # Use the "play note" system call
2128:           syscall     # Print out the output on screen
2129:
2130:       jal printNewline
2131:       jal printNewline
2132:
2133:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2134:       la $a0, zeroMap     # Load zeroMap message into $a0 using la (load Address)
2135:       syscall     # Print out the output on screen
2136:
2137:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2138:       la $a0, numExit     # Load numExit message into $a0 using la (load Address)
2139:       syscall     # Print out the output on screen
2140:
2141:   While_Q_:
2142:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
2143:       syscall     # Print out the output on screen
2144:
2145:       move $s0, $v0   # move value held in $v0 to register $s0
2146:
2147:       addi $t0, $zero, 0      # Add 0 to register $t0 in order to do comparisons
2148:       beq $s0, $t0, if_Zero_Q_    # If value held in register $t0 is equal to value
```

```
held in $t0 (0) go to if_Zero_Q_, else keep going
2149:
2150:       j if_num_Q_ # jump to if_num_Q_
2151:
2152:   if_Zero_Q_:
2153:       jal gameMap7    # Printing Game Map6 by calling child function gameMap6
2154:
2155:       la $a0, beep3       # Load the address of "beep3" into $a0
2156:       la $a1, duration3   # Load the address of "duration3" into $a1
2157:       la $a2, instrument3 # Load the address of "instrument3" into $a2
2158:       la $a3, volume3     # Load the address of "volume3" into $a3
2159:
2160:       lb $a0, 0($a0)      # Load the value of "beep3" into $a0
2161:       lb $a1, 0($a1)      # Load the value of "duration3" into $a1
2162:       lb $a2, 0($a2)      # Load the value of "instrument3" into $a2
2163:       lb $a3, 0($a3)      # Load the value of "volume3" into $a3
2164:
2165:           li $v0, 31  # Use the "play note" system call
2166:           syscall     # Print out the output on screen
2167:
2168:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2169:       la $a0, prizeTwoIfZero  # Load prizeTwoIfZero message into $a0 using la (load
Address)
2170:       syscall     # Print out the output on screen
2171:
2172:       j exit_While_Q_
2173:   if_num_Q_:
2174:       la $a0, beep3       # Load the address of "beep3" into $a0
2175:       la $a1, duration3   # Load the address of "duration3" into $a1
2176:       la $a2, instrument3 # Load the address of "instrument3" into $a2
2177:       la $a3, volume3     # Load the address of "volume3" into $a3
2178:
2179:       lb $a0, 0($a0)      # Load the value of "beep3" into $a0
2180:       lb $a1, 0($a1)      # Load the value of "duration3" into $a1
2181:       lb $a2, 0($a2)      # Load the value of "instrument3" into $a2
2182:       lb $a3, 0($a3)      # Load the value of "volume3" into $a3
2183:
2184:           li $v0, 31  # Use the "play note" system call
2185:           syscall     # Print out the output on screen
2186:
2187:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2188:       la $a0, prizeTwoIfZero  # Load prizeTwoIfZero message into $a0 using la (load
Address)
2189:       syscall     # Print out the output on screen
2190:
2191:       j exit_While_Q_
2192:
2193:   exit_While_Q_:
2194:
2195:
2196:
2197:
2198:
```

```
2199:       # Telling the system to stop executing the program
2200:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
2201:       syscall     # Print out the output on screen
2202:
2203:
2204:
2205:
2206:   # newline function
2207:   printNewline:
2208:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2209:       la $a0, newline     # Load newLine into $a0 using la (load Address)
2210:       syscall     # Print out the output on screen
2211:
2212:       jr $ra      # Finish and return to main and continue executing
2213:
2214:
2215:   rectangleRed:
2216:       # $a0 is xmin (i.e., left edge; must be within the display)
2217:       # $a1 is width (must be nonnegative and within the display)
2218:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2219:       # $a3 is height (must be nonnegative and within the display)
2220:
2221:       beq $a1, $zero, rectangleRedReturn # zero width: draw nothing
2222:       beq $a3, $zero, rectangleRedReturn # zero height: draw nothing
2223:
2224:       li $t0, 0x00FF0000  # color: Red
2225:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2226:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2227:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2228:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2229:       sll $a1, $a1, 2     # Shift logical left by 2
2230:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2231:       sll $a3, $a3, 11    # SHift logical left by 11
2232:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2233:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2234:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2235:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2236:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2237:       li $t4, 0x800       # bytes per display row
2238:
2239:   rectangleRedYloop:
2240:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2241:
2242:   rectangleRedXloop:
2243:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2244:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2245:       bne $t3, $t2, rectangleRedXloop     # keep going if not past the right edge of
 the rectangle
2246:
2247:       addu $a2, $a2, $t4      # advace one row worth for the left edge
```

```
2248:       addu $t2, $t2, $t4      # and right edge pointers
2249:       bne $a2, $a3, rectangleRedYloop     # keep going if not off the bottom of the
rectangle
2250:
2251:   rectangleRedReturn:
2252:       jr $ra      # Finish and return to main and continue executing
2253:       syscall         # Print out the output on screen
2254:
2255:
2256:   rectangleGreen:
2257:       # $a0 is xmin (i.e., left edge; must be within the display)
2258:       # $a1 is width (must be nonnegative and within the display)
2259:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2260:       # $a3 is height (must be nonnegative and within the display)
2261:
2262:       beq $a1, $zero, rectangleGreenReturn    # zero width: draw nothing
2263:       beq $a3, $zero, rectangleGreenReturn    # zero height: draw nothing
2264:
2265:       li $t0, 0X0000FF00   # color: Green
2266:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2267:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2268:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2269:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2270:       sll $a1, $a1, 2     # Shift logical left by 2
2271:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2272:       sll $a3, $a3, 11    # SHift logical left by 11
2273:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2274:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2275:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2276:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2277:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2278:       li $t4, 0x800       # bytes per display row
2279:
2280:   rectangleGreenYloop:
2281:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2282:
2283:   rectangleGreenXloop:
2284:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2285:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2286:       bne $t3, $t2, rectangleGreenXloop   # keep going if not past the right edge of
 the rectangle
2287:
2288:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2289:       addu $t2, $t2, $t4  # and right edge pointers
2290:       bne $a2, $a3, rectangleGreenYloop   # keep going if not off the bottom of the
rectangle
2291:
2292:   rectangleGreenReturn:
2293:       jr $ra      # Finish and return to main and continue executing
2294:       syscall         # Print out the output on screen
2295:
2296:
```

```
2297:   rectangleYellow:
2298:       # $a0 is xmin (i.e., left edge; must be within the display)
2299:       # $a1 is width (must be nonnegative and within the display)
2300:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2301:       # $a3 is height (must be nonnegative and within the display)
2302:
2303:       beq $a1, $zero, rectangleYellowReturn # zero width: draw nothing
2304:       beq $a3, $zero, rectangleYellowReturn # zero height: draw nothing
2305:
2306:       li $t0, 0X00FFFF00  # color: Yellow
2307:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2308:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2309:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2310:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2311:       sll $a1, $a1, 2     # Shift logical left by 2
2312:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2313:       sll $a3, $a3, 11    # SHift logical left by 11
2314:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2315:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2316:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2317:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2318:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2319:       li $t4, 0x800       # bytes per display row
2320:
2321:   rectangleYellowYloop:
2322:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2323:
2324:   rectangleYellowXloop:
2325:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2326:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2327:       bne $t3, $t2, rectangleYellowXloop  # keep going if not past the right edge of
 the rectangle
2328:
2329:       addu $a2, $a2, $t4      # advace one row worth for the left edge
2330:       addu $t2, $t2, $t4      # and right edge pointers
2331:       bne $a2, $a3, rectangleYellowYloop  # keep going if not off the bottom of the
rectangle
2332:
2333:   rectangleYellowReturn:
2334:       jr $ra      # Finish and return to main and continue executing
2335:       syscall         # Print out the output on screen
2336:
2337:
2338:   rectangleBlue:
2339:       # $a0 is xmin (i.e., left edge; must be within the display)
2340:       # $a1 is width (must be nonnegative and within the display)
2341:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2342:       # $a3 is height (must be nonnegative and within the display)
2343:
2344:       beq $a1, $zero, rectangleBlueReturn  # zero width: draw nothing
2345:       beq $a3, $zero, rectangleBlueReturn     # zero height: draw nothing
2346:
```

```
2347:       li $t0, 0X000000FF  # color: Blue
2348:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2349:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2350:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2351:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2352:       sll $a1, $a1, 2     # Shift logical left by 2
2353:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2354:       sll $a3, $a3, 11    # SHift logical left by 11
2355:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2356:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2357:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2358:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2359:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2360:       li $t4, 0x800       # bytes per display row
2361:
2362:   rectangleBlueYloop:
2363:       move $t3, $a2   # pointer to current pixel for X loop; start at left edge
2364:
2365:   rectangleBlueXloop:
2366:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2367:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2368:       bne $t3, $t2, rectangleBlueXloop    # keep going if not past the right edge of
 the rectangle
2369:
2370:       addu $a2, $a2, $t4      # advace one row worth for the left edge
2371:       addu $t2, $t2, $t4      # and right edge pointers
2372:       bne $a2, $a3, rectangleBlueYloop    # keep going if not off the bottom of the
rectangle
2373:
2374:   rectangleBlueReturn:
2375:       jr $ra      # Finish and return to main and continue executing
2376:       syscall         # Print out the output on screen
2377:
2378:
2379:   rectanglePurple:
2380:       # $a0 is xmin (i.e., left edge; must be within the display)
2381:       # $a1 is width (must be nonnegative and within the display)
2382:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2383:       # $a3 is height (must be nonnegative and within the display)
2384:
2385:       beq $a1, $zero, rectanglePurpleReturn       # zero width: draw nothing
2386:       beq $a3, $zero, rectanglePurpleReturn       # zero height: draw nothing
2387:
2388:       li $t0, 0X00FF00FF  # color: Purple
2389:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2390:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2391:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2392:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2393:       sll $a1, $a1, 2     # Shift logical left by 2
2394:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2395:       sll $a3, $a3, 11    # SHift logical left by 11
```

```
2396:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2397:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2398:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2399:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2400:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2401:       li $t4, 0x800       # bytes per display row
2402:
2403:   rectanglePurpleYloop:
2404:       move $t3, $a2   # pointer to current pixel for X loop; start at left edge
2405:
2406:   rectanglePurpleXloop:
2407:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2408:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2409:       bne $t3, $t2, rectanglePurpleXloop  # keep going if not past the right edge of
 the rectangle
2410:
2411:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2412:       addu $t2, $t2, $t4  # and right edge pointers
2413:       bne $a2, $a3, rectanglePurpleYloop  # keep going if not off the bottom of the
rectangle
2414:
2415:   rectanglePurpleReturn:
2416:       jr $ra      # Finish and return to main and continue executing
2417:       syscall         # Print out the output on screen
2418:
2419:
2420:   rectangleCyan:
2421:       # $a0 is xmin (i.e., left edge; must be within the display)
2422:       # $a1 is width (must be nonnegative and within the display)
2423:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2424:       # $a3 is height (must be nonnegative and within the display)
2425:
2426:       beq $a1, $zero, rectangleCyanReturn     # zero width: draw nothing
2427:       beq $a3, $zero, rectangleCyanReturn     # zero height: draw nothing
2428:
2429:       li $t0, 0X0000FFFF  # color: Cyan
2430:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2431:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2432:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2433:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2434:       sll $a1, $a1, 2     # Shift logical left by 2
2435:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2436:       sll $a3, $a3, 11    # SHift logical left by 11
2437:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2438:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2439:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2440:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2441:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2442:       li $t4, 0x800       # bytes per display row
2443:
```

```
2444:   rectangleCyanYloop:
2445:       move $t3, $a2    # pointer to current pixel for X loop; start at left edge
2446:
2447:   rectangleCyanXloop:
2448:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2449:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2450:       bne $t3, $t2, rectangleCyanXloop     # keep going if not past the right edge o
f the rectangle
2451:
2452:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2453:       addu $t2, $t2, $t4  # and right edge pointers
2454:       bne $a2, $a3, rectangleCyanYloop    # keep going if not off the bottom of the
rectangle
2455:
2456:   rectangleCyanReturn:
2457:       jr $ra      # Finish and return to main and continue executing
2458:       syscall         # Print out the output on screen
2459:
2460:
2461:   rectangleWhite:
2462:       # $a0 is xmin (i.e., left edge; must be within the display)
2463:       # $a1 is width (must be nonnegative and within the display)
2464:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2465:       # $a3 is height (must be nonnegative and within the display)
2466:
2467:       beq $a1, $zero, rectangleWhiteReturn    # zero width: draw nothing
2468:       beq $a3, $zero, rectangleWhiteReturn    # zero height: draw nothing
2469:
2470:       li $t0, 0X00FFFFFF  # color: White
2471:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2472:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2473:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2474:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2475:       sll $a1, $a1, 2     # Shift logical left by 2
2476:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2477:       sll $a3, $a3, 11    # SHift logical left by 11
2478:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2479:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2480:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2481:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2482:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2483:       li $t4, 0x800       # bytes per display row
2484:
2485:   rectangleWhiteYloop:
2486:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2487:
2488:   rectangleWhiteXloop:
2489:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2490:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2491:       bne $t3, $t2, rectangleWhiteXloop   # keep going if not past the right edge of
 the rectangle
2492:
```

```
2493:       addu $a2, $a2, $t4      # advace one row worth for the left edge
2494:       addu $t2, $t2, $t4      # and right edge pointers
2495:       bne $a2, $a3, rectangleWhiteYloop   # keep going if not off the bottom of the
rectangle
2496:
2497:   rectangleWhiteReturn:
2498:       jr $ra      # Finish and return to main and continue executing
2499:       syscall         # Print out the output on screen
2500:
2501:
2502:   Hline:
2503:       # $a0 is xmin (i.e., left edge; must be within the display)
2504:       # $a1 is width (must be nonnegative and within the display)
2505:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2506:       # $a3 is height (must be nonnegative and within the display)
2507:
2508:       beq $a1, $zero, HlineReturn     # zero width: draw nothing
2509:       beq $a3, $zero, HlineReturn     # zero height: draw nothing
2510:
2511:       li $t0, -1      # color: white
2512:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2513:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2514:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2515:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2516:       sll $a1, $a1, 2     # Shift logical left by 2
2517:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2518:       sll $a3, $a3, 11    # SHift logical left by 11
2519:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2520:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2521:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2522:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2523:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2524:       li $t4, 0x8000      # bytes per display row
2525:
2526:   HlineYloop:
2527:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2528:
2529:   HlineXloop:
2530:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2531:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2532:       bne $t3, $t2, HlineXloop    # keep going if not past the right edge of the rec
tangle
2533:
2534:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2535:       addu $t2, $t2, $t4   # and right edge pointers
2536:
2537:   HlineReturn:
2538:       jr $ra      # Finish and return to main and continue executing
2539:
2540:
2541:   Vline:
2542:       # $a0 is xmin (i.e., left edge; must be within the display)
```

```
2543:       # $a1 is width (must be nonnegative and within the display)
2544:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2545:       # $a3 is height (must be nonnegative and within the display)
2546:
2547:       beq $a1, $zero, VlineReturn # zero width: draw nothing
2548:       beq $a3, $zero, VlineReturn # zero height: draw nothing
2549:
2550:       li $t0, -1 # color: white
2551:       la $t1, frameBuffer # Load address held at frameBuffer to register $t1
2552:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2553:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2554:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2555:       sll $a1, $a1, 2     # Shift logical left by 2
2556:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2557:       sll $a3, $a3, 11    # SHift logical left by 11
2558:       addu $t2, $a2, $t1  # translate y values to display row starting addresses
2559:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2560:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2561:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2562:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2563:       li $t4, 0x800       # bytes per display row
2564:
2565:   VlineYloop:
2566:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2567:
2568:   VlineXloop:
2569:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2570:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2571:
2572:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2573:       addu $t2, $t2, $t4  # and right edge pointers
2574:       bne $a2, $a3, VlineYloop    # keep going if not off the bottom of the rectangl
e
2575:
2576:   VlineReturn:
2577:       jr $ra      # Finish and return to main and continue executing
2578:
2579:
2580:   HlineBlack:
2581:       # $a0 is xmin (i.e., left edge; must be within the display)
2582:       # $a1 is width (must be nonnegative and within the display)
2583:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2584:       # $a3 is height (must be nonnegative and within the display)
2585:
2586:       beq $a1, $zero, HlineBlackReturn    # zero width: draw nothing
2587:       beq $a3, $zero, HlineBlackReturn    # zero height: draw nothing
2588:
2589:       li $t0, 0x00000000  # color: Black
2590:       la $t1, frameBuffer
2591:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2592:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
```

```
2593:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2594:       sll $a1, $a1, 2     # Shift logical left by 2
2595:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2596:       sll $a3, $a3, 11    # SHift logical left by 11
2597:       addu $t2, $a2, $t1  # translate y values to display row starting addresses
2598:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2599:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2600:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2601:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2602:       li $t4, 0x8000      # bytes per display row
2603:
2604:   HlineBlackYloop:
2605:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2606:
2607:   HlineBlackXloop:
2608:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2609:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2610:       bne $t3, $t2, HlineBlackXloop   # keep going if not past the right edge of the
 rectangle
2611:
2612:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2613:       addu $t2, $t2, $t4  # and right edge pointers
2614:
2615:   HlineBlackReturn:
2616:       jr $ra      # Finish and return to main and continue executing
2617:
2618:
2619:   VlineBlack:
2620:       # $a0 is xmin (i.e., left edge; must be within the display)
2621:       # $a1 is width (must be nonnegative and within the display)
2622:       # $a2 is ymin  (i.e., top edge, increasing down; must be within the display)
2623:       # $a3 is height (must be nonnegative and within the display)
2624:
2625:       beq $a1, $zero, VlineBlackReturn    # zero width: draw nothing
2626:       beq $a3, $zero, VlineBlackReturn    # zero height: draw nothing
2627:
2628:       li $t0, 0x00000000  # color: Black
2629:       la $t1, frameBuffer  # Load address held at frameBuffer to register $t1
2630:       add $a1, $a1, $a0   # simplify loop tests by switching to first too-far value
2631:       add $a3, $a3, $a2   # add value held at register $a3 with value held at $a2 an
d store value at $a3
2632:       sll $a0, $a0, 2     # scale x values to bytes (4 bytes per pixel)
2633:       sll $a1, $a1, 2     # Shift logical left by 2
2634:       sll $a2, $a2, 11    # scale y values to bytes (512*4 bytes per display row)
2635:       sll $a3, $a3, 11    # SHift logical left by 11
2636:       addu $t2, $a2, $t1   # translate y values to display row starting addresses
2637:       addu $a3, $a3, $t1  # add value held at register $a3 with value held at $a1 an
d store value at $a3
2638:       addu $a2, $t2, $a0  # translate y values to rectangle row starting addresses
2639:       addu $a3, $a3, $a0  # add value held at register $a3 with value held at $a0 an
d store value at $a3
2640:       addu $t2, $t2, $a1  # and compute the ending address for first rectangle row
2641:       li $t4, 0x800       # bytes per display row
```

```
2642:
2643:   VlineBlackYloop:
2644:       move $t3, $a2       # pointer to current pixel for X loop; start at left edge
2645:
2646:   VlineBlackXloop:
2647:       sw $t0,($t3)        # Store value held at register $t3 into $t0
2648:       addiu $t3, $t3, 4   # Update value held at register $t4 by adding 4 to it
2649:
2650:       addu $a2, $a2, $t4  # advace one row worth for the left edge
2651:       addu $t2, $t2, $t4  # and right edge pointers
2652:       bne $a2, $a3, VlineBlackYloop       # keep going if not off the bottom of the
rectangle
2653:
2654:   VlineBlackReturn:
2655:       jr $ra      # Finish and return to main and continue executing
2656:
2657:
2658:   # Game Introduction function
2659:   printGameIntro:
2660:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2661:       la $a0, GameIntro1  # Load GameIntro1 into $a0 using la (load Address)
2662:       syscall     # Print out the output on screen
2663:
2664:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2665:       la $a0, GameIntro2  # Load GameIntro2 into $a0 using la (load Address)
2666:       syscall     # Print out the output on screen
2667:
2668:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2669:       la $a0, GameIntro3  # Load GameIntro3 into $a0 using la (load Address)
2670:       syscall     # Print out the output on screen
2671:
2672:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2673:       la $a0, GameIntro4  # Load GameIntro4 into $a0 using la (load Address)
2674:       syscall     # Print out the output on screen
2675:
2676:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2677:       la $a0, GameIntro5  # Load GameIntro5 into $a0 using la (load Address)
2678:       syscall     # Print out the output on screen
2679:
2680:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2681:       la $a0, GameIntro6  # Load GameIntro6 into $a0 using la (load Address)
2682:       syscall     # Print out the output on screen
2683:
2684:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2685:       la $a0, GameIntro7  # Load GameIntro7 into $a0 using la (load Address)
2686:       syscall     # Print out the output on screen
2687:
2688:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
```

```
ter $v0
2689:       la $a0, GameIntro8  # Load GameIntro8 into $a0 using la (load Address)
2690:       syscall     # Print out the output on screen
2691:
2692:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2693:       la $a0, GameIntro9  # Load GameIntro9 into $a0 using la (load Address)
2694:       syscall     # Print out the output on screen
2695:
2696:       jr $ra      # Finish and return to main and continue executing
2697:
2698:
2699:
2700:   # Gold Medal Prize
2701:   GoldMedal:
2702:       addi $sp, $sp, -8   # Alocatie memeory in the stack for 8 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
2703:       sw   $ra, 0($sp)    # storing the nested function "Medal" called in this funct
ion in the stack in the first location in the stack pointer at 0 using sw (store word)
2704:       sw   $ra, 4($sp)    # storing the nested function "Medals" called in this func
tion in the stack in the second location in the stack pointer at 4 using sw (store word)
2705:
2706:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2707:       la $a0, GoldMedal1.1    # Load GoldMedal1.1 into $a0 using la (load Address)
2708:       syscall     # Print out the output on screen
2709:
2710:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2711:       la $a0, GoldMedal1.2    # Load GoldMedal1.2 into $a0 using la (load Address)
2712:       syscall     # Print out the output on screen
2713:
2714:       jal Medal   # Jump to Medal Function and print the first part of the prize
2715:
2716:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2717:       la $a0, GoldMedal1.3    # Load GoldMedal1.3 into $a0 using la (load Address)
2718:       syscall     # Print out the output on screen
2719:
2720:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2721:       la $a0, GoldMedal1.4    # Load GoldMedal1.4 into $a0 using la (load Address)
2722:       syscall     # Print out the output on screen
2723:
2724:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2725:       la $a0, GoldMedal1.5    # Load GoldMedal1.5 into $a0 using la (load Address)
2726:       syscall     # Print out the output on screen
2727:
2728:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2729:       la $a0, GoldMedal1.6    # Load GoldMedal1.6 into $a0 using la (load Address)
2730:       syscall     # Print out the output on screen
2731:
```

```
2732:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2733:       la $a0, GoldMedal1.7    # Load GoldMedal1.7 into $a0 using la (load Address)
2734:       syscall     # Print out the output on screen
2735:
2736:       jal Medals  # Jump to Medals Function and print the second part of the prize
2737:
2738:       lw   $ra, 0($sp)    # Loading the nested function "Medal" called in this funct
ion from the stack in the first location in the stack pointer at 0 using lw (load word)
2739:       lw   $ra, 4($sp)    # Loading the nested function "Medals" called in this func
tion from the stack in the second location in the stack pointer at 4 using lw (load word)
2740:       addi $sp, $sp, 8    # Alocatie memeory back to the stack for 8 bytes (negative
 because we are allocating space from the stack, positive is when we are adding space to t
he stack)
2741:
2742:       jr $ra      # Finish and return to main and continue executing
2743:
2744:
2745:
2746:   # Silver Medal Prize
2747:   SilverMedal:
2748:       addi $sp, $sp, -8   # Alocatie memeory in the stack for 8 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
2749:       sw   $ra, 0($sp)    # storing the nested function "Medal" called in this funct
ion in the stack in the first location in the stack pointer at 0 using sw (store word)
2750:       sw   $ra, 4($sp)    # storing the nested function "Medals" called in this func
tion in the stack in the second location in the stack pointer at 4 using sw (store word)
2751:
2752:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2753:       la $a0, SilverMedal2.1  # Load SilverMedal2.1 into $a0 using la (load Address)
2754:       syscall     # Print out the output on screen
2755:
2756:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2757:       la $a0, SilverMedal2.2  # Load SilverMedal2.2 into $a0 using la (load Address)
2758:       syscall     # Print out the output on screen
2759:
2760:       jal Medal   # Jump to Medal Function and print the first part of the prize
2761:
2762:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2763:       la $a0, SilverMedal2.3  # Load SilverMedal2.3 into $a0 using la (load Address)
2764:       syscall     # Print out the output on screen
2765:
2766:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2767:       la $a0, SilverMedal2.4  # Load SilverMedal2.4 into $a0 using la (load Address)
2768:       syscall     # Print out the output on screen
2769:
2770:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2771:       la $a0, SilverMedal2.5  # Load SilverMedal2.5 into $a0 using la (load Address)
2772:       syscall     # Print out the output on screen
```

```
2773:
2774:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2775:       la $a0, SilverMedal2.6  # Load SilverMedal2.6 into $a0 using la (load Address)
2776:       syscall     # Print out the output on screen
2777:
2778:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2779:       la $a0, SilverMedal2.7  # Load SilverMedal2.7 into $a0 using la (load Address)
2780:       syscall     # Print out the output on screen
2781:
2782:       jal Medals  # Jump to Medals Function and print the second part of the prize
2783:
2784:       lw   $ra, 0($sp)    # Loading the nested function "Medal" called in this funct
ion from the stack in the first location in the stack pointer at 0 using lw (load word)
2785:       lw   $ra, 4($sp)    # Loading the nested function "Medals" called in this func
tion from the stack in the second location in the stack pointer at 4 using lw (load word)
2786:       addi $sp, $sp, 8    # Alocatie memeory back to the stack for 8 bytes (negative
 because we are allocating space from the stack, positive is when we are adding space to t
he stack)
2787:
2788:       jr $ra      # Finish and return to main and continue executing
2789:
2790:
2791:
2792:   # Bronze Medal Prize
2793:   BronzeMedal:
2794:       addi $sp, $sp, -8   # Alocatie memeory in the stack for 8 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
2795:       sw   $ra, 0($sp)    # storing the nested function "Medal" called in this funct
ion in the stack in the first location in the stack pointer at 0 using sw (store word)
2796:       sw   $ra, 4($sp)    # storing the nested function "Medals" called in this func
tion in the stack in the second location in the stack pointer at 4 using sw (store word)
2797:
2798:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2799:       la $a0, BronzeMedal3.1  # Load BronzeMedal3.1 into $a0 using la (load Address)
2800:       syscall     # Print out the output on screen
2801:
2802:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2803:       la $a0, BronzeMedal3.2  # Load BronzeMedal3.2 into $a0 using la (load Address)
2804:       syscall     # Print out the output on screen
2805:
2806:       jal Medal   # Jump to Medal Function and print the first part of the prize
2807:
2808:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2809:       la $a0, BronzeMedal3.3  # Load BronzeMedal3.3 into $a0 using la (load Address)
2810:       syscall     # Print out the output on screen
2811:
2812:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2813:       la $a0, BronzeMedal3.4  # Load BronzeMedal3.4 into $a0 using la (load Address)
```

```
2814:       syscall     # Print out the output on screen
2815:
2816:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2817:       la $a0, BronzeMedal3.5  # Load BronzeMedal3.5 into $a0 using la (load Address)
2818:       syscall     # Print out the output on screen
2819:
2820:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2821:       la $a0, BronzeMedal3.6  # Load BronzeMedal3.6 into $a0 using la (load Address)
2822:       syscall     # Print out the output on screen
2823:
2824:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2825:       la $a0, BronzeMedal3.7  # Load BronzeMedal3.7 into $a0 using la (load Address)
2826:       syscall     # Print out the output on screen
2827:
2828:       jal Medals  # Jump to Medals Function and print the second part of the prize
2829:
2830:       lw   $ra, 0($sp)    # Loading the nested function "Medal" called in this funct
ion from the stack in the first location in the stack pointer at 0 using lw (load word)
2831:       lw   $ra, 4($sp)    # Loading the nested function "Medals" called in this func
tion from the stack in the second location in the stack pointer at 4 using lw (load word)
2832:       addi $sp, $sp, 8    # Alocatie memeory back to the stack for 8 bytes (negative
 because we are allocating space from the stack, positive is when we are adding space to t
he stack)
2833:
2834:       jr $ra      # Finish and return to main and continue executing
2835:
2836:
2837:
2838:   # First part of prizes
2839:   Medal:
2840:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2841:       la $a0, Medal.1     # Load Medal.1 into $a0 using la (load Address)
2842:       syscall     # Print out the output on screen
2843:
2844:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2845:       la $a0, Medal.2     # Load Medal.2 into $a0 using la (load Address)
2846:       syscall     # Print out the output on screen
2847:
2848:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2849:       la $a0, Medal.3 # Load Medal.3 into $a0 using la (load Address)
2850:       syscall     # Print out the output on screen
2851:
2852:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2853:       la $a0, Medal.4     # Load Medal.4 into $a0 using la (load Address)
2854:       syscall     # Print out the output on screen
2855:
2856:       jr $ra      # Finish and return to main and continue executing
2857:
```

```
2858:
2859:
2860:   # Second part of prizes
2861:   Medals:
2862:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2863:       la $a0, Medals.1    # Load Medals.1 into $a0 using la (load Address)
2864:       syscall     # Print out the output on screen
2865:
2866:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2867:       la $a0, Medals.2    # Load Medals.2 into $a0 using la (load Address)
2868:       syscall     # Print out the output on screen
2869:
2870:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2871:       la $a0, Medals.3    # Load Medals.3 into $a0 using la (load Address)
2872:       syscall     # Print out the output on screen
2873:
2874:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2875:       la $a0, Medals.4    # Load Medals.4 into $a0 using la (load Address)
2876:       syscall     # Print out the output on screen
2877:
2878:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2879:       la $a0, Medals.5    # Load Medals.5 into $a0 using la (load Address)
2880:       syscall     # Print out the output on screen
2881:
2882:       jr $ra      # Finish and return to main and continue executing
2883:
2884:
2885:
2886:   # gameMaps1() function
2887:   gameMaps1:
2888:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2889:       la $a0, gameMaps1.1 # Load gameMaps1.1 into $a0 using la (load Address)
2890:       syscall     # Print out the output on screen
2891:
2892:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2893:       la $a0, gameMaps1.2 # Load gameMaps1.2 into $a0 using la (load Address)
2894:       syscall     # Print out the output on screen
2895:
2896:       jr $ra      # Finish and return to main and continue executing
2897:
2898:
2899:
2900:   # gameMaps2() function
2901:   gameMaps2:
2902:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2903:       la $a0, gameMaps2.1 # Load gameMaps2.1 into $a0 using la (load Address)
2904:       syscall     # Print out the output on screen
```

```
2905:
2906:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2907:       la $a0, gameMaps2.2 # Load gameMaps2.2 into $a0 using la (load Address)
2908:       syscall     # Print out the output on screen
2909:
2910:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2911:       la $a0, gameMaps2.3 # Load gameMaps2.3 into $a0 using la (load Address)
2912:       syscall     # Print out the output on screen
2913:
2914:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2915:       la $a0, gameMaps2.4 # Load gameMaps2.4 into $a0 using la (load Address)
2916:       syscall     # Print out the output on screen
2917:
2918:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2919:       la $a0, gameMaps2.5 # Load gameMaps2.5 into $a0 using la (load Address)
2920:       syscall     # Print out the output on screen
2921:
2922:       jr $ra      # Finish and return to main and continue executing
2923:
2924:
2925:
2926:   # gameMaps3() function
2927:   gameMaps3:
2928:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2929:       la $a0, gameMaps3.1 # Load gameMaps3.1 into $a0 using la (load Address)
2930:       syscall     # Print out the output on screen
2931:
2932:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2933:       la $a0, gameMaps3.2 # Load gameMaps3.1 into $a0 using la (load Address)
2934:       syscall     # Print out the output on screen
2935:
2936:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2937:       la $a0, gameMaps3.3 # Load gameMaps3.3 into $a0 using la (load Address)
2938:       syscall     # Print out the output on screen
2939:
2940:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2941:       la $a0, gameMaps3.4 # Load gameMaps3.4 into $a0 using la (load Address)
2942:       syscall     # Print out the output on screen
2943:
2944:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2945:       la $a0, gameMaps3.5 # Load gameMaps3.5 into $a0 using la (load Address)
2946:       syscall     # Print out the output on screen
2947:
2948:       jr $ra      # Finish and return to main and continue executing
2949:
2950:
```

```
2951:
2952:   # gameMaps() function
2953:   gameMap:
2954:       addi $sp, $sp, -8   # Alocatie memeory in the stack for 8 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
2955:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
2956:       sw   $ra, 4($sp)    # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 4 using sw (store word)
2957:       sw   $ra, 8($sp)    # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 8 using sw (store word
)
2958:
2959:       jal gameMaps1   # Printing first part of Game Map by calling child function ga
meMaps1
2960:
2961:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2962:       la $a0, gameMap.1   # Load gameMap.1 into $a0 using la (load Address)
2963:       syscall     # Print out the output on screen
2964:
2965:       jal gameMaps2   # Printing first part of Game Map by calling child function ga
meMaps2
2966:
2967:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2968:       la $a0, gameMap.2   # Load gameMap.2 into $a0 using la (load Address)
2969:       syscall     # Print out the output on screen
2970:
2971:       jal gameMaps3   # Printing first part of Game Map by calling child function ga
meMaps3
2972:
2973:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2974:       la $a0, gameMap.3   # Load gameMap.3 into $a0 using la (load Address)
2975:       syscall     # Print out the output on screen
2976:
2977:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
2978:       la $a0, gameMap.4   # Load gameMap.4 into $a0 using la (load Address)
2979:       syscall     # Print out the output on screen
2980:
2981:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
2982:       lw   $ra, 4($sp)    # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
2983:       lw   $ra, 8($sp)    # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
2984:       addi $sp, $sp, 8    # Restore/add space back to the stack by adding 8 to it (t
he 12 bytes that we allocated from the stack at the beginning of the fucntion)
2985:
2986:       jr $ra      # Finish and return to main and continue executing
2987:
2988:
```

```
2989:
2990:   # Map10Points function
2991:   Map10Points:
2992:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
2993:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
2994:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
2995:       # i: row index of the desired element
2996:       # m: number of columns in the array
2997:       # j: column index of the desired element
2998:       # element size: size of the element in bytes
2999:
3000:       # Initializing registers with values at desired element to be printed
3001:       # Printing value 10 at index MapPoints[0][0]
3002:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3003:       li $t1, 0       # row index of value 10 (i = 0)
3004:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3005:       li $t3, 0       # column index of value 10 (j = 0)
3006:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3007:
3008:       # Calculate the address of value 10 using the formula
3009:       mul $t5, $t1, $t2   # (i * m)
3010:       add $t5, $t5, $t3   # (i * m + j)
3011:       sll $t5, $t5, 2     # (i * m + j) * element_size
3012:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3013:
3014:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3015:
3016:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3017:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3018:       syscall         # Print out the output on screen
3019:
3020:       jr $ra      # Finish and return to main and continue executing
3021:
3022:
3023:
3024:   # Map20Points function
3025:   Map20Points:
3026:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
3027:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
3028:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
3029:       # i: row index of the desired element
3030:       # m: number of columns in the array
3031:       # j: column index of the desired element
3032:       # element size: size of the element in bytes
3033:
3034:       # Printing value 20 at index MapPoints[0][1]
```

```
3035:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3036:       li $t1, 0       # row index of value 10 (i = 0)
3037:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3038:       li $t3, 1       # column index of value 10 (j = 1)
3039:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3040:
3041:       # Calculate the address of value 10 using the formula
3042:       mul $t5, $t1, $t2   # (i * m)
3043:       add $t5, $t5, $t3   # (i * m + j)
3044:       sll $t5, $t5, 2     # (i * m + j) * element_size
3045:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3046:
3047:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3048:
3049:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3050:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3051:       syscall         # Print out the output on screen
3052:
3053:       jr $ra      # Finish and return to main and continue executing
3054:
3055:
3056:   # Map30Points function
3057:   Map30Points:
3058:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
3059:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
3060:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
3061:       # i: row index of the desired element
3062:       # m: number of columns in the array
3063:       # j: column index of the desired element
3064:       # element size: size of the element in bytes
3065:
3066:       # Printing value 30 at index MapPoints[0][2]
3067:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3068:       li $t1, 0       # row index of value 10 (i = 0)
3069:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3070:       li $t3, 2       # column index of value 10 (j = 2)
3071:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3072:
3073:       # Calculate the address of value 10 using the formula
3074:       mul $t5, $t1, $t2   # (i * m)
3075:       add $t5, $t5, $t3   # (i * m + j)
3076:       sll $t5, $t5, 2     # (i * m + j) * element_size
3077:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3078:
3079:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3080:
3081:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
```

```
3082:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3083:       syscall         # Print out the output on screen
3084:
3085:       jr $ra      # Finish and return to main and continue executing
3086:
3087:
3088:
3089:   # Map40Points function
3090:   Map40Points:
3091:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
3092:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
3093:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
3094:       # i: row index of the desired element
3095:       # m: number of columns in the array
3096:       # j: column index of the desired element
3097:       # element size: size of the element in bytes
3098:
3099:       # Printing value 40 at index MapPoints[1][0]
3100:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3101:       li $t1, 1       # row index of value 10 (i = 1)
3102:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3103:       li $t3, 0       # column index of value 10 (j = 0)
3104:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3105:
3106:       # Calculate the address of value 10 using the formula
3107:       mul $t5, $t1, $t2   # (i * m)
3108:       add $t5, $t5, $t3   # (i * m + j)
3109:       sll $t5, $t5, 2     # (i * m + j) * element_size
3110:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3111:
3112:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3113:
3114:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3115:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3116:       syscall         # Print out the output on screen
3117:
3118:       jr $ra      # Finish and return to main and continue executing
3119:
3120:
3121:
3122:   # Map50Points function
3123:   Map50Points:
3124:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
3125:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
3126:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
```

```
3127:       # i: row index of the desired element
3128:       # m: number of columns in the array
3129:       # j: column index of the desired element
3130:       # element size: size of the element in bytes
3131:
3132:       # Printing value 50 at index MapPoints[1][1]
3133:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3134:       li $t1, 1       # row index of value 10 (i = 1)
3135:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3136:       li $t3, 1       # column index of value 10 (j = 1)
3137:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3138:
3139:       # Calculate the address of value 10 using the formula
3140:       mul $t5, $t1, $t2   # (i * m)
3141:       add $t5, $t5, $t3   # (i * m + j)
3142:       sll $t5, $t5, 2     # (i * m + j) * element_size
3143:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3144:
3145:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3146:
3147:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3148:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3149:       syscall         # Print out the output on screen
3150:
3151:       jr $ra      # Finish and return to main and continue executing
3152:
3153:
3154:
3155:   # Map60Points function
3156:   Map60Points:
3157:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
3158:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
3159:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
3160:       # i: row index of the desired element
3161:       # m: number of columns in the array
3162:       # j: column index of the desired element
3163:       # element size: size of the element in bytes
3164:
3165:       # Printing value 60 at index MapPoints[1][2]
3166:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3167:       li $t1, 1       # row index of value 10 (i = 1)
3168:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3169:       li $t3, 2       # column index of value 10 (j = 2)
3170:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3171:
3172:       # Calculate the address of value 10 using the formula
3173:       mul $t5, $t1, $t2   # (i * m)
3174:       add $t5, $t5, $t3   # (i * m + j)
```

```
3175:       sll $t5, $t5, 2     # (i * m + j) * element_size
3176:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3177:
3178:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3179:
3180:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3181:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3182:       syscall         # Print out the output on screen
3183:
3184:       jr $ra      # Finish and return to main and continue executing
3185:
3186:
3187:
3188:   # Map100Points function
3189:   Map100Points:
3190:       # CALLING/PRINTING AN INTEGER FROM 2D ARRAY (Game Map Points (MapPoints[3][3])
)
3191:       # First use ---> element_address = base_address + (i * m + j) * element_size <
--- to calculate the element address of the printed/desired element
3192:       # base_address: starting address of the element (base address of a two-dimensi
onal array or an array is a reference starting point)
3193:       # i: row index of the desired element
3194:       # m: number of columns in the array
3195:       # j: column index of the desired element
3196:       # element size: size of the element in bytes
3197:
3198:       # Printing value 100 at index MapPoints[2][0]
3199:       la $t0, MapPoints   # base address of array MapPoints (base_address = addess o
f MapPoints)
3200:       li $t1, 2       # row index of value 10 (i = 2)
3201:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3202:       li $t3, 0       # column index of value 10 (j = 0)
3203:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3204:
3205:       # Calculate the address of value 10 using the formula
3206:       mul $t5, $t1, $t2   # (i * m)
3207:       add $t5, $t5, $t3   # (i * m + j)
3208:       sll $t5, $t5, 2     # (i * m + j) * element_size
3209:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3210:
3211:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3212:
3213:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3214:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3215:       syscall         # Print out the output on screen
3216:
3217:       jr $ra      # Finish and return to main and continue executing
3218:
3219:
3220:
3221:   # gameMap1(10 Points) function
```

```
3222:   gameMap1:
3223:       addi $sp, $sp, -12  # Alocatie memeory in the stack for 12 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
3224:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3225:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3226:       sw   $ra, 8($sp)    # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 8 using sw (store word)
3227:       sw   $ra, 12($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 12 using sw (store wor
d)
3228:
3229:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3230:
3231:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3232:       la $a0, gameMap1.1  # Load gameMap1.1 into $a0 using la (load Address)
3233:       syscall     # Print out the output on screen
3234:
3235:       jal Map10Points  # Printing 10 points for the question 1
3236:
3237:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3238:       la $a0, gameMap1.2  # Load gameMap1.2 into $a0 using la (load Address)
3239:       syscall     # Print out the output on screen
3240:
3241:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
3242:
3243:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3244:       la $a0, gameMap1.3  # Load gameMap1.3 into $a0 using la (load Address)
3245:       syscall     # Print out the output on screen
3246:
3247:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
3248:
3249:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3250:       la $a0, gameMap1.4  # Load gameMap1.4 into $a0 using la (load Address)
3251:       syscall     # Print out the output on screen
3252:
3253:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3254:       la $a0, gameMap1.5  # Load gameMap1.5 into $a0 using la (load Address)
3255:       syscall     # Print out the output on screen
3256:
3257:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
3258:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
```

```
3259:       lw   $ra, 8($sp)    # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
3260:       lw   $ra, 12($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3261:       addi $sp, $sp, 12   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3262:
3263:       jr $ra      # Finish and return to main and continue executing
3264:
3265:
3266:
3267:   # gameMap2(10 Points, 20 points) function
3268:   gameMap2:
3269:       addi $sp, $sp, -16  # Alocatie memeory in the stack for 16 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
3270:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3271:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3272:       sw   $ra, 8($sp)    # storing the nested function "Map20Points" called in this
 function in the stack in the second location in the stack pointer at 8 using sw (store wo
rd)
3273:       sw   $ra, 12($sp)   # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 12 using sw (store word
)
3274:       sw   $ra, 16($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 16 using sw (store wor
d)
3275:
3276:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3277:
3278:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3279:       la $a0, gameMap2.1  # Load gameMap2.1 into $a0 using la (load Address)
3280:       syscall     # Print out the output on screen
3281:
3282:       jal Map10Points  # Printing 10 points for the question 1
3283:
3284:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3285:       la $a0, gameMap2.2  # Load gameMap2.2 into $a0 using la (load Address)
3286:       syscall     # Print out the output on screen
3287:
3288:       jal Map20Points  # Printing 20 points for the question 2
3289:
3290:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3291:       la $a0, gameMap2.3  # Load gameMap2.3 into $a0 using la (load Address)
3292:       syscall     # Print out the output on screen
3293:
3294:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
```

```
3295:
3296:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3297:       la $a0, gameMap2.4  # Load gameMap2.4 into $a0 using la (load Address)
3298:       syscall     # Print out the output on screen
3299:
3300:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
3301:
3302:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3303:       la $a0, gameMap2.5  # Load gameMap2.5 into $a0 using la (load Address)
3304:       syscall     # Print out the output on screen
3305:
3306:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3307:       la $a0, gameMap2.6  # Load gameMap2.6 into $a0 using la (load Address)
3308:       syscall     # Print out the output on screen
3309:
3310:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
3311:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
3312:       lw   $ra, 8($sp)    # Loading/restore the nested function "Map20Points" called
 in this function, in the stack back to register $ra using lw (load word)
3313:       lw   $ra, 12($sp)   # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
3314:       lw   $ra, 16($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3315:       addi $sp, $sp, 16   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3316:
3317:       jr $ra      # Finish and return to main and continue executing
3318:
3319:
3320:
3321:   # gameMap3(10 points, 20 points, 30 points)
3322:   gameMap3:
3323:       addi $sp, $sp, -20  # Alocatie memeory in the stack for 20 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
3324:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3325:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3326:       sw   $ra, 8($sp)    # storing the nested function "Map20Points" called in this
 function in the stack in the second location in the stack pointer at 8 using sw (store wo
rd)
3327:       sw   $ra, 12($sp)   # storing the nested function "Map30Points" called in this
 function in the stack in the second location in the stack pointer at 12 using sw (store w
ord)
3328:       sw   $ra, 16($sp)   # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 16 using sw (store word
)
```

```
3329:       sw   $ra, 20($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 20 using sw (store wor
d)
3330:
3331:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3332:
3333:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3334:       la $a0, gameMap3.1  # Load gameMap3.1 into $a0 using la (load Address)
3335:       syscall     # Print out the output on screen
3336:
3337:       jal Map10Points  # Printing 10 points for the question 1
3338:
3339:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3340:       la $a0, gameMap3.2  # Load gameMap3.2 into $a0 using la (load Address)
3341:       syscall     # Print out the output on screen
3342:
3343:       jal Map20Points  # Printing 20 points for the question 2
3344:
3345:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3346:       la $a0, gameMap3.3  # Load gameMap3.3 into $a0 using la (load Address)
3347:       syscall     # Print out the output on screen
3348:
3349:       jal Map30Points  # Printing 30 points for the question 3
3350:
3351:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3352:       la $a0, gameMap3.4  # Load gameMap3.4 into $a0 using la (load Address)
3353:       syscall     # Print out the output on screen
3354:
3355:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
3356:
3357:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3358:       la $a0, gameMap3.5  # Load gameMap3.5 into $a0 using la (load Address)
3359:       syscall     # Print out the output on screen
3360:
3361:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
3362:
3363:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3364:       la $a0, gameMap3.6  # Load gameMap3.6 into $a0 using la (load Address)
3365:       syscall     # Print out the output on screen
3366:
3367:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3368:       la $a0, gameMap3.7  # Load gameMap3.7 into $a0 using la (load Address)
3369:       syscall     # Print out the output on screen
3370:
3371:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
```

```
n this function, in the stack back to register $ra using lw (load word)
3372:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
3373:       lw   $ra, 8($sp)    # Loading/restore the nested function "Map20Points" called
 in this function, in the stack back to register $ra using lw (load word)
3374:       lw   $ra, 12($sp)   # Loading/restore the nested function "Map30Points" called
 in this function, in the stack back to register $ra using lw (load word)
3375:       lw   $ra, 16($sp)   # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
3376:       lw   $ra, 20($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3377:       addi $sp, $sp, 20   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3378:
3379:       jr $ra      # Finish and return to main and continue executing
3380:
3381:
3382:
3383:   # gameMap4(10 points, 20 points, 30 points, 40 points)
3384:   gameMap4:
3385:       addi $sp, $sp, -24  # Alocatie memeory in the stack for 24 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
3386:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3387:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3388:       sw   $ra, 8($sp)    # storing the nested function "Map20Points" called in this
 function in the stack in the second location in the stack pointer at 8 using sw (store wo
rd)
3389:       sw   $ra, 12($sp)   # storing the nested function "Map30Points" called in this
 function in the stack in the second location in the stack pointer at 12 using sw (store w
ord)
3390:       sw   $ra, 16($sp)   # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 16 using sw (store word
)
3391:       sw   $ra, 20($sp)   # storing the nested function "Map40Points" called in this
 function in the stack in the second location in the stack pointer at 20 using sw (store w
ord)
3392:       sw   $ra, 24($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 24 using sw (store wor
d)
3393:
3394:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3395:
3396:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3397:       la $a0, gameMap4.1  # Load gameMap4.1 into $a0 using la (load Address)
3398:       syscall     # Print out the output on screen
3399:
3400:       jal Map10Points  # Printing 10 points for the question 1
3401:
3402:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
```

```
ter $v0
3403:       la $a0, gameMap4.2  # Load gameMap4.2 into $a0 using la (load Address)
3404:       syscall     # Print out the output on screen
3405:
3406:       jal Map20Points  # Printing 20 points for the question 2
3407:
3408:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3409:       la $a0, gameMap4.3  # Load gameMap4.3 into $a0 using la (load Address)
3410:       syscall     # Print out the output on screen
3411:
3412:       jal Map30Points  # Printing 30 points for the question 3
3413:
3414:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3415:       la $a0, gameMap4.4  # Load gameMap4.4 into $a0 using la (load Address)
3416:       syscall     # Print out the output on screen
3417:
3418:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
3419:
3420:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3421:       la $a0, gameMap4.5  # Load gameMap4.5 into $a0 using la (load Address)
3422:       syscall     # Print out the output on screen
3423:
3424:       jal Map40Points  # Printing 40 points for the question 4
3425:
3426:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3427:       la $a0, gameMap4.6  # Load gameMap4.6 into $a0 using la (load Address)
3428:       syscall     # Print out the output on screen
3429:
3430:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
3431:
3432:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3433:       la $a0, gameMap4.7  # Load gameMap4.7 into $a0 using la (load Address)
3434:       syscall     # Print out the output on screen
3435:
3436:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3437:       la $a0, gameMap4.8  # Load gameMap4.8 into $a0 using la (load Address)
3438:       syscall     # Print out the output on screen
3439:
3440:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
3441:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
3442:       lw   $ra, 8($sp)    # Loading/restore the nested function "Map20Points" called
 in this function, in the stack back to register $ra using lw (load word)
3443:       lw   $ra, 12($sp)   # Loading/restore the nested function "Map30Points" called
 in this function, in the stack back to register $ra using lw (load word)
3444:       lw   $ra, 16($sp)   # Loading/restore the nested function "gameMaps2" called i
```

```
n this function, in the stack back to register $ra using lw (load word)
3445:       lw   $ra, 20($sp)   # Loading/restore the nested function "Map40Points" called
 in this function, in the stack back to register $ra using lw (load word)
3446:       lw   $ra, 24($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3447:       addi $sp, $sp, 24   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3448:
3449:       jr $ra      # Finish and return to main and continue executing
3450:
3451:
3452:
3453:         # gameMap5(10 points, 20 points, 30 points, 40 points, 50 points)
3454:         gameMap5:
3455:       addi $sp, $sp, -28  # Alocatie memeory in the stack for 28 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
3456:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3457:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3458:       sw   $ra, 8($sp)    # storing the nested function "Map20Points" called in this
 function in the stack in the second location in the stack pointer at 8 using sw (store wo
rd)
3459:       sw   $ra, 12($sp)   # storing the nested function "Map30Points" called in this
 function in the stack in the second location in the stack pointer at 12 using sw (store w
ord)
3460:       sw   $ra, 16($sp)   # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 16 using sw (store word
)
3461:       sw   $ra, 20($sp)   # storing the nested function "Map50Points" called in this
 function in the stack in the second location in the stack pointer at 20 using sw (store w
ord)
3462:       sw   $ra, 24($sp)   # storing the nested function "Map40Points" called in this
 function in the stack in the second location in the stack pointer at 24 using sw (store w
ord)
3463:       sw   $ra, 28($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 28 using sw (store wor
d)
3464:
3465:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3466:
3467:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3468:       la $a0, gameMap5.1  # Load gameMap5.1 into $a0 using la (load Address)
3469:       syscall     # Print out the output on screen
3470:
3471:       jal Map10Points  # Printing 10 points for the question 1
3472:
3473:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3474:       la $a0, gameMap5.2  # Load gameMap5.2 into $a0 using la (load Address)
3475:       syscall     # Print out the output on screen
```

```
3476:
3477:       jal Map20Points  # Printing 20 points for the question 2
3478:
3479:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3480:       la $a0, gameMap5.3  # Load gameMap5.3 into $a0 using la (load Address)
3481:       syscall     # Print out the output on screen
3482:
3483:       jal Map30Points  # Printing 30 points for the question 3
3484:
3485:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3486:       la $a0, gameMap5.4  # Load gameMap5.4 into $a0 using la (load Address)
3487:       syscall     # Print out the output on screen
3488:
3489:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
3490:
3491:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3492:       la $a0, gameMap5.5  # Load gameMap5.5 into $a0 using la (load Address)
3493:       syscall     # Print out the output on screen
3494:
3495:       jal Map50Points  # Printing 50 points for the question 5
3496:
3497:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3498:       la $a0, gameMap5.6  # Load gameMap5.6 into $a0 using la (load Address)
3499:       syscall     # Print out the output on screen
3500:
3501:       jal Map40Points  # Printing 40 points for the question 4
3502:
3503:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3504:       la $a0, gameMap5.7  # Load gameMap5.7 into $a0 using la (load Address)
3505:       syscall     # Print out the output on screen
3506:
3507:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
3508:
3509:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3510:       la $a0, gameMap5.8  # Load gameMap5.8 into $a0 using la (load Address)
3511:       syscall     # Print out the output on screen
3512:
3513:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3514:       la $a0, gameMap5.9  # Load gameMap5.9 into $a0 using la (load Address)
3515:       syscall     # Print out the output on screen
3516:
3517:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
3518:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
3519:       lw   $ra, 8($sp)    # Loading/restore the nested function "Map20Points" called
```

```
 in this function, in the stack back to register $ra using lw (load word)
3520:       lw   $ra, 12($sp)   # Loading/restore the nested function "Map30Points" called
 in this function, in the stack back to register $ra using lw (load word)
3521:       lw   $ra, 16($sp)   # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
3522:       lw   $ra, 20($sp)   # Loading/restore the nested function "Map50Points" called
 in this function, in the stack back to register $ra using lw (load word)
3523:       lw   $ra, 24($sp)   # Loading/restore the nested function "Map40Points" called
 in this function, in the stack back to register $ra using lw (load word)
3524:       lw   $ra, 28($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3525:       addi $sp, $sp, 28   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3526:
3527:       jr $ra      # Finish and return to main and continue executing
3528:
3529:
3530:
3531:         # gameMap6(10 points, 20 points, 30 points, 40 points, 50 points, 60 points)
3532:         gameMap6:
3533:           addi $sp, $sp, -32  # Alocatie memeory in the stack for 32 bytes (negative
 because we are allocating space from the stack, positive is when we are adding space to t
he stack)
3534:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3535:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3536:       sw   $ra, 8($sp)    # storing the nested function "Map20Points" called in this
 function in the stack in the second location in the stack pointer at 8 using sw (store wo
rd)
3537:       sw   $ra, 12($sp)   # storing the nested function "Map30Points" called in this
 function in the stack in the second location in the stack pointer at 12 using sw (store w
ord)
3538:       sw   $ra, 16($sp)   # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 16 using sw (store word
)
3539:       sw   $ra, 20($sp)   # storing the nested function "Map60Points" called in this
 function in the stack in the second location in the stack pointer at 20 using sw (store w
ord)
3540:       sw   $ra, 24($sp)   # storing the nested function "Map50Points" called in this
 function in the stack in the second location in the stack pointer at 24 using sw (store w
ord)
3541:       sw   $ra, 28($sp)   # storing the nested function "Map40Points" called in this
 function in the stack in the second location in the stack pointer at 28 using sw (store w
ord)
3542:       sw   $ra, 32($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 32 using sw (store wor
d)
3543:
3544:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3545:
3546:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
```

```
3547:       la $a0, gameMap6.1  # Load gameMap6.1 into $a0 using la (load Address)
3548:       syscall     # Print out the output on screen
3549:
3550:       jal Map10Points  # Printing 10 points for the question 1
3551:
3552:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3553:       la $a0, gameMap6.2  # Load gameMap6.2 into $a0 using la (load Address)
3554:       syscall     # Print out the output on screen
3555:
3556:       jal Map20Points  # Printing 20 points for the question 2
3557:
3558:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3559:       la $a0, gameMap6.3  # Load gameMap6.3 into $a0 using la (load Address)
3560:       syscall     # Print out the output on screen
3561:
3562:       jal Map30Points  # Printing 30 points for the question 3
3563:
3564:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3565:       la $a0, gameMap6.4  # Load gameMap6.4 into $a0 using la (load Address)
3566:       syscall     # Print out the output on screen
3567:
3568:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
3569:
3570:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3571:       la $a0, gameMap6.5  # Load gameMap6.5 into $a0 using la (load Address)
3572:       syscall     # Print out the output on screen
3573:
3574:       jal Map60Points  # Printing 60 points for the question 6
3575:
3576:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3577:       la $a0, gameMap6.6  # Load gameMap6.6 into $a0 using la (load Address)
3578:       syscall     # Print out the output on screen
3579:
3580:       jal Map50Points  # Printing 50 points for the question 5
3581:
3582:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3583:       la $a0, gameMap6.7  # Load gameMap6.7 into $a0 using la (load Address)
3584:       syscall     # Print out the output on screen
3585:
3586:       jal Map40Points  # Printing 40 points for the question 4
3587:
3588:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3589:       la $a0, gameMap6.8  # Load gameMap6.8 into $a0 using la (load Address)
3590:       syscall     # Print out the output on screen
3591:
3592:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
```

```
3593:
3594:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3595:       la $a0, gameMap6.9  # Load gameMap5.9 into $a0 using la (load Address)
3596:       syscall     # Print out the output on screen
3597:
3598:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3599:       la $a0, gameMap6.10 # Load gameMap6.10 into $a0 using la (load Address)
3600:       syscall     # Print out the output on screen
3601:
3602:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
3603:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
3604:       lw   $ra, 8($sp)    # Loading/restore the nested function "Map20Points" called
 in this function, in the stack back to register $ra using lw (load word)
3605:       lw   $ra, 12($sp)   # Loading/restore the nested function "Map30Points" called
 in this function, in the stack back to register $ra using lw (load word)
3606:       lw   $ra, 16($sp)   # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
3607:       lw   $ra, 20($sp)   # Loading/restore the nested function "Map60Points" called
 in this function, in the stack back to register $ra using lw (load word)
3608:       lw   $ra, 24($sp)   # Loading/restore the nested function "Map50Points" called
 in this function, in the stack back to register $ra using lw (load word)
3609:       lw   $ra, 28($sp)   # Loading/restore the nested function "Map40Points" called
 in this function, in the stack back to register $ra using lw (load word)
3610:       lw   $ra, 32($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3611:       addi $sp, $sp, 32   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3612:
3613:       jr $ra      # Finish and return to main and continue executing
3614:
3615:
3616:
3617:         # gameMap7(10 points, 20 points, 30 points, 40 points, 50 points, 60 points,
 100 points)
3618:         gameMap7:
3619:           addi $sp, $sp, -36  # Alocatie memeory in the stack for 36 bytes (negative
 because we are allocating space from the stack, positive is when we are adding space to t
he stack)
3620:       sw   $ra, 0($sp)    # storing the nested function "gameMaps1" called in this f
unction in the stack in the first location in the stack pointer at 0 using sw (store word)
3621:       sw   $ra, 4($sp)    # storing the nested function "Map10Points" called in this
 function in the stack in the second location in the stack pointer at 4 using sw (store wo
rd)
3622:       sw   $ra, 8($sp)    # storing the nested function "Map20Points" called in this
 function in the stack in the second location in the stack pointer at 8 using sw (store wo
rd)
3623:       sw   $ra, 12($sp)   # storing the nested function "Map30Points" called in this
 function in the stack in the second location in the stack pointer at 12 using sw (store w
ord)
3624:       sw   $ra, 16($sp)   # storing the nested function "gameMaps2" called in this f
unction in the stack in the third location in the stack pointer at 16 using sw (store word
```

```
)
3625:       sw   $ra, 20($sp)   # storing the nested function "Map60Points" called in this
 function in the stack in the second location in the stack pointer at 20 using sw (store w
ord)
3626:       sw   $ra, 24($sp)   # storing the nested function "Map50Points" called in this
 function in the stack in the second location in the stack pointer at 24 using sw (store w
ord)
3627:       sw   $ra, 28($sp)   # storing the nested function "Map40Points" called in this
 function in the stack in the second location in the stack pointer at 28 using sw (store w
ord)
3628:       sw   $ra, 32($sp)   # storing the nested function "gameMaps3" called in this f
unction in the stack in the fourth location in the stack pointer at 32 using sw (store wor
d)
3629:       sw   $ra, 36($sp)   # storing the nested function "Map100Points" called in thi
s function in the stack in the second location in the stack pointer at 36 using sw (store
word)
3630:
3631:       jal gameMaps1   # Printing first part of Game Maps1 by calling child function
gameMaps1
3632:
3633:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3634:       la $a0, gameMap7.1  # Load gameMap7.1 into $a0 using la (load Address)
3635:       syscall     # Print out the output on screen
3636:
3637:       jal Map10Points  # Printing 10 points for the question 1
3638:
3639:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3640:       la $a0, gameMap7.2  # Load gameMap7.2 into $a0 using la (load Address)
3641:       syscall     # Print out the output on screen
3642:
3643:       jal Map20Points  # Printing 20 points for the question 2
3644:
3645:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3646:       la $a0, gameMap7.3  # Load gameMap7.3 into $a0 using la (load Address)
3647:       syscall     # Print out the output on screen
3648:
3649:       jal Map30Points  # Printing 30 points for the question 3
3650:
3651:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3652:       la $a0, gameMap7.4  # Load gameMap7.4 into $a0 using la (load Address)
3653:       syscall     # Print out the output on screen
3654:
3655:       jal gameMaps2   # Printing second part of Game Map by calling child function g
ameMaps2
3656:
3657:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3658:       la $a0, gameMap7.5  # Load gameMap7.5 into $a0 using la (load Address)
3659:       syscall     # Print out the output on screen
3660:
3661:       jal Map60Points  # Printing 60 points for the question 6
```

```
3662:
3663:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3664:       la $a0, gameMap7.6  # Load gameMap7.6 into $a0 using la (load Address)
3665:       syscall     # Print out the output on screen
3666:
3667:       jal Map50Points  # Printing 50 points for the question 5
3668:
3669:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3670:       la $a0, gameMap7.7  # Load gameMap7.7 into $a0 using la (load Address)
3671:       syscall     # Print out the output on screen
3672:
3673:       jal Map40Points  # Printing 40 points for the question 4
3674:
3675:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3676:       la $a0, gameMap7.8  # Load gameMap7.8 into $a0 using la (load Address)
3677:       syscall     # Print out the output on screen
3678:
3679:       jal gameMaps3   # Printing third part of Game Map by calling child function ga
meMaps3
3680:
3681:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3682:       la $a0, gameMap7.9  # Load gameMap7.9 into $a0 using la (load Address)
3683:       syscall     # Print out the output on screen
3684:
3685:       jal Map100Points     # Printing 100 points for the question 7
3686:
3687:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3688:       la $a0, gameMap7.10 # Load gameMap7.10 into $a0 using la (load Address)
3689:       syscall     # Print out the output on screen
3690:
3691:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3692:       la $a0, gameMap7.11 # Load gameMap7.11 into $a0 using la (load Address)
3693:       syscall     # Print out the output on screen
3694:
3695:       lw   $ra, 0($sp)    # Loading/restore the nested function "gameMaps1" called i
n this function, in the stack back to register $ra using lw (load word)
3696:       lw   $ra, 4($sp)    # Loading/restore the nested function "Map10Points" called
 in this function, in the stack back to register $ra using lw (load word)
3697:       lw   $ra, 8($sp)    # Loading/restore the nested function "Map20Points" called
 in this function, in the stack back to register $ra using lw (load word)
3698:       lw   $ra, 12($sp)   # Loading/restore the nested function "Map30Points" called
 in this function, in the stack back to register $ra using lw (load word)
3699:       lw   $ra, 16($sp)   # Loading/restore the nested function "gameMaps2" called i
n this function, in the stack back to register $ra using lw (load word)
3700:       lw   $ra, 20($sp)   # Loading/restore the nested function "Map60Points" called
 in this function, in the stack back to register $ra using lw (load word)
3701:       lw   $ra, 24($sp)   # Loading/restore the nested function "Map50Points" called
 in this function, in the stack back to register $ra using lw (load word)
3702:       lw   $ra, 28($sp)   # Loading/restore the nested function "Map40Points" called
```

```
 in this function, in the stack back to register $ra using lw (load word)
3703:       lw   $ra, 32($sp)   # Loading/restore the nested function "gameMaps3" called i
n this function, in the stack back to register $ra using lw (load word)
3704:       lw   $ra, 36($sp)   # Loading/restore the nested function "Map100Points" calle
d in this function, in the stack back to register $ra using lw (load word)
3705:       addi $sp, $sp, 36   # Restore/add space back to the stack by adding 12 to it (
the 12 bytes that we allocated from the stack at the beginning of the fucntion)
3706:
3707:       jr $ra      # Finish and return to main and continue executing
3708:
3709:
3710:   # Pseudocode:
3711:   # void PolynomialQuestionOne(const int (*ArrayQ1)[COLS]){
3712:   #   print(Question 1)
3713:   #   while(true){
3714:   #   continue;
3715:   # }
3716:   #   while(false){
3717:   #   exit(0);
3718:   # }
3719:   # Question 1 function
3720:   PolynomialQuestionOne:
3721:       addi $sp, $sp, -32  # Alocatie memeory in the stack for 32 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
3722:       sw   $s3, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
3723:       sw   $ra, 4($sp)    # storing the nested function "FindQ1Num1" called in this
function in the stack in the second location in the stack pointer at 4 using sw (store wor
d)
3724:       sw   $ra, 8($sp)    # storing the nested function "FindQ1Num2" called in this
function in the stack in the third location in the stack pointer at 8 using sw (store word
)
3725:       sw   $ra, 12($sp)   # storing the nested function "FindQ1Num3" called in this
function in the stack in the fourth location in the stack pointer at 12 using sw (store wo
rd)
3726:       sw   $ra, 16($sp)   # storing the nested function "FindQ1Num3" called in this
function in the stack in the fifth location in the stack pointer at 16 using sw (store wor
d)
3727:       sw   $ra, 20($sp)   # storing the nested function "FindQ1Num1" called in this
function in the stack in the sisxth location in the stack pointer at 20 using sw (store wo
rd)
3728:       sw   $ra, 24($sp)   # storing the nested function "FindQ1Num2" called in this
function in the stack in the seventh location in the stack pointer at 24 using sw (store w
ord)
3729:       sw   $ra, 28($sp)   # storing the nested function "FindQ1Num3" called in this
function in the stack in the eighth location in the stack pointer at 28 using sw (store wo
rd)
3730:
3731:       # Printing Question 1 to the player
3732:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
3733:       la $a0, Question1       # Load Question1 into $a0 using la (load Address)
3734:       syscall     # Print out the output on screen
3735:
```

```
3736:       addi $t3, $zero, 2  # Initialize register $t3 with value 2 using addi
3737:       div $s3, $t3        # Divide value held in register $s3 with $t3
3738:
3739:       mfhi $t4        # Load the remainder of the division to register $t4
3740:
3741:       beq $t4, $zero, firstVersion        # Branch if equal, if value held in regist
er $t4 is equal to value in $zero go to firstVersion
3742:
3743:       j secondVersion     # Else go to secondVersion
3744:
3745:   firstVersion:
3746:       jal FindQ1Num1      # Jump to FindQ1Num1 function and print the first number o
f the question
3747:
3748:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
3749:       la $a0, firstVer1    # Load firstVer1 into $a0 using la (load Address)
3750:       syscall     # Print out the output on screen
3751:
3752:       jal FindQ1Num2      # Jump to FindQ1Num2 function and print the second number
of the question
3753:
3754:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
3755:       la $a0, firstVer2    # Load firstVer2 into $a0 using la (load Address)
3756:       syscall     # Print out the output on screen
3757:
3758:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
3759:       la $a0, roots       # Load roots into $a0 using la (load Address)
3760:       syscall     # Print out the output on screen
3761:
3762:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
3763:       syscall     # Print out the output on screen
3764:       move $s1, $v0   # move value held in $v0 to register $s1
3765:
3766:       jal FindQ1Num3      # Jump to FindQ1Num3 function and perform
3767:       lw $s2, valueX_Q1   # Load value held in valueX_Q1 to register $s2
3768:
3769:       bne $s1, $s2, incorrect_Q1  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s2 go to incorrect_Q1
3770:
3771:       j Correct_Q1        # Else go to Correct_Q1
3772:
3773:   Correct_Q1:
3774:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3775:       la $a0, correct   # Load correct into $a0 using la (load Address)
3776:       syscall   # Print out the output on screen
3777:
3778:       la $a0, sound5      # Load the address of "sound5" into $a0
3779:       la $a1, duration5   # Load the address of "duration5" into $a0
3780:       la $a2, instrument5     # Load the address of "instrument5" into $a0
3781:       la $a3, volume5     # Load the address of "volume5" into $a0
```

```
3782:
3783:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
3784:       lb $a1, 0($a1)      # Load the value of "duration5" into $a0
3785:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a0
3786:       lb $a3, 0($a3)      # Load the value of "volume5" into $a0
3787:
3788:       # Make the system call to play the sound
3789:           li $v0, 31   # Use the "play note" system call
3790:           syscall     # Print out the output on screen
3791:
3792:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
3793:       li $a0, 300 # wait for 300 millisecond
3794:       syscall     # Print out the output on screen
3795:
3796:       la $a0, sound5      # Load the address of "sound5" into $a0
3797:       la $a1, duration5   # Load the address of "duration5" into $a1
3798:       la $a2, instrument5     # Load the address of "instrument5" into $a2
3799:       la $a3, volume5     # Load the address of "volume5" into $a3
3800:
3801:       lb $a0, 0($a0)      # Load the value of "sound5" into $a1
3802:       lb $a1, 0($a1)      # Load the value of "duration5" into $a2
3803:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a3
3804:       lb $a3, 0($a3)      # Load the value of "volume5" into $a4
3805:
3806:       # Make the system call to play the sound
3807:           li $v0, 31   # Use the "play note" system call
3808:           syscall     # Print out the output on screen
3809:
3810:       j exit_Q1   # Jump to exit
3811:
3812:   incorrect_Q1:
3813:       la $a0, beep6       # Load the address of "beep6" into $a0
3814:       la $a1, duration6   # Load the address of "duration6" into $a1
3815:       la $a2, instrument6 # Load the address of "instrument6" into $a2
3816:       la $a3, volume6     # Load the address of "volume6" into $a3
3817:
3818:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
3819:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
3820:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
3821:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
3822:
3823:       # Make the system call to play the sound
3824:           li $v0, 31  # Use the "play note" system call
3825:           syscall     # Print out the output on screen
3826:
3827:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3828:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
3829:       syscall   # Print out the output on screen
3830:
3831:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
3832:       syscall     # Print out the output on screen
3833:
3834:       move $s1, $v0   # move value held in $v0 to register $s1
```

```
3835:
3836:       jal FindQ1Num3      # Jump to FindQ1Num3 function and perform
3837:       lw $s2, valueX_Q1   # Load value held in valueX_Q1 to register $s2
3838:
3839:       bne $s1, $s2, lost_Q1       # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s2 go to lost_Q1
3840:
3841:       j Correct_Q1        # Else go to Correct_Q1
3842:
3843:   secondVersion:
3844:       jal FindQ1Num1      # Jump to FindQ1Num1 function and print the first number o
f the question
3845:
3846:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3847:       la $a0, secondVer1   # Load firstVer1 into $a0 using la (load Address)
3848:       syscall   # Print out the output on screen
3849:
3850:       jal FindQ1Num2      # Jump to FindQ1Num2 function and print the second number
of the question
3851:
3852:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3853:       la $a0, firstVer2   # Load firstVer2 into $a0 using la (load Address)
3854:       syscall   # Print out the output on screen
3855:
3856:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3857:       la $a0, roots   # Load roots into $a0 using la (load Address)
3858:       syscall   # Print out the output on screen
3859:
3860:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
3861:       syscall     # Print out the output on screen
3862:
3863:       move $s1, $v0   # move value held in $v0 to register $s1
3864:
3865:       jal FindQ1Num3      # Jump to FindQ1Num3 function and perform
3866:       lw $s2, valueX_Q1   # Load value held in valueX_Q1 to register $s2
3867:
3868:       bne $s1, $s2, incorrect_Q1  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s2 go to incorrect_Q1
3869:
3870:       j Correct_Q1        # Else go to Correct_Q1
3871:
3872:   lost_Q1:
3873:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
3874:       la $a0, lost   # Load lost into $a0 using la (load Address)
3875:       syscall   # Print out the output on screen
3876:
3877:       la $a0, beep6       # Load the address of "beep6" into $a0
3878:       la $a1, duration6   # Load the address of "duration6" into $a1
3879:       la $a2, instrument6 # Load the address of "instrument6" into $a2
3880:       la $a3, volume6     # Load the address of "volume6" into $a3
```

```
3881:
3882:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
3883:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
3884:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
3885:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
3886:
3887:       # Make the system call to play the sound
3888:           li $v0, 31  # Use the "play note" system call
3889:           syscall     # Print out the output on screen
3890:
3891:       # Telling the system to stop executing the program
3892:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
3893:       syscall     # Print out the output on screen
3894:
3895:   exit_Q1:
3896:
3897:       lw   $s3, 0($sp)    # Loading the value held in register $s3 from the stack in
 the first location in the stack pointer 0 using lw (load word)
3898:       lw   $ra, 4($sp)    # Loading the nested function "FindQ1Num1" called in this
function from the stack in the second location in the stack pointer at 4 using lw (load wo
rd)
3899:       lw   $ra, 8($sp)    # Loading the nested function "FindQ1Num2" called in this
function from the stack in the second location in the stack pointer at 8 using lw (load wo
rd)
3900:       lw   $ra, 12($sp)   # Loading the nested function "FindQ1Num3" called in this
function from the stack in the second location in the stack pointer at 12 using lw (load w
ord)
3901:       lw   $ra, 16($sp)   # Loading the nested function "FindQ1Num3" called in this
function from the stack in the second location in the stack pointer at 16 using lw (load w
ord)
3902:       lw   $ra, 20($sp)   # Loading the nested function "FindQ1Num1" called in this
function from the stack in the second location in the stack pointer at 20 using lw (load w
ord)
3903:       lw   $ra, 24($sp)   # Loading the nested function "FindQ1Num2" called in this
function from the stack in the second location in the stack pointer at 24 using lw (load w
ord)
3904:       lw   $ra, 28($sp)   # Loading the nested function "FindQ1Num3" called in this
function from the stack in the second location in the stack pointer at 28 using lw (load w
ord)
3905:       addi $sp, $sp, 32   # Alocatie memeory back to the stack for 32 bytes (negativ
e because we are allocating space from the stack, positive is when we are adding space to
the stack)
3906:
3907:       jr $ra      # Finish and return to main and continue executing
3908:
3909:
3910:   # Find first number of question 1
3911:   FindQ1Num1:
3912:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 8 bytes (4 bytes for t
he value held at register $s0, and 4 bytes for the nested function) (negative because we a
re allocating space from the stack, positive is when we are adding space to the stack)
3913:       sw   $s3, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
3914:
```

```
3915:       la $t0, ArrayQ1 # base address of array MapPoints (base_address = addess of Ma
pPoints)
3916:       move $t1, $s3
3917:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3918:       li $t3, 0       # column index of value (j = 0)
3919:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3920:
3921:       # Calculate the address of value using the formula
3922:       mul $t5, $t1, $t2   # (i * m)
3923:       add $t5, $t5, $t3   # (i * m + j)
3924:       sll $t5, $t5, 2     # (i * m + j) * element_size
3925:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3926:
3927:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3928:
3929:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3930:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
3931:       syscall         # Print out the output on screen
3932:
3933:       lw $s3, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
3934:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
3935:
3936:       jr $ra
3937:
3938:
3939:   # Find the second number of question 1
3940:   FindQ1Num2:
3941:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 8 bytes (4 bytes for t
he value held at register $s0, and 4 bytes for the nested function) (negative because we a
re allocating space from the stack, positive is when we are adding space to the stack)
3942:       sw   $s3, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
3943:
3944:       la $t0, ArrayQ1 # base address of array MapPoints (base_address = addess of Ma
pPoints)
3945:       move $t1, $s3
3946:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3947:       li $t3, 1       # column index of value (j = 1)
3948:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3949:
3950:       # Calculate the address of value using the formula
3951:       mul $t5, $t1, $t2   # (i * m)
3952:       add $t5, $t5, $t3   # (i * m + j)
3953:       sll $t5, $t5, 2     # (i * m + j) * element_size
3954:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3955:
3956:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3957:
3958:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
3959:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
```

```
order to print on screen
3960:       syscall         # Print out the output on screen
3961:
3962:       lw $s3, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
3963:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
3964:
3965:       jr $ra      # Finish and return to main and continue executing
3966:
3967:
3968:   # Find the answer of question 1
3969:   FindQ1Num3:
3970:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 8 bytes (4 bytes for t
he value held at register $s0, and 4 bytes for the nested function) (negative because we a
re allocating space from the stack, positive is when we are adding space to the stack)
3971:       sw   $s3, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
3972:
3973:       la $t0, ArrayQ1 # base address of array MapPoints (base_address = addess of Ma
pPoints)
3974:       move $t1, $s3
3975:       li $t2, 3       # number of columns in array MapPoints (m = 3)
3976:       li $t3, 2       # column index of value (j = 2)
3977:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
3978:
3979:       # Calculate the address of value using the formula
3980:       mul $t5, $t1, $t2   # (i * m)
3981:       add $t5, $t5, $t3   # (i * m + j)
3982:       sll $t5, $t5, 2     # (i * m + j) * element_size
3983:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
3984:
3985:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
3986:
3987:       sw $t6, valueX_Q1       # Store value held in register $t6 in valueX since it
is the answer
3988:
3989:       lw $s3, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
3990:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
3991:
3992:       jr $ra      # Finish and return to main and continue executing
3993:
3994:
3995:   # Pseudocode:
3996:   # void PolynomialQuestionTwo(const int (*ArrayQ2)[COLS]){
3997:   #   print(Question 2)
3998:   #   while(true){
3999:   #   continue;
4000:   # }
4001:   #   while(false){
4002:   #   exit(0);
4003:   # }
4004:   # Question 2 function
```

```
4005:   PolynomialQuestionTwo:
4006:       addi $sp, $sp, -24  # Alocatie memeory in the stack for 24 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
4007:       sw   $s6, 0($sp)    # storing the value held in register $s6 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4008:       sw   $ra, 4($sp)    # storing the nested function "FindQ2Num1" called in this
function in the stack in the second location in the stack pointer at 4 using sw (store wor
d)
4009:       sw   $ra, 8($sp)    # storing the nested function "FindQ2Num2" called in this
function in the stack in the third location in the stack pointer at 8 using sw (store word
)
4010:       sw   $ra, 12($sp)   # storing the nested function "FindQ2Num3" called in this
function in the stack in the fourth location in the stack pointer at 12 using sw (store wo
rd)
4011:       sw   $ra, 16($sp)   # storing the nested function "FindQ2Num2" called in this
function in the stack in the fifth location in the stack pointer at 16 using sw (store wor
d)
4012:       sw   $ra, 20($sp)   # storing the nested function "FindQ2Num3" called in this
function in the stack in the sisth location in the stack pointer at 20 using sw (store wor
d)
4013:
4014:       # Printing Question 2 to the player
4015:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4016:       la $a0, Question2   # Load Question2 into $a0 using la (load Address)
4017:       syscall   # Print out the output on screen
4018:
4019:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4020:       la $a0, firstVer2.1   # Load firstVer2.1 into $a0 using la (load Address)
4021:       syscall   # Print out the output on screen
4022:
4023:       jal FindQ2Num1      # Jump to FindQ2Num1 function and print the first number o
f the question
4024:
4025:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4026:       la $a0, firstVer2.2   # Load firstVer2.2 into $a0 using la (load Address)
4027:       syscall   # Print out the output on screen
4028:
4029:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4030:       la $a0, roots   # Load roots into $a0 using la (load Address)
4031:       syscall   # Print out the output on screen
4032:
4033:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4034:       syscall     # Print out the output on screen
4035:       move $s1, $v0   # move value held in $v0 to register $s1
4036:
4037:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4038:       syscall     # Print out the output on screen
4039:       move $s2, $v0   # move value held in $v0 to register $s2
```

```
4040:
4041:       jal FindQ2Num2      # Jump to FindQ2Num2 and perform
4042:       lw $s4, valueX1_Q2  # Load value held in valueX1_Q2 to register $s4
4043:
4044:       jal FindQ2Num3      # Jump to FindQ2Num3 and perform
4045:       lw $s5, valueX2_Q2  # Load value held in valueX2_Q2 to register $s5
4046:
4047:       bne $s1, $s4, check_y1_Q2   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q2
4048:       bne $s1, $s5, check_y2_Q2   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q2
4049:       j check_y1_Q2       # Else, jump to check_y1_Q2
4050:
4051:   check_y1_Q2:
4052:       beq $s2, $s4, Correct_Q2    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q2
4053:
4054:       j incorrect_Q2      # Else jump to incorrect_Q2
4055:
4056:   check_y2_Q2:
4057:       beq $s2, $s5, Correct_Q2    # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to Correct_Q2
4058:
4059:       j incorrect_Q2      # Else jump to incorrect_Q2
4060:
4061:   Correct_Q2:
4062:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4063:       la $a0, correct   # Load correct into $a0 using la (load Address)
4064:       syscall   # Print out the output on screen
4065:
4066:       la $a0, sound5      # Load the address of "sound5" into $a0
4067:       la $a1, duration5   # Load the address of "duration5" into $a1
4068:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4069:       la $a3, volume5     # Load the address of "volume5" into $a3
4070:
4071:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4072:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4073:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4074:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4075:
4076:       # Make the system call to play the sound
4077:           li $v0, 31   # Use the "play note" system call
4078:           syscall     # Print out the output on screen
4079:
4080:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
4081:       li $a0, 300 # wait for 300 millisecond
4082:       syscall     # Print out the output on screen
4083:
4084:       la $a0, sound5      # Load the address of "sound5" into $a0
4085:       la $a1, duration5   # Load the address of "duration5" into $a1
4086:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4087:       la $a3, volume5     # Load the address of "volume5" into $a3
4088:
4089:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
```

```
4090:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4091:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4092:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4093:
4094:       # Make the system call to play the sound
4095:           li $v0, 31   # Use the "play note" system call
4096:           syscall     # Print out the output on screen
4097:
4098:       j exit_Q2   # Jump go to exit_Q2
4099:
4100:   incorrect_Q2:
4101:       la $a0, beep6       # Load the address of "beep6" into $a0
4102:       la $a1, duration6   # Load the address of "duration6" into $a1
4103:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4104:       la $a3, volume6     # Load the address of "volume6" into $a3
4105:
4106:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
4107:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
4108:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4109:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4110:
4111:       # Make the system call to play the sound
4112:           li $v0, 31  # Use the "play note" system call
4113:           syscall     # Print out the output on screen
4114:
4115:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4116:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
4117:       syscall   # Print out the output on screen
4118:
4119:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4120:       syscall     # Print out the output on screen
4121:       move $s1, $v0   # move value held in $v0 to register $s1
4122:
4123:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4124:       syscall     # Print out the output on screen
4125:       move $s2, $v0   # move value held in $v0 to register $s2
4126:
4127:       jal FindQ2Num2      # Jump to FindQ2Num2 and perform
4128:       lw $s4, valueX1_Q2  # Load value held in valueX1_Q2 to register $s4
4129:
4130:       jal FindQ2Num3      # Jump to FindQ2Num3 and perform
4131:       lw $s5, valueX2_Q2  # Load value held in valueX2_Q2 to register $s5
4132:
4133:       bne $s1, $s4, check_y1_Q2_  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q2_
4134:       bne $s1, $s5, check_y2_Q2_  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q2_
4135:       j check_y1_Q2_      # Else, jump to check_y1_Q2_
4136:
4137:   check_y1_Q2_:
4138:       beq $s2, $s4, Correct_Q2    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q2
```

```
4139:
4140:       j lost_Q2       # Jump to lost_Q2
4141:
4142:   check_y2_Q2_:
4143:       beq $s2, $s5, Correct_Q2    # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s5 go to Correct_Q2
4144:
4145:       j lost_Q2   # Jump to lost_Q2
4146:
4147:   lost_Q2:
4148:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4149:       la $a0, lost   # Load lost into $a0 using la (load Address)
4150:       syscall   # Print out the output on screen
4151:
4152:       la $a0, beep6       # Load the address of "beep6" into $a0
4153:       la $a1, duration6   # Load the address of "duration6" into $a1
4154:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4155:       la $a3, volume6     # Load the address of "volume6" into $a3
4156:
4157:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
4158:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
4159:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4160:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4161:
4162:       # Make the system call to play the sound
4163:           li $v0, 31  # Use the "play note" system call
4164:           syscall     # Print out the output on screen
4165:
4166:       # Telling the system to stop executing the program
4167:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
4168:       syscall     # Print out the output on screen
4169:
4170:   exit_Q2:
4171:
4172:       lw   $s6, 0($sp)    # Loading the value held in register $s6 in the stack in t
he first location in the stack pointer at 0 using sw (load word)
4173:       lw   $ra, 4($sp)    # Loading the nested function "FindQ2Num1" called in this
function from the stack in the second location in the stack pointer at 4 using lw (load wo
rd)
4174:       lw   $ra, 8($sp)    # Loading the nested function "FindQ2Num2" called in this
function from the stack in the third location in the stack pointer at 8 using lw (load wor
d)
4175:       lw   $ra, 12($sp)   # Loading the nested function "FindQ2Num3" called in this
function from the stack in the fourth location in the stack pointer at 12 using lw (load w
ord)
4176:       lw   $ra, 16($sp)   # Loading the nested function "FindQ2Num2" called in this
function from the stack in the fifth location in the stack pointer at 16 using lw (load wo
rd)
4177:       lw   $ra, 20($sp)   # Loading the nested function "FindQ2Num3" called in this
function from the stack in the sixth location in the stack pointer at 20 using lw (load wo
rd)
4178:       addi $sp, $sp, 24   # Alocatie memeory back to the stack for 24 bytes (negativ
e because we are allocating space from the stack, positive is when we are adding space to
```

```
the stack)
4179:
4180:
4181:       jr $ra      # Finish and return to main and continue executing
4182:
4183:
4184:   # Find first number of question 2
4185:   FindQ2Num1:
4186:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4187:       sw   $s6, 0($sp)    # storing the value held in register $s6 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4188:
4189:       la $t0, ArrayQ2     # base address of ArrayQ2 (base_address = addess of MapPoi
nts)
4190:       move $t1, $s6
4191:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4192:       li $t3, 0       # column index of value (j = 0)
4193:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4194:
4195:       # Calculate the address of value using the formula
4196:       mul $t5, $t1, $t2   # (i * m)
4197:       add $t5, $t5, $t3   # (i * m + j)
4198:       sll $t5, $t5, 2     # (i * m + j) * element_size
4199:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4200:
4201:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4202:
4203:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
4204:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
4205:       syscall         # Print out the output on screen
4206:
4207:       lw $s6, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4208:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4209:
4210:       jr $ra      # Finish and return to main and continue executing
4211:
4212:
4213:   # Find the answer of question 2
4214:   FindQ2Num2:
4215:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4216:       sw   $s6, 0($sp)    # storing the value held in register $s6 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4217:
4218:       la $t0, ArrayQ2     # base address of ArrayQ2 (base_address = addess of MapPoi
nts)
4219:       move $t1, $s6
4220:       li $t2, 3       # number of columns in array MapPoints (m = 3)
```

```
4221:       li $t3, 1       # column index of value (j = 1)
4222:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4223:
4224:       # Calculate the address of value using the formula
4225:       mul $t5, $t1, $t2   # (i * m)
4226:       add $t5, $t5, $t3   # (i * m + j)
4227:       sll $t5, $t5, 2     # (i * m + j) * element_size
4228:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4229:
4230:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4231:
4232:       sw $t6, valueX1_Q2      # Store value held in register $t6 in valueX1_Q2 since
 it is the answer
4233:
4234:       lw $s6, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4235:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4236:
4237:       jr $ra      # Finish and return to main and continue executing
4238:
4239:
4240:   # Find the answer of question 2
4241:   FindQ2Num3:
4242:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4243:       sw   $s6, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4244:
4245:       la $t0, ArrayQ2     # base address of ArrayQ2 (base_address = addess of MapPoi
nts)
4246:       move $t1, $s6
4247:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4248:       li $t3, 2       # column index of value (j = 2)
4249:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4250:
4251:       # Calculate the address of value using the formula
4252:       mul $t5, $t1, $t2   # (i * m)
4253:       add $t5, $t5, $t3   # (i * m + j)
4254:       sll $t5, $t5, 2     # (i * m + j) * element_size
4255:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4256:
4257:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4258:
4259:       sw $t6, valueX2_Q2      # Store value held in register $t6 in valueX2_Q2 since
 it is the answer
4260:
4261:       lw $s6, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4262:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4263:
4264:       jr $ra      # Finish and return to main and continue executing
4265:
```

```
4266:
4267:   # Pseudocode:
4268:   # void PolynomialQuestionThree(const int (*ArrayQ3)[COLS]){
4269:   #   print(Question 3)
4270:   #   while(true){
4271:   #   continue;
4272:   # }
4273:   #   while(false){
4274:   #   exit(0);
4275:   # }
4276:   # Question 3 function
4277:   PolynomialQuestionThree:
4278:       addi $sp, $sp, -36  # Alocatie memeory in the stack for 36 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
4279:       sw   $s7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4280:       sw   $ra, 4($sp)    # storing the nested function "FindQ3Num1" called in this
function in the stack in the second location in the stack pointer at 4 using sw (store wor
d)
4281:       sw   $ra, 8($sp)    # storing the nested function "FindQ3Num2" called in this
function in the stack in the third location in the stack pointer at 8 using sw (store word
)
4282:       sw   $ra, 12($sp)   # storing the nested function "FindQ3Num3" called in this
function in the stack in the fourth location in the stack pointer at 12 using sw (store wo
rd)
4283:       sw   $ra, 16($sp)   # storing the nested function "FindQ3Num2" called in this
function in the stack in the fifth location in the stack pointer at 16 using sw (store wor
d)
4284:       sw   $ra, 20($sp)   # storing the nested function "FindQ3Num3" called in this
function in the stack in the sisxth location in the stack pointer at 20 using sw (store wo
rd)
4285:       sw   $ra, 24($sp)   # storing the nested function "FindQ3Num1" called in this
function in the stack in the seventh location in the stack pointer at 24 using sw (store w
ord)
4286:       sw   $ra, 28($sp)   # storing the nested function "FindQ3Num2" called in this
function in the stack in the eighth location in the stack pointer at 28 using sw (store wo
rd)
4287:       sw   $ra, 32($sp)   # storing the nested function "FindQ3Num3" called in this
function in the stack in the eighth location in the stack pointer at 32 using sw (store wo
rd)
4288:
4289:       # Printing Question 3 to the player
4290:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
4291:       la $a0, Question3       # Load Question3 into $a0 using la (load Address)
4292:       syscall     # Print out the output on screen
4293:
4294:       addi $t3, $zero, 2  # Initialize register $t3 with value 2 using addi
4295:       div $s7, $t3        # Divide value held in register $s7 with $t3
4296:
4297:       mfhi $t4        # Load the remainder of the division to register $t4
4298:
4299:       beq $t4, $zero, firstVersion_Q3     # Branch if equal, if value held in regist
er $t4 is equal to value in $zero go to firstVersion_Q3
```

```
4300:
4301:       j secondVersion_Q3      # Else go to secondVersion_Q3
4302:
4303:   firstVersion_Q3:
4304:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4305:       la $a0, firstVer3.1   # Load firstVer3.1 into $a0 using la (load Address)
4306:       syscall   # Print out the output on screen
4307:
4308:       jal FindQ3Num1      # Jump to FindQ3Num1 function and print the first number o
f the question
4309:
4310:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4311:       la $a0, firstVer3.2   # Load firstVer3.2 into $a0 using la (load Address)
4312:       syscall   # Print out the output on screen
4313:
4314:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4315:       la $a0, roots   # Load roots into $a0 using la (load Address)
4316:       syscall   # Print out the output on screen
4317:
4318:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4319:       syscall     # Print out the output on screen
4320:       move $s1, $v0   # move value held in $v0 to register $s1
4321:
4322:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4323:       syscall     # Print out the output on screen
4324:       move $s2, $v0   # move value held in $v0 to register $s2
4325:
4326:       jal FindQ3Num2      # Jump to FindQ3Num2 and perform
4327:       lw $s4, valueX1_Q3  # Load value held in valueX1_Q3 to register $s4
4328:
4329:       jal FindQ3Num3      # Jump to FindQ3Num3 and perform
4330:       lw $s5, valueX2_Q3  # Load value held in valueX2_Q3 to register $s5
4331:
4332:       bne $s1, $s4, check_y1_Q3   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q3
4333:       bne $s1, $s5, check_y2_Q3   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q3
4334:       j check_y1_Q3       # Else, jump to check_y1_Q3
4335:
4336:   check_y1_Q3:
4337:       beq $s2, $s4, Correct_Q3    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q3
4338:
4339:       j incorrect_Q3      # Else jump to incorrect_Q3
4340:
4341:   check_y2_Q3:
4342:       beq $s2, $s5, Correct_Q3    # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to Correct_Q3
4343:
4344:       j incorrect_Q3      # Else jump to incorrect_Q3
```

```
4345:
4346:   Correct_Q3:
4347:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4348:       la $a0, correct   # Load correct into $a0 using la (load Address)
4349:       syscall   # Print out the output on screen
4350:
4351:       la $a0, sound5      # Load the address of "sound5" into $a0
4352:       la $a1, duration5   # Load the address of "duration5" into $a1
4353:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4354:       la $a3, volume5     # Load the address of "volume5" into $a3
4355:
4356:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4357:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4358:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4359:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4360:
4361:       # Make the system call to play the sound
4362:           li $v0, 31   # Use the "play note" system call
4363:           syscall     # Print out the output on screen
4364:
4365:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
4366:       li $a0, 300 # wait for 300 millisecond
4367:       syscall     # Print out the output on screen
4368:
4369:       la $a0, sound5      # Load the address of "sound5" into $a0
4370:       la $a1, duration5   # Load the address of "duration5" into $a1
4371:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4372:       la $a3, volume5     # Load the address of "volume5" into $a3
4373:
4374:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4375:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4376:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4377:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4378:
4379:       # Make the system call to play the sound
4380:           li $v0, 31   # Use the "play note" system call
4381:           syscall     # Print out the output on screen
4382:
4383:       j exit_Q3   # Jump go to exit_Q3
4384:
4385:   incorrect_Q3:
4386:       la $a0, beep6       # Load the address of "beep6" into $a0
4387:       la $a1, duration6   # Load the address of "duration6" into $a1
4388:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4389:       la $a3, volume6     # Load the address of "volume6" into $a3
4390:
4391:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
4392:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
4393:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4394:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4395:
4396:       # Make the system call to play the sound
4397:           li $v0, 31  # Use the "play note" system call
4398:           syscall     # Print out the output on screen
```

```
4399:
4400:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4401:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
4402:       syscall   # Print out the output on screen
4403:
4404:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4405:       syscall     # Print out the output on screen
4406:       move $s1, $v0   # move value held in $v0 to register $s1
4407:
4408:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4409:       syscall     # Print out the output on screen
4410:       move $s2, $v0   # move value held in $v0 to register $s2
4411:
4412:       jal FindQ3Num2      # Jump to FindQ3Num2 and perform
4413:       lw $s4, valueX1_Q3  # Load value held in valueX1_Q3 to register $s4
4414:
4415:       jal FindQ3Num3      # Jump to FindQ3Num3 and perform
4416:       lw $s5, valueX2_Q3  # Load value held in valueX2_Q3 to register $s5
4417:
4418:       bne $s1, $s4, check_y1_Q3_  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q3_
4419:       bne $s1, $s5, check_y2_Q3_  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q3_
4420:       j check_y1_Q3_      # Else, jump to check_y1_Q3_
4421:
4422:   check_y1_Q3_:
4423:       beq $s2, $s4, Correct_Q3    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q3
4424:
4425:       j lost_Q3       # Jump to lost_Q3
4426:
4427:   check_y2_Q3_:
4428:       beq $s2, $s5, Correct_Q3    # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s5 go to Correct_Q3
4429:
4430:       j lost_Q3   # Jump to lost_Q3
4431:
4432:   secondVersion_Q3:
4433:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
4434:       la $a0, secondVer3.1    # Load secondVer3.1 into $a0 using la (load Address)
4435:       syscall     # Print out the output on screen
4436:
4437:       jal FindQ3Num1      # Jump to FindQ3Num1 function and print the first number o
f the question
4438:
4439:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
4440:       la $a0, firstVer3.2      # Load firstVer3.2 into $a0 using la (load Address)
4441:       syscall     # Print out the output on screen
4442:
4443:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
```

```
egister $v0
4444:       la $a0, roots     # Load roots into $a0 using la (load Address)
4445:       syscall     # Print out the output on screen
4446:
4447:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4448:       syscall     # Print out the output on screen
4449:       move $s1, $v0   # move value held in $v0 to register $s1
4450:
4451:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4452:       syscall     # Print out the output on screen
4453:       move $s2, $v0   # move value held in $v0 to register $s2
4454:
4455:       jal FindQ3Num2      # Jump to FindQ3Num2 and perform
4456:       lw $s4, valueX1_Q3  # Load value held in valueX1_Q3 to register $s4
4457:
4458:       jal FindQ3Num3      # Jump to FindQ3Num3 and perform
4459:       lw $s5, valueX2_Q3  # Load value held in valueX2_Q3 to register $s5
4460:
4461:       bne $s1, $s4, check_y1_Q3   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q3
4462:       bne $s1, $s5, check_y2_Q3   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q3
4463:       j check_y1_Q3       # Else, jump to check_y1_Q3
4464:
4465:   lost_Q3:
4466:       la $a0, beep6       # Load the address of "beep6" into $a0
4467:       la $a1, duration6   # Load the address of "duration6" into $a1
4468:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4469:       la $a3, volume6     # Load the address of "volume6" into $a3
4470:
4471:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
4472:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
4473:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4474:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4475:
4476:       # Make the system call to play the sound
4477:           li $v0, 31  # Use the "play note" system call
4478:           syscall     # Print out the output on screen
4479:
4480:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4481:       la $a0, lost   # Load lost into $a0 using la (load Address)
4482:       syscall   # Print out the output on screen
4483:
4484:       # Telling the system to stop executing the program
4485:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
4486:       syscall     # Print out the output on screen
4487:
4488:   exit_Q3:
4489:
4490:
4491:       lw   $s7, 0($sp)    # Loading the value held in register $s3 from the stack in
```

```
 the first location in the stack pointer 0 using lw (load word)
4492:       lw   $ra, 4($sp)    # Loading the nested function "FindQ3Num1" called in this
function from the stack in the second location in the stack pointer at 4 using lw (load wo
rd)
4493:       lw   $ra, 8($sp)    # Loading the nested function "FindQ3Num2" called in this
function from the stack in the second location in the stack pointer at 8 using lw (load wo
rd)
4494:       lw   $ra, 12($sp)   # Loading the nested function "FindQ3Num3" called in this
function from the stack in the second location in the stack pointer at 12 using lw (load w
ord)
4495:       lw   $ra, 16($sp)   # Loading the nested function "FindQ3Num2" called in this
function from the stack in the second location in the stack pointer at 16 using lw (load w
ord)
4496:       lw   $ra, 20($sp)   # Loading the nested function "FindQ3Num3" called in this
function from the stack in the second location in the stack pointer at 20 using lw (load w
ord)
4497:       lw   $ra, 24($sp)   # Loading the nested function "FindQ3Num1" called in this
function from the stack in the second location in the stack pointer at 24 using lw (load w
ord)
4498:       lw   $ra, 28($sp)   # Loading the nested function "FindQ3Num2" called in this
function from the stack in the second location in the stack pointer at 28 using lw (load w
ord)
4499:       lw   $ra, 32($sp)   # Loading the nested function "FindQ3Num3" called in this
function from the stack in the second location in the stack pointer at 32 using lw (load w
ord)
4500:       addi $sp, $sp, 36   # Alocatie memeory back to the stack for 36 bytes (negativ
e because we are allocating space from the stack, positive is when we are adding space to
the stack)
4501:
4502:
4503:       jr $ra      # Finish and return to main and continue executing
4504:
4505:
4506:   # Find first number of question 3
4507:   FindQ3Num1:
4508:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4509:       sw   $s7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4510:
4511:       la $t0, ArrayQ3     # base address of Array3 (base_address = addess of MapPoin
ts)
4512:       move $t1, $s7
4513:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4514:       li $t3, 0       # column index of value (j = 0)
4515:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4516:
4517:       # Calculate the address of value using the formula
4518:       mul $t5, $t1, $t2   # (i * m)
4519:       add $t5, $t5, $t3   # (i * m + j)
4520:       sll $t5, $t5, 2     # (i * m + j) * element_size
4521:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4522:
4523:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
```

```
4524:
4525:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
4526:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
4527:       syscall         # Print out the output on screen
4528:
4529:       lw $s7, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4530:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4531:
4532:       jr $ra
4533:
4534:
4535:   # Find the answer of question 3
4536:   FindQ3Num2:
4537:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4538:       sw   $s7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4539:
4540:       la $t0, ArrayQ3     # base address of ArrayQ3 (base_address = addess of MapPoi
nts)
4541:       move $t1, $s7
4542:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4543:       li $t3, 1       # column index of value (j = 1)
4544:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4545:
4546:       # Calculate the address of value using the formula
4547:       mul $t5, $t1, $t2   # (i * m)
4548:       add $t5, $t5, $t3   # (i * m + j)
4549:       sll $t5, $t5, 2     # (i * m + j) * element_size
4550:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4551:
4552:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4553:
4554:       sw $t6, valueX1_Q3      # Store value held in register $t6 in valueX1_Q3 since
 it is the answer
4555:
4556:       lw $s7, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4557:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4558:
4559:       jr $ra      # Finish and return to main and continue executing
4560:
4561:
4562:   # Find the answer of question 3
4563:   FindQ3Num3:
4564:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4565:       sw   $s7, 0($sp)    # storing the value held in register $s3 in the stack in t
```

```
he first location in the stack pointer at 0 using sw (store word)
4566:
4567:       la $t0, ArrayQ3     # base address of ArrayQ3 (base_address = addess of MapPoi
nts)
4568:       move $t1, $s7
4569:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4570:       li $t3, 2       # column index of value (j = 2)
4571:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4572:
4573:       # Calculate the address of value using the formula
4574:       mul $t5, $t1, $t2   # (i * m)
4575:       add $t5, $t5, $t3   # (i * m + j)
4576:       sll $t5, $t5, 2     # (i * m + j) * element_size
4577:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4578:
4579:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4580:
4581:       sw $t6, valueX2_Q3      # Store value held in register $t6 in valueX2_Q3 since
 it is the answer
4582:
4583:       lw $s7, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4584:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4585:
4586:       jr $ra      # Finish and return to main and continue executing
4587:
4588:
4589:   # Pseudocode:
4590:   # void PolynomialQuestionFour(const int (*ArrayQ4)[COLS]){
4591:   #   print(Question 4)
4592:   #   while(true){
4593:   #   continue;
4594:   # }
4595:   #   while(false){
4596:   #   exit(0);
4597:   # }
4598:   # Question 4 function
4599:   PolynomialQuestionFour:
4600:       addi $sp, $sp, -24  # Alocatie memeory in the stack for 24 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
4601:       sw   $t7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4602:       sw   $ra, 4($sp)    # storing the nested function "FindQ4Num1" called in this
function in the stack in the second location in the stack pointer at 4 using sw (store wor
d)
4603:       sw   $ra, 8($sp)    # storing the nested function "FindQ4Num2" called in this
function in the stack in the third location in the stack pointer at 8 using sw (store word
)
4604:       sw   $ra, 12($sp)   # storing the nested function "FindQ4Num3" called in this
function in the stack in the fourth location in the stack pointer at 12 using sw (store wo
rd)
4605:       sw   $ra, 16($sp)   # storing the nested function "FindQ4Num2" called in this
function in the stack in the fifth location in the stack pointer at 16 using sw (store wor
```

```
d)
4606:       sw   $ra, 20($sp)   # storing the nested function "FindQ4Num3" called in this
function in the stack in the sisth location in the stack pointer at 20 using sw (store wor
d)
4607:
4608:       # Printing Question 4 to the player
4609:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4610:       la $a0, Question4   # Load Question4 into $a0 using la (load Address)
4611:       syscall   # Print out the output on screen
4612:
4613:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4614:       la $a0, firstVer4.1   # Load firstVer4.1 into $a0 using la (load Address)
4615:       syscall   # Print out the output on screen
4616:
4617:       jal FindQ4Num1      # Jump to FindQ4Num1 function and print the first number o
f the question
4618:
4619:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4620:       la $a0, firstVer4.2   # Load firstVer2.2 into $a0 using la (load Address)
4621:       syscall   # Print out the output on screen
4622:
4623:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4624:       la $a0, roots   # Load roots into $a0 using la (load Address)
4625:       syscall   # Print out the output on screen
4626:
4627:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4628:       syscall     # Print out the output on screen
4629:       move $s1, $v0   # move value held in $v0 to register $s1
4630:
4631:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4632:       syscall     # Print out the output on screen
4633:       move $s2, $v0   # move value held in $v0 to register $s2
4634:
4635:       jal FindQ4Num2      # Jump to FindQ4Num2 and perform
4636:       lw $s4, valueX1_Q4  # Load value held in valueX1_Q4 to register $s4
4637:
4638:       jal FindQ4Num3      # Jump to FindQ4Num3 and perform
4639:       lw $s5, valueX2_Q4  # Load value held in valueX2_Q4 to register $s5
4640:
4641:       bne $s1, $s4, check_y1_Q4   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q4
4642:       bne $s1, $s5, check_y2_Q4   # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q4
4643:       j check_y1_Q4       # Else, jump to check_y1_Q4
4644:
4645:   check_y1_Q4:
4646:       beq $s2, $s4, Correct_Q4    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q4
4647:
```

```
4648:       j incorrect_Q4      # Else jump to incorrect_Q4
4649:
4650:   check_y2_Q4:
4651:       beq $s2, $s5, Correct_Q4    # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to Correct_Q4
4652:
4653:       j incorrect_Q4      # Else jump to incorrect_Q4
4654:
4655:   Correct_Q4:
4656:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4657:       la $a0, correct   # Load correct into $a0 using la (load Address)
4658:       syscall   # Print out the output on screen
4659:
4660:       la $a0, sound5      # Load the address of "sound5" into $a0
4661:       la $a1, duration5   # Load the address of "duration5" into $a1
4662:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4663:       la $a3, volume5     # Load the address of "volume5" into $a3
4664:
4665:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4666:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4667:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4668:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4669:
4670:       # Make the system call to play the sound
4671:           li $v0, 31   # Use the "play note" system call
4672:           syscall     # Print out the output on screen
4673:
4674:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
4675:       li $a0, 300 # wait for 300 millisecond
4676:       syscall     # Print out the output on screen
4677:
4678:       la $a0, sound5      # Load the address of "sound5" into $a0
4679:       la $a1, duration5   # Load the address of "duration5" into $a1
4680:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4681:       la $a3, volume5     # Load the address of "volume5" into $a3
4682:
4683:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4684:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4685:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4686:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4687:
4688:       # Make the system call to play the sound
4689:           li $v0, 31   # Use the "play note" system call
4690:           syscall     # Print out the output on screen
4691:
4692:       j exit_Q4   # Jump go to exit_Q2
4693:
4694:   incorrect_Q4:
4695:       la $a0, beep6       # Load the address of "beep6" into $a0
4696:       la $a1, duration6   # Load the address of "duration6" into $a1
4697:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4698:       la $a3, volume6     # Load the address of "volume6" into $a3
4699:
4700:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
```

```
4701:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
4702:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4703:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4704:
4705:       # Make the system call to play the sound
4706:           li $v0, 31  # Use the "play note" system call
4707:           syscall     # Print out the output on screen
4708:
4709:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4710:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
4711:       syscall   # Print out the output on screen
4712:
4713:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4714:       syscall     # Print out the output on screen
4715:       move $s1, $v0   # move value held in $v0 to register $s1
4716:
4717:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4718:       syscall     # Print out the output on screen
4719:       move $s2, $v0   # move value held in $v0 to register $s2
4720:
4721:       jal FindQ4Num2      # Jump to FindQ4Num2 and perform
4722:       lw $s4, valueX1_Q4  # Load value held in valueX1_Q2 to register $s4
4723:
4724:       jal FindQ4Num3      # Jump to FindQ4Num3 and perform
4725:       lw $s5, valueX2_Q4  # Load value held in valueX2_Q2 to register $s5
4726:
4727:       bne $s1, $s4, check_y1_Q4_  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s4 go to check_y1_Q4_
4728:       bne $s1, $s5, check_y2_Q4_  # Branch if not equal, if value held in register $
s1 is NOT equal to value in $s5 go to check_y2_Q4_
4729:       j check_y1_Q4_      # Else, jump to check_y1_Q4_
4730:
4731:   check_y1_Q4_:
4732:       beq $s2, $s4, Correct_Q4    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q4
4733:
4734:       j lost_Q4       # Jump to lost_Q4
4735:
4736:   check_y2_Q4_:
4737:       beq $s2, $s5, Correct_Q4    # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s5 go to Correct_Q4
4738:
4739:       j lost_Q4   # Jump to lost_Q4
4740:
4741:   lost_Q4:
4742:       la $a0, beep6       # Load the address of "beep6" into $a0
4743:       la $a1, duration6   # Load the address of "duration6" into $a1
4744:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4745:       la $a3, volume6     # Load the address of "volume6" into $a3
4746:
4747:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
4748:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
```

```
4749:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4750:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4751:
4752:       # Make the system call to play the sound
4753:           li $v0, 31  # Use the "play note" system call
4754:           syscall     # Print out the output on screen
4755:
4756:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4757:       la $a0, lost   # Load lost into $a0 using la (load Address)
4758:       syscall   # Print out the output on screen
4759:
4760:       # Telling the system to stop executing the program
4761:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
4762:       syscall     # Print out the output on screen
4763:
4764:   exit_Q4:
4765:
4766:       lw   $t7, 0($sp)    # Loading the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (load word)
4767:       lw   $ra, 4($sp)    # Loading the nested function "FindQ4Num1" called in this
function from the stack in the second location in the stack pointer at 4 using lw (load wo
rd)
4768:       lw   $ra, 8($sp)    # Loading the nested function "FindQ4Num2" called in this
function from the stack in the third location in the stack pointer at 8 using lw (load wor
d)
4769:       lw   $ra, 12($sp)   # Loading the nested function "FindQ4Num3" called in this
function from the stack in the fourth location in the stack pointer at 12 using lw (load w
ord)
4770:       lw   $ra, 16($sp)   # Loading the nested function "FindQ4Num2" called in this
function from the stack in the fifth location in the stack pointer at 16 using lw (load wo
rd)
4771:       lw   $ra, 20($sp)   # Loading the nested function "FindQ4Num3" called in this
function from the stack in the sixth location in the stack pointer at 20 using lw (load wo
rd)
4772:       addi $sp, $sp, 24   # Alocatie memeory back to the stack for 24 bytes (negativ
e because we are allocating space from the stack, positive is when we are adding space to
the stack)
4773:
4774:
4775:       jr $ra      # Finish and return to main and continue executing
4776:
4777:
4778:   # Find first number of question 4
4779:   FindQ4Num1:
4780:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4781:       sw   $t7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4782:
4783:       la $t0, ArrayQ4     # base address of ArrayQ4 (base_address = addess of MapPoi
nts)
4784:       move $t1, $t7
```

```
4785:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4786:       li $t3, 0       # column index of value (j = 0)
4787:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4788:
4789:       # Calculate the address of value using the formula
4790:       mul $t5, $t1, $t2   # (i * m)
4791:       add $t5, $t5, $t3   # (i * m + j)
4792:       sll $t5, $t5, 2     # (i * m + j) * element_size
4793:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4794:
4795:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4796:
4797:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
4798:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
4799:       syscall         # Print out the output on screen
4800:
4801:       lw $t7, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4802:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4803:
4804:       jr $ra      # Finish and return to main and continue executing
4805:
4806:
4807:   # Find the answer of question 4
4808:   FindQ4Num2:
4809:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4810:       sw   $t7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4811:
4812:       la $t0, ArrayQ4     # base address of ArrayQ4 (base_address = addess of MapPoi
nts)
4813:       move $t1, $t7
4814:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4815:       li $t3, 1       # column index of value (j = 1)
4816:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4817:
4818:       # Calculate the address of value using the formula
4819:       mul $t5, $t1, $t2   # (i * m)
4820:       add $t5, $t5, $t3   # (i * m + j)
4821:       sll $t5, $t5, 2     # (i * m + j) * element_size
4822:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4823:
4824:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4825:
4826:       sw $t6, valueX1_Q4      # Store value held in register $t6 in valueX1_Q4 since
 it is the answer
4827:
4828:       lw $t7, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4829:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
```

```
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4830:
4831:       jr $ra      # Finish and return to main and continue executing
4832:
4833:
4834:   # Find the answer of question 4
4835:   FindQ4Num3:
4836:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 8 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
4837:       sw   $t7, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
4838:
4839:       la $t0, ArrayQ4     # base address of ArrayQ4 (base_address = addess of MapPoi
nts)
4840:       move $t1, $t7
4841:       li $t2, 3       # number of columns in array MapPoints (m = 3)
4842:       li $t3, 2       # column index of value (j = 2)
4843:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
4844:
4845:       # Calculate the address of value 10 using the formula
4846:       mul $t5, $t1, $t2   # (i * m)
4847:       add $t5, $t5, $t3   # (i * m + j)
4848:       sll $t5, $t5, 2     # (i * m + j) * element_size
4849:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
4850:
4851:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
4852:
4853:       sw $t6, valueX2_Q4      # Store value held in register $t6 in valueX2_Q2 since
 it is the answer
4854:
4855:       lw $t7, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
4856:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
4857:
4858:       jr $ra      # Finish and return to main and continue executing
4859:
4860:
4861:   # Pseudocode:
4862:   # void PolynomialQuestionFive(const int (*ArrayQ5)[COLS]){
4863:   #   print(Question 5)
4864:   #   while(true){
4865:   #   continue;
4866:   # }
4867:   #   while(false){
4868:   #   exit(0);
4869:   # }
4870:   # Question 5 function
4871:   PolynomialQuestionFive:
4872:       addi $sp, $sp, -36  # Alocatie memeory in the stack for 36 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
4873:       sw   $t8, 0($sp)    # storing the value held in register $t8 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
```

```
4874:       sw   $ra, 4($sp)    # storing the nested function "FindQ5Num1" called in this
function in the stack in the second location in the stack pointer at 4 using sw (store wor
d)
4875:       sw   $ra, 8($sp)    # storing the nested function "FindQ5Num2" called in this
function in the stack in the third location in the stack pointer at 8 using sw (store word
)
4876:       sw   $ra, 12($sp)   # storing the nested function "FindQ5Num3" called in this
function in the stack in the fourth location in the stack pointer at 12 using sw (store wo
rd)
4877:       sw   $ra, 16($sp)   # storing the nested function "FindQ5Num2" called in this
function in the stack in the fifth location in the stack pointer at 16 using sw (store wor
d)
4878:       sw   $ra, 20($sp)   # storing the nested function "FindQ5Num3" called in this
function in the stack in the sisxth location in the stack pointer at 20 using sw (store wo
rd)
4879:       sw   $ra, 24($sp)   # storing the nested function "FindQ5Num1" called in this
function in the stack in the seventh location in the stack pointer at 24 using sw (store w
ord)
4880:       sw   $ra, 28($sp)   # storing the nested function "FindQ5Num2" called in this
function in the stack in the eighth location in the stack pointer at 28 using sw (store wo
rd)
4881:       sw   $ra, 32($sp)   # storing the nested function "FindQ5Num3" called in this
function in the stack in the eighth location in the stack pointer at 32 using sw (store wo
rd)
4882:
4883:       # Printing Question 5 to the player
4884:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
4885:       la $a0, Question5       # Load Question5 into $a0 using la (load Address)
4886:       syscall     # Print out the output on screen
4887:
4888:       addi $t3, $zero, 2  # Initialize register $t3 with value 2 using addi
4889:       div $t8, $t3        # Divide value held in register $t8 with $t3
4890:
4891:       mfhi $t4        # Load the remainder of the division to register $t4
4892:
4893:       beq $t4, $zero, firstVersion_Q5     # Branch if equal, if value held in regist
er $t4 is equal to value in $zero go to firstVersion_Q5
4894:
4895:       j secondVersion_Q5      # Else go to secondVersion_Q5
4896:
4897:   firstVersion_Q5:
4898:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4899:       la $a0, firstVer5.1   # Load firstVer5.1 into $a0 using la (load Address)
4900:       syscall   # Print out the output on screen
4901:
4902:       jal FindQ5Num1      # Jump to FindQ5Num1 function and print the first number o
f the question
4903:
4904:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4905:       la $a0, firstVer5.2   # Load firstVer5.2 into $a0 using la (load Address)
4906:       syscall   # Print out the output on screen
4907:
```

```
4908:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4909:       la $a0, roots   # Load roots into $a0 using la (load Address)
4910:       syscall   # Print out the output on screen
4911:
4912:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4913:       syscall     # Print out the output on screen
4914:       move $s1, $v0   # move value held in $v0 to register $s1
4915:
4916:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
4917:       syscall     # Print out the output on screen
4918:       move $s2, $v0   # move value held in $v0 to register $s2
4919:
4920:       jal FindQ5Num2      # Jump to FindQ5Num2 and perform
4921:       lw $s4, valueX1_Q5  # Load value held in valueX1_Q5 to register $s4
4922:
4923:       jal FindQ5Num3      # Jump to FindQ5Num3 and perform
4924:       lw $s5, valueX2_Q5  # Load value held in valueX2_Q5 to register $s5
4925:
4926:       beq $s1, $s4, check_y1_Q5   # Branch if equal, if value held in register $s1 i
s equal to value in $s4 go to check_y1_Q5
4927:       beq $s1, $s5, check_y1_Q5   # Branch if equal, if value held in register $s1 i
s equal to value in $s5 go to check_y1_Q5
4928:       j check_y2_Q5       # Else, jump to check_y2_Q5
4929:
4930:   check_y1_Q5:
4931:       beq $s2, $s4, Correct_Q5    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q5
4932:       beq $s2, $s5, Correct_Q5    # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to Correct_Q5
4933:       j incorrect_Q5      # Else jump to incorrect_Q5
4934:
4935:   check_y2_Q5:
4936:       bne $s2, $s4, incorrect_Q5  # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s4 go to incorrect_Q5
4937:       bne $s2, $s5, incorrect_Q5  # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s5 go to incorrect_Q5
4938:
4939:       j Correct_Q5        # Else jump to Correct_Q5
4940:
4941:   Correct_Q5:
4942:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4943:       la $a0, correct   # Load correct into $a0 using la (load Address)
4944:       syscall   # Print out the output on screen
4945:
4946:       la $a0, sound5      # Load the address of "sound5" into $a0
4947:       la $a1, duration5   # Load the address of "duration5" into $a1
4948:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4949:       la $a3, volume5     # Load the address of "volume5" into $a3
4950:
4951:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4952:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
```

```
4953:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4954:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4955:
4956:       # Make the system call to play the sound
4957:           li $v0, 31   # Use the "play note" system call
4958:           syscall     # Print out the output on screen
4959:
4960:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
4961:       li $a0, 300 # wait for 300 millisecond
4962:       syscall     # Print out the output on screen
4963:
4964:       la $a0, sound5      # Load the address of "sound5" into $a0
4965:       la $a1, duration5   # Load the address of "duration5" into $a1
4966:       la $a2, instrument5     # Load the address of "instrument5" into $a2
4967:       la $a3, volume5     # Load the address of "volume5" into $a3
4968:
4969:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
4970:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
4971:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
4972:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
4973:
4974:       # Make the system call to play the sound
4975:           li $v0, 31   # Use the "play note" system call
4976:           syscall     # Print out the output on screen
4977:
4978:       j exit_Q5   # Jump go to exit_Q5
4979:
4980:   incorrect_Q5:
4981:       la $a0, beep6       # Load the address of "beep6" into $a0
4982:       la $a1, duration6   # Load the address of "duration6" into $a1
4983:       la $a2, instrument6 # Load the address of "instrument6" into $a2
4984:       la $a3, volume6     # Load the address of "volume6" into $a3
4985:
4986:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
4987:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
4988:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
4989:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
4990:
4991:       # Make the system call to play the sound
4992:           li $v0, 31  # Use the "play note" system call
4993:           syscall     # Print out the output on screen
4994:
4995:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
4996:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
4997:       syscall   # Print out the output on screen
4998:
4999:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5000:       syscall     # Print out the output on screen
5001:       move $s1, $v0   # move value held in $v0 to register $s1
5002:
5003:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5004:       syscall     # Print out the output on screen
```

```
5005:       move $s2, $v0   # move value held in $v0 to register $s2
5006:
5007:       jal FindQ5Num2      # Jump to FindQ5Num2 and perform
5008:       lw $s4, valueX1_Q5  # Load value held in valueX1_Q5 to register $s4
5009:
5010:       jal FindQ5Num3      # Jump to FindQ5Num3 and perform
5011:       lw $s5, valueX2_Q5  # Load value held in valueX2_Q5 to register $s5
5012:
5013:       beq $s1, $s4, check_y1_Q5_  # Branch if equal, if value held in register $s1 i
s equal to value in $s4 go to check_y1_Q5
5014:       beq $s1, $s5, check_y1_Q5_  # Branch if equal, if value held in register $s1 i
s equal to value in $s5 go to check_y1_Q5
5015:       j check_y2_Q5_      # Else, jump to check_y3_Q5
5016:
5017:   check_y1_Q5_:
5018:       beq $s2, $s4, Correct_Q5    # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to Correct_Q5
5019:       beq $s2, $s5, Correct_Q5    # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to Correct_Q5
5020:       j lost_Q5       # Else jump to lost_Q5
5021:
5022:   check_y2_Q5_:
5023:       bne $s2, $s4, lost_Q5   # Branch if not equal, if value held in register $s2 i
s NOT equal to value in $s5 go to lost_Q5
5024:       bne $s2, $s5, lost_Q5   # Branch if not equal, if value held in register $s2 i
s NOT equal to value in $s5 go to lost_Q5
5025:
5026:       j lost_Q5       # Else jump to lost_Q5
5027:
5028:   secondVersion_Q5:
5029:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
5030:       la $a0, secondVer5.1    # Load secondVer5.1 into $a0 using la (load Address)
5031:       syscall     # Print out the output on screen
5032:
5033:       jal FindQ5Num1      # Jump to FindQ5Num1 function and print the first number o
f the question
5034:
5035:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
5036:       la $a0, firstVer5.2      # Load firstVer5.2 into $a0 using la (load Address)
5037:       syscall     # Print out the output on screen
5038:
5039:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
5040:       la $a0, roots     # Load roots into $a0 using la (load Address)
5041:       syscall     # Print out the output on screen
5042:
5043:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5044:       syscall     # Print out the output on screen
5045:       move $s1, $v0   # move value held in $v0 to register $s1
5046:
5047:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
```

```
5048:       syscall     # Print out the output on screen
5049:       move $s2, $v0   # move value held in $v0 to register $s2
5050:
5051:       jal FindQ5Num2      # Jump to FindQ3Num2 and perform
5052:       lw $s4, valueX1_Q5  # Load value held in valueX1_Q5 to register $s4
5053:
5054:       jal FindQ5Num3      # Jump to FindQ3Num3 and perform
5055:       lw $s5, valueX2_Q5  # Load value held in valueX2_Q5 to register $s5
5056:
5057:       beq $s1, $s4, check_y1_Q5   # Branch if equal, if value held in register $s1 i
s equal to value in $s4 go to check_y1_Q5
5058:       beq $s1, $s5, check_y1_Q5   # Branch if equal, if value held in register $s1 i
s equal to value in $s5 go to check_y1_Q5
5059:       j check_y2_Q5       # Else, jump to check_y2_Q5
5060:
5061:   lost_Q5:
5062:       la $a0, beep6       # Load the address of "beep6" into $a0
5063:       la $a1, duration6   # Load the address of "duration6" into $a1
5064:       la $a2, instrument6 # Load the address of "instrument6" into $a2
5065:       la $a3, volume6     # Load the address of "volume6" into $a3
5066:
5067:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
5068:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
5069:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
5070:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
5071:
5072:       # Make the system call to play the sound
5073:           li $v0, 31  # Use the "play note" system call
5074:           syscall     # Print out the output on screen
5075:
5076:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5077:       la $a0, lost   # Load lost into $a0 using la (load Address)
5078:       syscall   # Print out the output on screen
5079:
5080:       # Telling the system to stop executing the program
5081:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
5082:       syscall     # Print out the output on screen
5083:
5084:   exit_Q5:
5085:
5086:
5087:       lw   $t8, 0($sp)    # Loading the value held in register $t8 from the stack in
 the first location in the stack pointer 0 using lw (load word)
5088:       lw   $ra, 4($sp)    # Loading the nested function "FindQ5Num1" called in this
function from the stack in the second location in the stack pointer at 4 using lw (load wo
rd)
5089:       lw   $ra, 8($sp)    # Loading the nested function "FindQ5Num2" called in this
function from the stack in the second location in the stack pointer at 8 using lw (load wo
rd)
5090:       lw   $ra, 12($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 12 using lw (load w
ord)
5091:       lw   $ra, 16($sp)   # Loading the nested function "FindQ5Num2" called in this
```

```
function from the stack in the second location in the stack pointer at 16 using lw (load w
ord)
5092:       lw   $ra, 20($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 20 using lw (load w
ord)
5093:       lw   $ra, 24($sp)   # Loading the nested function "FindQ5Num1" called in this
function from the stack in the second location in the stack pointer at 24 using lw (load w
ord)
5094:       lw   $ra, 28($sp)   # Loading the nested function "FindQ5Num2" called in this
function from the stack in the second location in the stack pointer at 28 using lw (load w
ord)
5095:       lw   $ra, 32($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 32 using lw (load w
ord)
5096:       addi $sp, $sp, 36   # Alocatie memeory back to the stack for 36 bytes (negativ
e because we are allocating space from the stack, positive is when we are adding space to
the stack)
5097:
5098:       jr $ra      # Finish and return to main and continue executing
5099:
5100:
5101:   # Find first number of question 5
5102:   FindQ5Num1:
5103:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
5104:       sw   $t8, 0($sp)    # storing the value held in register $t8 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
5105:
5106:       la $t0, ArrayQ5     # base address of Array5 (base_address = addess of MapPoin
ts)
5107:       move $t1, $t8
5108:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5109:       li $t3, 0       # column index of value (j = 0)
5110:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5111:
5112:       # Calculate the address of value using the formula
5113:       mul $t5, $t1, $t2   # (i * m)
5114:       add $t5, $t5, $t3   # (i * m + j)
5115:       sll $t5, $t5, 2     # (i * m + j) * element_size
5116:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5117:
5118:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5119:
5120:       li $v0, 1       # Telling the system to print an INTEGER by putting value 1 in
to register $v0
5121:       add $a0, $zero, $t6 # add/move value held in register $t6 to $a0 using add in
order to print on screen
5122:       syscall         # Print out the output on screen
5123:
5124:       lw $t8, 0($sp)      # Loading/restore the original value held in the stack at
register $t8 back to register $s0 using lw (load word)
5125:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
5126:
```

```
5127:       jr $ra
5128:
5129:
5130:   # Find the answer of question 5
5131:   FindQ5Num2:
5132:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
5133:       sw   $t8, 0($sp)    # storing the value held in register $t8 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
5134:
5135:       la $t0, ArrayQ5     # base address of ArrayQ5 (base_address = addess of MapPoi
nts)
5136:       move $t1, $t8
5137:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5138:       li $t3, 1       # column index of value (j = 1)
5139:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5140:
5141:       # Calculate the address of value using the formula
5142:       mul $t5, $t1, $t2   # (i * m)
5143:       add $t5, $t5, $t3   # (i * m + j)
5144:       sll $t5, $t5, 2     # (i * m + j) * element_size
5145:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5146:
5147:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5148:
5149:       sw $t6, valueX1_Q5      # Store value held in register $t6 in valueX1_Q5 since
 it is the answer
5150:
5151:       lw $t8, 0($sp)      # Loading/restore the original value held in the stack at
register $t8 back to register $s0 using lw (load word)
5152:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
5153:
5154:       jr $ra      # Finish and return to main and continue executing
5155:
5156:
5157:   # Find the answer of question 5
5158:   FindQ5Num3:
5159:       addi $sp, $sp, -4   # Alocatie memeory in the stack for 4 bytes (negative beca
use we are allocating space from the stack, positive is when we are adding space to the st
ack)
5160:       sw   $t8, 0($sp)    # storing the value held in register $t8 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
5161:
5162:       la $t0, ArrayQ5     # base address of ArrayQ5 (base_address = addess of MapPoi
nts)
5163:       move $t1, $t8
5164:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5165:       li $t3, 2       # column index of value (j = 2)
5166:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5167:
5168:       # Calculate the address of value using the formula
5169:       mul $t5, $t1, $t2   # (i * m)
5170:       add $t5, $t5, $t3   # (i * m + j)
```

```
5171:       sll $t5, $t5, 2     # (i * m + j) * element_size
5172:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5173:
5174:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5175:
5176:       sw $t6, valueX2_Q5      # Store value held in register $t6 in valueX2_Q5 since
 it is the answer
5177:
5178:       lw $t8, 0($sp)      # Loading/restore the original value held in the stack at
register $s0 back to register $s0 using lw (load word)
5179:       addi $sp, $sp, 4    # Restore/add space back to the stack by adding 4 to it (t
he 4 bytes that we allocated from the stack at the beginning of the fucntion)
5180:
5181:       jr $ra      # Finish and return to main and continue executing
5182:
5183:
5184:   # Pseudocode:
5185:   # void PolynomialQuestionSix(const int* ArrayQ6Int, const int (*ArrayQ6Ans)[COLS])
{
5186:   #   print(Question 6)
5187:   #   while(true){
5188:   #   continue;
5189:   # }
5190:   #   while(false){
5191:   #   exit(0);
5192:   # }
5193:   # Question 6 function
5194:   PolynomialQuestionSix:
5195:       addi $sp, $sp, -52  # Alocatie memeory in the stack for 40 bytes (negative bec
ause we are allocating space from the stack, positive is when we are adding space to the s
tack)
5196:       sw   $a3, 0($sp)    # storing the value held in register $s3 in the stack in t
he first location in the stack pointer at 0 using sw (store word)
5197:       sw   $ra, 4($sp)    # storing the nested function "FindQ6Num1" called in this
function in the stack in the second location in the stack pointer at 4 using sw (store wor
d)
5198:       sw   $ra, 8($sp)    # storing the nested function "FindQ6Num2" called in this
function in the stack in the third location in the stack pointer at 8 using sw (store word
)
5199:       sw   $ra, 12($sp)   # storing the nested function "FindQ6Num3" called in this
function in the stack in the fourth location in the stack pointer at 12 using sw (store wo
rd)
5200:       sw   $ra, 16($sp)   # storing the nested function "FindQ6Num1" called in this
function in the stack in the fifth location in the stack pointer at 16 using sw (store wor
d)
5201:       sw   $ra, 20($sp)   # storing the nested function "FindQ6Num2" called in this
function in the stack in the sisxth location in the stack pointer at 20 using sw (store wo
rd)
5202:       sw   $ra, 24($sp)   # storing the nested function "FindQ6Num3" called in this
function in the stack in the seventh location in the stack pointer at 24 using sw (store w
ord)
5203:       sw   $ra, 28($sp)   # storing the nested function "FindQ6Num1" called in this
function in the stack in the eighth location in the stack pointer at 28 using sw (store wo
rd)
5204:       sw   $ra, 32($sp)   # storing the nested function "FindQ6Num2" called in this
```

```
function in the stack in the eighth location in the stack pointer at 32 using sw (store wo
rd)
5205:       sw   $ra, 36($sp)   # storing the nested function "FindQ6Num3" called in this
function in the stack in the eighth location in the stack pointer at 36 using sw (store wo
rd)
5206:       sw   $ra, 40($sp)   # storing the nested function "FindQ6Num1" called in this
function in the stack in the eighth location in the stack pointer at 40 using sw (store wo
rd)
5207:       sw   $ra, 44($sp)   # storing the nested function "FindQ6Num2" called in this
function in the stack in the eighth location in the stack pointer at 42 using sw (store wo
rd)
5208:       sw   $ra, 48($sp)   # storing the nested function "FindQ6Num3" called in this
function in the stack in the eighth location in the stack pointer at 48 using sw (store wo
rd)
5209:
5210:       # Printing Question 6 to the player
5211:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
5212:       la $a0, Question6       # Load Question6 into $a0 using la (load Address)
5213:       syscall     # Print out the output on screen
5214:
5215:       addi $t3, $zero, 2  # Initialize register $t3 with value 2 using addi
5216:       div $a3, $t3        # Divide value held in register $s3 with $t3
5217:
5218:       mfhi $t4        # Load the remainder of the division to register $t4
5219:
5220:       beq $t4, $zero, firstVersion_Q6     # Branch if equal, if value held in regist
er $t4 is equal to value in $zero go to firstVersion_Q6
5221:
5222:       j secondVersion_Q6      # Else go to secondVersion_Q6
5223:
5224:   firstVersion_Q6:
5225:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5226:       la $a0, firstVer6.1  # Load firstVer6.1 into $a0 using la (load Address)
5227:       syscall   # Print out the output on screen
5228:
5229:       # Printing value 6 from ArrayQ6Int at ArrayQ6Int[0]
5230:       lw $t5, ArrayQ6Int($zero)    # Loading value at index 0 in 1D ArrayQ6Int Quest
ion 6 IF WE WANT TO PRINT IT by using lw (load word)
5231:
5232:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5233:       addi $a0, $t5, 0     # Add value held in register $t5 to register $a0 in order
 to print out on screen
5234:       syscall     # Print out the output on screen
5235:
5236:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5237:       la $a0, firstVer6.2   # Load firstVer6.2 into $a0 using la (load Address)
5238:       syscall   # Print out the output on screen
5239:
5240:       # Printing value 11 from ArrayQ6Int at ArrayQ6Int[1]
5241:       li $t9, 4   # Loading register $t9 with the index 4
5242:           lw $t6, ArrayQ6Int($t9)     # Loading value at index 4 in 1D ArrayQ6Int Qu
```

```
estion 6 IF WE WANT TO PRINT IT by using lw (load word)
5243:
5244:           li $v0, 1   # Telling the system to print an INTEGER by putting value 1 in
to register $v0
5245:           move $a0, $t6   # Moving value held in register $t6 to register $a0 in ord
er to print out on screen
5246:       syscall     # Print out the output on screen
5247:
5248:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5249:       la $a0, firstVer6.3   # Load firstVer6.3 into $a0 using la (load Address)
5250:       syscall   # Print out the output on screen
5251:
5252:       # Printing value 6 from ArrayQ6Int at ArrayQ6Int[2]
5253:       addi $t9, $t9, 4    # Changing register $t9 to index 8 by adding 4 to its inde
x since it is at index 4
5254:       lw $t7, ArrayQ6Int($t9)     # Loading value at index 8 in 1D ArrayQ6Int Questi
on 6 IF WE WANT TO PRINT IT by using lw (load word)
5255:
5256:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5257:       move $a0, $t7   # Moving value held in register $t7 to register $a0 in order t
o print out on screen
5258:       syscall     # Print out the output on screen
5259:
5260:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5261:       la $a0, firstVer6.4   # Load firstVer6.4 into $a0 using la (load Address)
5262:       syscall   # Print out the output on screen
5263:
5264:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5265:       la $a0, roots   # Load roots into $a0 using la (load Address)
5266:       syscall   # Print out the output on screen
5267:
5268:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5269:       syscall     # Print out the output on screen
5270:       move $s1, $v0   # move value held in $v0 to register $s1
5271:
5272:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5273:       syscall     # Print out the output on screen
5274:       move $s2, $v0   # move value held in $v0 to register $s2
5275:
5276:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5277:       syscall     # Print out the output on screen
5278:       move $s6, $v0   # move value held in $v0 to register $s6
5279:
5280:       jal FindQ6Num1      # Jump to FindQ6Num1 and perform
5281:       lw $s4, valueX1_Q6  # Load value held in valueX1_Q6 to register $s4
5282:
5283:       jal FindQ6Num2      # Jump to FindQ6Num2 and perform
5284:       lw $s5, valueX2_Q6  # Load value held in valueX2_Q6 to register $s5
```

```
5285:
5286:       jal FindQ6Num3      # Jump to FindQ6Num3 and perform
5287:       lw $s7, valueX3_Q6  # Load value held in valueX3_Q6 to register $s7
5288:
5289:       beq $s1, $s4, check_y1_Q6   # Branch if equal, if value held in register $s1 i
s equal to value in $s4 go to check_y1_Q6
5290:       beq $s1, $s5, check_y2_Q6   # Branch if equal, if value held in register $s1 i
s equal to value in $s5 go to check_y2_Q6
5291:       beq $s1, $s7, check_y3_Q6   # Branch if equal, if value held in register $s1 i
s equal to value in $s7 go to check_y3_Q6
5292:       j _check_y1_Q6      # Else, jump to _check_y1_Q6
5293:
5294:   check_y1_Q6:
5295:       beq $s2, $s5, check_z1_Q6   # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z1_Q6
5296:       beq $s2, $s7, check_z2_Q6   # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z2_Q6
5297:       j _check_z1_Q6      # Else jump to _check_z1_Q6
5298:
5299:   check_y2_Q6:
5300:       beq $s2, $s4, check_z1_Q6   # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z1_Q6
5301:       beq $s2, $s7, check_z3_Q6   # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z3_Q6
5302:       j _check_z1_Q6      # Else jump to _check_z1_Q6
5303:
5304:   check_y3_Q6:
5305:       beq $s2, $s4, check_z2_Q6   # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z2_Q6
5306:       beq $s2, $s5, check_z3_Q6   # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z3_Q6
5307:       j _check_z1_Q6      # Else jump to _check_z1_Q6
5308:
5309:   _check_y1_Q6:
5310:       bne $s2, $s4, _check_z1_Q6  # Branch not if equal, if value held in register $
s2 is NOT equal to value in $s4 go to _check_z1_Q6
5311:       bne $s2, $s5, _check_z1_Q6  # Branch not if equal, if value held in register $
s2 is NOT equal to value in $s5 go to _check_z1_Q6
5312:       bne $s2, $s7, _check_z1_Q6  # Branch not if equal, if value held in register $
s2 is NOT equal to value in $s7 go to _check_z1_Q6
5313:       j _check_z1_Q6      # Else jump to check_z1_Q6
5314:
5315:   check_z1_Q6:
5316:       beq $s6, $s7, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s7 go to Correct_Q6
5317:       j incorrect_Q6      # Else jump to incorrect_Q6
5318:
5319:   check_z2_Q6:
5320:       beq $s6, $s5, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s5 go to Correct_Q6
5321:       j incorrect_Q6      # Else jump to incorrect_Q6
5322:
5323:   check_z3_Q6:
5324:       beq $s6, $s4, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s4 go to Correct_Q6
```

```
5325:       j incorrect_Q6      # Else jump to incorrect_Q6
5326:
5327:   _check_z1_Q6:
5328:       bne $s6, $s4, incorrect_Q6  # Branch if not equal, if value held in register $
s6 is NOT equal to value in $s4 go to incorrect_Q6
5329:       bne $s6, $s5, incorrect_Q6  # Branch if not equal, if value held in register $
s6 is NOT equal to value in $s5 go to incorrect_Q6
5330:       bne $s6, $s7, incorrect_Q6  # Branch if not equal, if value held in register $
s6 is NOT equal to value in $s7 go to incorrect_Q6
5331:       j incorrect_Q6      # Else jump to incorrect_Q6
5332:
5333:
5334:   Correct_Q6:
5335:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5336:       la $a0, correct   # Load correct into $a0 using la (load Address)
5337:       syscall   # Print out the output on screen
5338:
5339:       la $a0, sound5      # Load the address of "sound5" into $a0
5340:       la $a1, duration5   # Load the address of "duration5" into $a1
5341:       la $a2, instrument5     # Load the address of "instrument5" into $a2
5342:       la $a3, volume5     # Load the address of "volume5" into $a3
5343:
5344:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
5345:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
5346:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
5347:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
5348:
5349:       # Make the system call to play the sound
5350:           li $v0, 31   # Use the "play note" system call
5351:           syscall     # Print out the output on screen
5352:
5353:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
5354:       li $a0, 300 # wait for 300 millisecond
5355:       syscall     # Print out the output on screen
5356:
5357:       la $a0, sound5      # Load the address of "sound5" into $a0
5358:       la $a1, duration5   # Load the address of "duration5" into $a1
5359:       la $a2, instrument5     # Load the address of "instrument5" into $a2
5360:       la $a3, volume5     # Load the address of "volume5" into $a3
5361:
5362:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
5363:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
5364:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
5365:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
5366:
5367:       # Make the system call to play the sound
5368:           li $v0, 31   # Use the "play note" system call
5369:           syscall     # Print out the output on screen
5370:
5371:       j exit_Q6   # Jump go to exit_Q6
5372:
5373:   incorrect_Q6:
5374:       la $a0, beep6       # Load the address of "beep6" into $a0
5375:       la $a1, duration6   # Load the address of "duration6" into $a1
```

```
5376:       la $a2, instrument6 # Load the address of "instrument6" into $a2
5377:       la $a3, volume6     # Load the address of "volume6" into $a3
5378:
5379:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
5380:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
5381:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
5382:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
5383:
5384:       # Make the system call to play the sound
5385:           li $v0, 31  # Use the "play note" system call
5386:           syscall     # Print out the output on screen
5387:
5388:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5389:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
5390:       syscall   # Print out the output on screen
5391:
5392:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5393:       syscall     # Print out the output on screen
5394:       move $s1, $v0   # move value held in $v0 to register $s1
5395:
5396:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5397:       syscall     # Print out the output on screen
5398:       move $s2, $v0   # move value held in $v0 to register $s2
5399:
5400:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5401:       syscall     # Print out the output on screen
5402:       move $s6, $v0   # move value held in $v0 to register $s6
5403:
5404:       jal FindQ6Num1      # Jump to FindQ6Num1 and perform
5405:       lw $s4, valueX1_Q6  # Load value held in valueX1_Q6 to register $s4
5406:
5407:       jal FindQ6Num2      # Jump to FindQ6Num2 and perform
5408:       lw $s5, valueX2_Q6  # Load value held in valueX2_Q6 to register $s5
5409:
5410:       jal FindQ6Num3      # Jump to FindQ6Num3 and perform
5411:       lw $s7, valueX3_Q6  # Load value held in valueX3_Q6 to register $s7
5412:
5413:       beq $s1, $s4, check_y1_Q6_  # Branch if equal, if value held in register $s1 i
s equal to value in $s4 go to check_y1_Q6_
5414:       beq $s1, $s5, check_y2_Q6_  # Branch if equal, if value held in register $s1 i
s equal to value in $s5 go to check_y2_Q6_
5415:       beq $s1, $s7, check_y3_Q6_  # Branch if equal, if value held in register $s1 i
s equal to value in $s7 go to check_y3_Q6_
5416:       j _check_y1_Q6_         # Else, jump to _check_y1_Q6_
5417:
5418:   check_y1_Q6_:
5419:       beq $s2, $s5, check_z1_Q6_  # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z1_Q6_
5420:       beq $s2, $s7, check_z2_Q6_  # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z2_Q6_
5421:       j _check_z1_Q6_     # Else jump to _check_z1_Q6_
```

```
5422:
5423:   check_y2_Q6_:
5424:       beq $s2, $s4, check_z1_Q6_  # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z1_Q6_
5425:       beq $s2, $s7, check_z3_Q6_  # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z3_Q6_
5426:       j _check_z1_Q6_     # Else jump to _check_z1_Q6_
5427:
5428:   check_y3_Q6_:
5429:       beq $s2, $s4, check_z2_Q6_  # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z2_Q6_
5430:       beq $s2, $s5, check_z3_Q6_  # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z3_Q6_
5431:       j _check_z1_Q6_     # Else jump to _check_z1_Q6_
5432:
5433:   _check_y1_Q6_:
5434:       bne $s2, $s4, _check_z1_Q6_ # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s4 go to _check_z1_Q6_
5435:       bne $s2, $s5, _check_z1_Q6_ # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s5 go to _check_z1_Q6_
5436:       bne $s2, $s7, _check_z1_Q6_ # Branch if not equal, if value held in register $
s2 is NOT equal to value in $s7 go to _check_z1_Q6_
5437:       j check_z1_Q6_      # Else jump to check_z1_Q6_
5438:
5439:   check_z1_Q6_:
5440:       beq $s6, $s7, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s7 go to Correct_Q6
5441:       j lost_Q6       # Else jump to lost_Q6
5442:
5443:   check_z2_Q6_:
5444:       beq $s6, $s5, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s5 go to Correct_Q6
5445:       j lost_Q6       # Else jump to lost_Q6
5446:
5447:   check_z3_Q6_:
5448:       beq $s6, $s4, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s4 go to Correct_Q6
5449:       j lost_Q6       # Else jump to lost_Q6
5450:
5451:   _check_z1_Q6_:
5452:       bne $s6, $s4, lost_Q6   # Branch if not equal, if value held in register $s6 i
s NOT equal to value in $s4 go to lost_Q6
5453:       bne $s6, $s5, lost_Q6   # Branch if not equal, if value held in register $s6 i
s NOT equal to value in $s5 go to lost_Q6
5454:       bne $s6, $s7, lost_Q6   # Branch if not equal, if value held in register $s6 i
s NOT equal to value in $s7 go to lost_Q6
5455:       j lost_Q6       # Else jump to lost_Q6
5456:
5457:
5458:   secondVersion_Q6:
5459:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5460:       la $a0, secondVer6.1  # Load secondVer6.1 into $a0 using la (load Address)
5461:       syscall   # Print out the output on screen
5462:
```

```
5463:       # Printing value 6 from ArrayQ6Int at ArrayQ6Int[0]
5464:       lw $t5, ArrayQ6Int($zero)    # Loading value at index 0 in 1D ArrayQ6Int Quest
ion 6 IF WE WANT TO PRINT IT by using lw (load word)
5465:
5466:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5467:       addi $a0, $t5, 0     # Add value held in register $t5 to register $a0 in order
 to print out on screen
5468:       syscall     # Print out the output on screen
5469:
5470:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5471:       la $a0, secondVer6.2   # Load secondVer6.2 into $a0 using la (load Address)
5472:       syscall   # Print out the output on screen
5473:
5474:       # Printing value 11 from ArrayQ6Int at ArrayQ6Int[1]
5475:       li $t9, 4   # Loading register $t9 with the index 4
5476:           lw $t6, ArrayQ6Int($t9)     # Loading value at index 4 in 1D ArrayQ6Int Qu
estion 6 IF WE WANT TO PRINT IT by using lw (load word)
5477:
5478:           li $v0, 1   # Telling the system to print an INTEGER by putting value 1 in
to register $v0
5479:           move $a0, $t6   # Moving value held in register $t6 to register $a0 in ord
er to print out on screen
5480:       syscall     # Print out the output on screen
5481:
5482:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5483:       la $a0, secondVer6.3   # Load secondVer6.3 into $a0 using la (load Address)
5484:       syscall   # Print out the output on screen
5485:
5486:       # Printing value 6 from ArrayQ6Int at ArrayQ6Int[2]
5487:       addi $t9, $t9, 4    # Changing register $t9 to index 8 by adding 4 to its inde
x since it is at index 4
5488:       lw $t7, ArrayQ6Int($t9)     # Loading value at index 8 in 1D ArrayQ6Int Questi
on 6 IF WE WANT TO PRINT IT by using lw (load word)
5489:
5490:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5491:       move $a0, $t7   # Moving value held in register $t7 to register $a0 in order t
o print out on screen
5492:       syscall     # Print out the output on screen
5493:
5494:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5495:       la $a0, firstVer6.4   # Load firstVer6.3 into $a0 using la (load Address)
5496:       syscall   # Print out the output on screen
5497:
5498:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5499:       la $a0, roots   # Load roots into $a0 using la (load Address)
5500:       syscall   # Print out the output on screen
5501:
5502:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
```

```
5503:       syscall     # Print out the output on screen
5504:       move $s1, $v0   # move value held in $v0 to register $s1
5505:
5506:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5507:       syscall     # Print out the output on screen
5508:       move $s2, $v0   # move value held in $v0 to register $s2
5509:
5510:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5511:       syscall     # Print out the output on screen
5512:       move $s6, $v0   # move value held in $v0 to register $s6
5513:
5514:       jal FindQ6Num1_     # Jump to FindQ6Num1 and perform
5515:       lw $s4, valueX1_Q6_ # Load value held in valueX1_Q6_ to register $s4
5516:
5517:       jal FindQ6Num2_     # Jump to FindQ6Num2 and perform
5518:       lw $s5, valueX2_Q6_ # Load value held in valueX2_Q6_ to register $s5
5519:
5520:       jal FindQ6Num3_     # Jump to FindQ6Num3 and perform
5521:       lw $s7, valueX3_Q6_ # Load value held in valueX3_Q6_ to register $s7
5522:
5523:       beq $s1, $s4, check_y1_Q6__ # Branch if equal, if value held in register $s1 i
s NOT equal to value in $s4 go to check_y1_Q6__
5524:       beq $s1, $s5, check_y2_Q6__ # Branch if equal, if value held in register $s1 i
s NOT equal to value in $s5 go to check_y2_Q6__
5525:       beq $s1, $s7, check_y3_Q6__ # Branch if equal, if value held in register $s1 i
s NOT equal to value in $s7 go to check_y3_Q6__
5526:       j _check_y1_Q6__        # Else, jump to _check_y1_Q6__
5527:
5528:   check_y1_Q6__:
5529:       beq $s2, $s5, check_z1_Q6__ # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z1_Q6__
5530:       beq $s2, $s7, check_z2_Q6__ # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z2_Q6__
5531:       j _check_z1_Q6__        # Else jump to _check_z1_Q6__
5532:
5533:   check_y2_Q6__:
5534:       beq $s2, $s4, check_z1_Q6__ # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z1_Q6__
5535:       beq $s2, $s7, check_z3_Q6__ # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z3_Q6__
5536:       j _check_z1_Q6__        # Else jump to _check_z1_Q6__
5537:
5538:   check_y3_Q6__:
5539:       beq $s2, $s4, check_z2_Q6__ # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z2_Q6__
5540:       beq $s2, $s5, check_z3_Q6__ # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z3_Q6__
5541:       j _check_z1_Q6__        # Else jump to _check_z1_Q6__
5542:
5543:   _check_y1_Q6__:
5544:       bne $s2, $s4, _check_z1_Q6__    # Branch not if equal, if value held in regist
er $s2 is NOT equal to value in $s4 go to _check_z1_Q6__
5545:       bne $s2, $s5, _check_z1_Q6__    # Branch not if equal, if value held in regist
```

```
er $s2 is NOT equal to value in $s5 go to _check_z1_Q6__
5546:       bne $s2, $s7, _check_z1_Q6__    # Branch not if equal, if value held in regist
er $s2 is NOT equal to value in $s5 go to _check_z1_Q6__
5547:       j check_z1_Q6__     # Else jump to check_z1_Q6__
5548:
5549:   check_z1_Q6__:
5550:       beq $s6, $s7, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s7 go to Correct_Q6
5551:       j incorrect_Q6_     # Else jump to incorrect_Q6_
5552:
5553:   check_z2_Q6__:
5554:       beq $s6, $s5, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s5 go to Correct_Q6
5555:       j incorrect_Q6_     # Else jump to incorrect_Q6_
5556:
5557:   check_z3_Q6__:
5558:       beq $s6, $s4, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s4 go to Correct_Q6
5559:       j incorrect_Q6_     # Else jump to incorrect_Q6_
5560:
5561:   _check_z1_Q6__:
5562:       bne $s6, $s4, incorrect_Q6_ # Branch if not equal, if value held in register $
s6 is NOT equal to value in $s4 go to incorrect_Q6_
5563:       bne $s6, $s5, incorrect_Q6_ # Branch if not equal, if value held in register $
s6 is NOT equal to value in $s5 go to incorrect_Q6_
5564:       bne $s6, $s7, incorrect_Q6_ # Branch if not equal, if value held in register $
s6 is NOT equal to value in $s7 go to incorrect_Q6_
5565:       j incorrect_Q6_     # Else jump to incorrect_Q6_
5566:
5567:
5568:   incorrect_Q6_:
5569:       la $a0, beep6       # Load the address of "beep6" into $a0
5570:       la $a1, duration6   # Load the address of "duration6" into $a1
5571:       la $a2, instrument6 # Load the address of "instrument6" into $a2
5572:       la $a3, volume6     # Load the address of "volume6" into $a3
5573:
5574:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
5575:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
5576:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
5577:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
5578:
5579:       # Make the system call to play the sound
5580:           li $v0, 31  # Use the "play note" system call
5581:           syscall     # Print out the output on screen
5582:
5583:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5584:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
5585:       syscall   # Print out the output on screen
5586:
5587:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5588:       syscall     # Print out the output on screen
5589:       move $s1, $v0   # move value held in $v0 to register $s1
5590:
```

```
5591:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5592:       syscall     # Print out the output on screen
5593:       move $s2, $v0   # move value held in $v0 to register $s2
5594:
5595:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5596:       syscall     # Print out the output on screen
5597:       move $s6, $v0   # move value held in $v0 to register $s6
5598:
5599:       jal FindQ6Num1      # Jump to FindQ6Num1 and perform
5600:       lw $s4, valueX1_Q6_ # Load value held in valueX1_Q6 to register $s4
5601:
5602:       jal FindQ6Num2      # Jump to FindQ6Num2 and perform
5603:       lw $s5, valueX2_Q6_ # Load value held in valueX2_Q6 to register $s5
5604:
5605:       jal FindQ6Num3      # Jump to FindQ6Num3 and perform
5606:       lw $s7, valueX3_Q6_ # Load value held in valueX3_Q6 to register $s7
5607:
5608:       beq $s1, $s4, check_y1__Q6_ # Branch if equal, if value held in register $s1 i
s equal to value in $s4 go to check_y1_Q6_
5609:       beq $s1, $s5, check_y2__Q6_ # Branch if equal, if value held in register $s1 i
s equal to value in $s5 go to check_y2_Q6_
5610:       beq $s1, $s7, check_y3__Q6_ # Branch if equal, if value held in register $s1 i
s equal to value in $s7 go to check_y3_Q6_
5611:       j _check_y1__Q6_        # Else, jump to _check_y1_Q6_
5612:
5613:   check_y1__Q6_:
5614:       beq $s2, $s5, check_z1__Q6_ # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z1_Q6_
5615:       beq $s2, $s7, check_z2__Q6_ # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z2_Q6_
5616:       j _check_z1__Q6_        # Else jump to _check_z1_Q6_
5617:
5618:   check_y2__Q6_:
5619:       beq $s2, $s4, check_z1__Q6_ # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z1_Q6_
5620:       beq $s2, $s7, check_z3__Q6_ # Branch if equal, if value held in register $s2 i
s equal to value in $s7 go to check_z3_Q6_
5621:       j _check_z1__Q6_        # Else jump to _check_z1_Q6_
5622:
5623:   check_y3__Q6_:
5624:       beq $s2, $s4, check_z2__Q6_ # Branch if equal, if value held in register $s2 i
s equal to value in $s4 go to check_z2_Q6_
5625:       beq $s2, $s5, check_z3__Q6_ # Branch if equal, if value held in register $s2 i
s equal to value in $s5 go to check_z3_Q6_
5626:       j _check_z1__Q6_        # Else jump to _check_z1_Q6_
5627:
5628:   _check_y1__Q6_:
5629:       bne $s2, $s4, _check_z1__Q6_    # Branch if not equal, if value held in regist
er $s2 is NOT equal to value in $s4 go to _check_z1_Q6_
5630:       bne $s2, $s5, _check_z1__Q6_    # Branch if not equal, if value held in regist
er $s2 is NOT equal to value in $s5 go to _check_z1_Q6_
5631:       bne $s2, $s7, _check_z1__Q6_    # Branch if not equal, if value held in regist
er $s2 is NOT equal to value in $s7 go to _check_z1_Q6_
```

```
5632:       j check_z1__Q6_     # Else jump to check_z1_Q6_
5633:
5634:   check_z1__Q6_:
5635:       beq $s6, $s7, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s7 go to Correct_Q6
5636:       j lost_Q6       # Else jump to lost_Q6
5637:
5638:   check_z2__Q6_:
5639:       beq $s6, $s5, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s5 go to Correct_Q6
5640:       j lost_Q6       # Else jump to lost_Q6
5641:
5642:   check_z3__Q6_:
5643:       beq $s6, $s4, Correct_Q6    # Branch if equal, if value held in register $s6 i
s equal to value in $s4 go to Correct_Q6
5644:       j lost_Q6       # Else jump to lost_Q6
5645:
5646:   _check_z1__Q6_:
5647:       bne $s6, $s4, lost_Q6   # Branch if not equal, if value held in register $s6 i
s NOT equal to value in $s4 go to lost_Q6
5648:       bne $s6, $s5, lost_Q6   # Branch if not equal, if value held in register $s6 i
s NOT equal to value in $s5 go to lost_Q6
5649:       bne $s6, $s7, lost_Q6   # Branch if not equal, if value held in register $s6 i
s NOT equal to value in $s7 go to lost_Q6
5650:       j lost_Q6       # Else jump to lost_Q6
5651:
5652:
5653:   lost_Q6:
5654:       la $a0, beep6       # Load the address of "beep6" into $a0
5655:       la $a1, duration6   # Load the address of "duration6" into $a1
5656:       la $a2, instrument6 # Load the address of "instrument6" into $a2
5657:       la $a3, volume6     # Load the address of "volume6" into $a3
5658:
5659:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
5660:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
5661:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
5662:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
5663:
5664:       # Make the system call to play the sound
5665:           li $v0, 31  # Use the "play note" system call
5666:           syscall     # Print out the output on screen
5667:
5668:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5669:       la $a0, lost   # Load lost into $a0 using la (load Address)
5670:       syscall   # Print out the output on screen
5671:
5672:       # Telling the system to stop executing the program
5673:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
5674:       syscall     # Print out the output on screen
5675:
5676:   exit_Q6:
5677:
5678:
```

```
5679:       lw   $a3, 0($sp)    # Loading the value held in register $a3 from the stack in
 the first location in the stack pointer 0 using lw (load word)
5680:       lw   $ra, 4($sp)    # Loading the nested function "FindQ5Num1" called in this
function from the stack in the second location in the stack pointer at 4 using lw (load wo
rd)
5681:       lw   $ra, 8($sp)    # Loading the nested function "FindQ5Num2" called in this
function from the stack in the second location in the stack pointer at 8 using lw (load wo
rd)
5682:       lw   $ra, 12($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 12 using lw (load w
ord)
5683:       lw   $ra, 16($sp)   # Loading the nested function "FindQ5Num2" called in this
function from the stack in the second location in the stack pointer at 16 using lw (load w
ord)
5684:       lw   $ra, 20($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 20 using lw (load w
ord)
5685:       lw   $ra, 24($sp)   # Loading the nested function "FindQ5Num1" called in this
function from the stack in the second location in the stack pointer at 24 using lw (load w
ord)
5686:       lw   $ra, 28($sp)   # Loading the nested function "FindQ5Num2" called in this
function from the stack in the second location in the stack pointer at 28 using lw (load w
ord)
5687:       lw   $ra, 32($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 32 using lw (load w
ord)
5688:       lw   $ra, 36($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 36 using lw (load w
ord)
5689:       lw   $ra, 40($sp)   # Loading the nested function "FindQ5Num2" called in this
function from the stack in the second location in the stack pointer at 40 using lw (load w
ord)
5690:       lw   $ra, 44($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 44 using lw (load w
ord)
5691:       lw   $ra, 48($sp)   # Loading the nested function "FindQ5Num3" called in this
function from the stack in the second location in the stack pointer at 48 using lw (load w
ord)
5692:       addi $sp, $sp, 52   # Alocatie memeory back to the stack for 40 bytes (negativ
e because we are allocating space from the stack, positive is when we are adding space to
the stack)
5693:
5694:       jr $ra      # Finish and return to main and continue executing
5695:
5696:
5697:   # Find first answer of question 6
5698:   FindQ6Num1:
5699:       la $t0, ArrayQ6Ans      # base address of ArrayQ6Ans (base_address = addess of
 MapPoints)
5700:       li $t1, 0       # row index of value (i = 0)
5701:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5702:       li $t3, 0       # column index of value (j = 0)
5703:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5704:
5705:       # Calculate the address of value using the formula
```

```
5706:       mul $t5, $t1, $t2   # (i * m)
5707:       add $t5, $t5, $t3   # (i * m + j)
5708:       sll $t5, $t5, 2     # (i * m + j) * element_size
5709:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5710:
5711:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5712:
5713:       sw $t6, valueX1_Q6      # Store value held in register $t6 in valueX1_Q6 since
 it is the answer
5714:
5715:       jr $ra
5716:
5717:
5718:   # Find second answer of question 6
5719:   FindQ6Num2:
5720:       la $t0, ArrayQ6Ans      # base address of ArrayQ6Ans (base_address = addess of
 MapPoints)
5721:       li $t1, 0       # row index of value (i = 0)
5722:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5723:       li $t3, 1       # column index of value (j = 1)
5724:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5725:
5726:       # Calculate the address of value using the formula
5727:       mul $t5, $t1, $t2   # (i * m)
5728:       add $t5, $t5, $t3   # (i * m + j)
5729:       sll $t5, $t5, 2     # (i * m + j) * element_size
5730:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5731:
5732:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5733:
5734:       sw $t6, valueX2_Q6      # Store value held in register $t6 in valueX1_Q6 since
 it is the answer
5735:
5736:       jr $ra      # Finish and return to main and continue executing
5737:
5738:
5739:   # Find third answer of question 6
5740:   FindQ6Num3:
5741:       la $t0, ArrayQ6Ans      # base address of ArrayQ6Ans (base_address = addess of
 MapPoints)
5742:       li $t1, 0       # row index of value (i = 0)
5743:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5744:       li $t3, 2       # column index of value (j = 2)
5745:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5746:
5747:       # Calculate the address of value using the formula
5748:       mul $t5, $t1, $t2   # (i * m)
5749:       add $t5, $t5, $t3   # (i * m + j)
5750:       sll $t5, $t5, 2     # (i * m + j) * element_size
5751:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5752:
5753:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5754:
5755:       sw $t6, valueX3_Q6      # Store value held in register $t6 in valueX3_Q6 since
 it is the answer
```

```
5756:
5757:       jr $ra      # Finish and return to main and continue executing
5758:
5759:
5760:   # Find first answer of question 6
5761:   FindQ6Num1_:
5762:       la $t0, ArrayQ6Ans      # base address of ArrayQ6Ans (base_address = addess of
 MapPoints)
5763:       li $t1, 1       # row index of value (i = 1)
5764:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5765:       li $t3, 0       # column index of value (j = 0)
5766:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5767:
5768:       # Calculate the address of value using the formula
5769:       mul $t5, $t1, $t2   # (i * m)
5770:       add $t5, $t5, $t3   # (i * m + j)
5771:       sll $t5, $t5, 2     # (i * m + j) * element_size
5772:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5773:
5774:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5775:
5776:       sw $t6, valueX1_Q6_     # Store value held in register $t6 in valueX1_Q6_ sinc
e it is the answer
5777:
5778:       jr $ra
5779:
5780:
5781:   # Find second answer of question 6
5782:   FindQ6Num2_:
5783:       la $t0, ArrayQ6Ans      # base address of ArrayQ6Ans (base_address = addess of
 MapPoints)
5784:       li $t1, 1       # row index of value (i = 1)
5785:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5786:       li $t3, 1       # column index of value (j = 1)
5787:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5788:
5789:       # Calculate the address of value using the formula
5790:       mul $t5, $t1, $t2   # (i * m)
5791:       add $t5, $t5, $t3   # (i * m + j)
5792:       sll $t5, $t5, 2     # (i * m + j) * element_size
5793:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5794:
5795:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5796:
5797:       sw $t6, valueX2_Q6_     # Store value held in register $t6 in valueX1_Q6_ sinc
e it is the answer
5798:
5799:       jr $ra      # Finish and return to main and continue executing
5800:
5801:
5802:   # Find third answer of question 6
5803:   FindQ6Num3_:
5804:       la $t0, ArrayQ6Ans      # base address of ArrayQ6Ans (base_address = addess of
 MapPoints)
5805:       li $t1, 1       # row index of value (i = 1)
```

```
5806:       li $t2, 3       # number of columns in array MapPoints (m = 3)
5807:       li $t3, 2       # column index of value (j = 2)
5808:       li $t4, 4       # element size in bytes (4 bytes) (element_size = 4)
5809:
5810:       # Calculate the address of value using the formula
5811:       mul $t5, $t1, $t2   # (i * m)
5812:       add $t5, $t5, $t3   # (i * m + j)
5813:       sll $t5, $t5, 2     # (i * m + j) * element_size
5814:       add $t5, $t5, $t0   # base_address + (i * m + j) * element_size
5815:
5816:       lw $t6, 0($t5)      # load value held in register $t5 into $t6
5817:
5818:       sw $t6, valueX3_Q6_     # Store value held in register $t6 in valueX3_Q6_ sinc
e it is the answer
5819:
5820:       jr $ra      # Finish and return to main and continue executing
5821:
5822:
5823:   # Pseudocode:
5824:   # void PolynomialQuestionSeven(const int* ArrayQ7Int, const int* ArrayQ7Ans1){
5825:   #   print(Question 7)
5826:   #   while(true){
5827:   #   continue;
5828:   # }
5829:   #   while(false){
5830:   #   exit(0);
5831:   # }
5832:   # Question 7 function
5833:   PolynomialQuestionSeven:
5834:       # Printing Question 7 to the player
5835:       li $v0, 4       # Telling the system to print a TEXT by putting value 4 into r
egister $v0
5836:       la $a0, Question7       # Load Question7 into $a0 using la (load Address)
5837:       syscall     # Print out the output on screen
5838:
5839:       # Printing value 3 from ArrayQ6Int at ArrayQ7Int[0]
5840:       lw $s2, ArrayQ7Int($zero)    # Loading value held at index 0 in ArrayQ7Int Que
stion 7 to register $s2
5841:
5842:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5843:       addi $a0, $s2, 0     # Add value held in register $s2 to register $a0 in order
 to print out on screen
5844:       syscall     # Print out the output on screen
5845:
5846:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5847:       la $a0, firstVer7.1  # Load firstVer7.1 into $a0 using la (load Address)
5848:       syscall   # Print out the output on screen
5849:
5850:       # Printing value 39 from ArrayQ7Int at ArrayQ7Int[1]
5851:       li $t0, 4       # Loading register $t0 with the index 4
5852:       lw $s4, ArrayQ7Int($t0)    # Loading value held at index 4 in ArrayQ7Int Quest
ion 7 to register $s4
5853:
```

```
5854:           li $v0, 1   # Telling the system to print an INTEGER by putting value 1 in
to register $v0
5855:           move $a0, $s4   # Moving value held in register $s4 to register $a0 in ord
er to print out on screen
5856:       syscall     # Print out the output on screen
5857:
5858:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5859:       la $a0, firstVer7.2   # Load firstVer7.2 into $a0 using la (load Address)
5860:       syscall   # Print out the output on screen
5861:
5862:       # Printing value 168 from ArrayQ7Int at ArrayQ7Int[2]
5863:       addi $t0, $t0, 4    # Updating register $t0 to index 8 by adding 4 to it
5864:       lw $s5, ArrayQ7Int($t0)    # Loading value held at index 8 in ArrayQ7Int Quest
ion 7 to register $s5
5865:
5866:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5867:       move $a0, $s5   # Moving value held in register $s5 to register $a0 in order t
o print out on screen
5868:       syscall     # Print out the output on screen
5869:
5870:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5871:       la $a0, firstVer7.3   # Load firstVer7.3 into $a0 using la (load Address)
5872:       syscall   # Print out the output on screen
5873:
5874:       # Printing value 276 from ArrayQ7Int at ArrayQ7Int[3]
5875:       addi $t0, $t0, 4    # Updating register $t0 to index 12 by adding 4 to it
5876:       lw $s6, ArrayQ7Int($t0)    # Loading value held at index 12 in ArrayQ7Int Ques
tion 7 to register $s6
5877:
5878:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5879:       move $a0, $s6   # Moving value held in register $s6 to register $a0 in order t
o print out on screen
5880:       syscall     # Print out the output on screen
5881:
5882:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5883:       la $a0, firstVer7.4   # Load firstVer7.4 into $a0 using la (load Address)
5884:       syscall   # Print out the output on screen
5885:
5886:       addi $t0, $t0, 4    # Updating register $t0 to index 16 by adding 4 to it
5887:       lw $s7, ArrayQ7Int($t0)    # Loading value held at index 16 in ArrayQ7Int Ques
tion 7 to register $s7
5888:
5889:       li $v0, 1   # Telling the system to print an INTEGER by putting value 1 into r
egister $v0
5890:       move $a0, $s7   # Moving value held in register $s7 to register $a0 in order t
o print out on screen
5891:       syscall     # Print out the output on screen
5892:
5893:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
```

```
5894:       la $a0, firstVer7.5   # Load firstVer7.5 into $a0 using la (load Address)
5895:       syscall   # Print out the output on screen
5896:
5897:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
5898:       la $a0, roots   # Load roots into $a0 using la (load Address)
5899:       syscall   # Print out the output on screen
5900:
5901:
5902:                 li $v0, 5 # Telling the system to get an INTEGER from the user by pu
tting value 5 into register $v0 (scanf)
5903:       syscall     # Print out the output on screen
5904:       move $t1, $v0   # move value held in $v0 to register $t1
5905:
5906:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5907:       syscall     # Print out the output on screen
5908:       move $t2, $v0   # move value held in $v0 to register $t2
5909:
5910:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5911:       syscall     # Print out the output on screen
5912:       move $t8, $v0   # move value held in $v0 to register $t8
5913:
5914:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
5915:       syscall     # Print out the output on screen
5916:       move $t9, $v0   # move value held in $v0 to register $t9
5917:
5918:       lw $t4, ArrayQ7Ans1($zero)    # Loading value held at index 4 in ArrayQ7Int Qu
estion 7 to register $t4
5919:       li $t3, 4       # Loading register $t3 with the index 4
5920:       lw $t5, ArrayQ7Ans1($t3)    # Loading value held at index 8 in ArrayQ7Int Ques
tion 7 to register $t5
5921:       addi $t3, $t3, 4    # Updating register $t0 to index 12 by adding 4 to it
5922:       lw $t6, ArrayQ7Ans1($t3)    # Loading value held at index 12 in ArrayQ7Int Que
stion 7 to to register $t6
5923:       addi $t3, $t3, 4    # Updating register $t0 to index 16 by adding 4 to it
5924:       lw $t7, ArrayQ7Ans1($t3)    # Loading value held at index 16 in ArrayQ7Int Que
stion 7 to register $t7
5925:
5926:       beq $t1, $t4, check_y1_Q7   # Branch if equal, if value held in register $t1 i
s equal to value in $t4 go to check_y1_Q7
5927:       beq $t1, $t5, check_y2_Q7   # Branch if equal, if value held in register $t1 i
s equal to value in $t5 go to check_y2_Q7
5928:       beq $t1, $t6, check_y3_Q7   # Branch if equal, if value held in register $t1 i
s equal to value in $t6 go to check_y3_Q7
5929:       beq $t1, $t7, check_y4_Q7   # Branch if equal, if value held in register $t1 i
s equal to value in $t7 go to check_y4_Q7
5930:       j _check_y1_Q7      # Else, jump to _check_y1_Q7
5931:
5932:   check_y1_Q7:
5933:       beq $t2, $t5, check_z1_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t5 go to check_z1_Q7
5934:       beq $t2, $t6, check_z2_Q7   # Branch if equal, if value held in register $t2 i
```

```
s equal to value in $t6 go to check_z2_Q7
5935:       beq $t2, $t7, check_z3_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t7 go to check_z3_Q7
5936:       j _check_z1_Q7      # Else jump to _check_z1_Q7
5937:
5938:   check_y2_Q7:
5939:       beq $t2, $t4, check_z1_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t4 go to check_z1_Q7
5940:       beq $t2, $t6, check_z4_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t6 go to check_z4_Q7
5941:       beq $t2, $t7, check_z5_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t7 go to check_z5_Q7
5942:       j _check_z1_Q7      # Else jump to _check_z1_Q7
5943:
5944:   check_y3_Q7:
5945:       beq $t2, $t4, check_z2_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t4 go to check_z2_Q7
5946:       beq $t2, $t5, check_z4_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t5 go to check_z4_Q7
5947:       beq $t2, $t7, check_z6_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t7 go to check_z6_Q7
5948:       j _check_z1_Q7      # Else jump to _check_z1_Q7
5949:
5950:   check_y4_Q7:
5951:       beq $t2, $t4, check_z3_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t4 go to check_z3_Q7
5952:       beq $t2, $t5, check_z5_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t5 go to check_z5_Q7
5953:       beq $t2, $t6, check_z6_Q7   # Branch if equal, if value held in register $t2 i
s equal to value in $t6 go to check_z6_Q7
5954:       j _check_z1_Q7      # Else jump to _check_z1_Q7
5955:
5956:   _check_y1_Q7:
5957:       bne $t2, $t4, _check_z1_Q7  # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t4 go to _check_z1_Q7
5958:       bne $t2, $t5, _check_z1_Q7  # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t5 go to _check_z1_Q7
5959:       bne $t2, $t6, _check_z1_Q7  # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t6 go to _check_z1_Q7
5960:       bne $t2, $t7, _check_z1_Q7  # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t7 go to _check_z1_Q7
5961:       j check_z1_Q7       # Else jump to check_z1_Q7
5962:
5963:   check_z1_Q7:
5964:       beq $t8, $t6, check_w4_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t6 go to check_w4_Q7
5965:       beq $t8, $t7, check_w3_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t7 go to check_w3_Q7
5966:       j _check_w1_Q7      # Else jump to _check_w1_Q7
5967:
5968:   check_z2_Q7:
5969:       beq $t8, $t5, check_w4_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t5 go to check_w4_Q7
5970:       beq $t8, $t7, check_w2_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t7 go to check_w2_Q7
```

```
5971:       j _check_w1_Q7      # Else jump to _check_w1_Q7
5972:
5973:   check_z3_Q7:
5974:       beq $t8, $t5, check_w3_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t5 go to check_w3_Q7
5975:       beq $t8, $t6, check_w2_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t6 go to check_w2_Q7
5976:       j _check_w1_Q7      # Else jump to _check_w1_Q7
5977:
5978:   check_z4_Q7:
5979:       beq $t8, $t4, check_w4_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t4 go to check_w4_Q7
5980:       beq $t8, $t7, check_w1_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t7 go to check_w1_Q7
5981:       j _check_w1_Q7      # Else jump to _check_w1_Q7
5982:
5983:   check_z5_Q7:
5984:       beq $t8, $t4, check_w3_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t4 go to check_w3_Q7
5985:       beq $t8, $t6, check_w1_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t6 go to check_w1_Q7
5986:       j _check_w1_Q7      # Else jump to _check_w1_Q7
5987:
5988:   check_z6_Q7:
5989:       beq $t8, $t4, check_w2_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t4 go to check_w2_Q7
5990:       beq $t8, $t5, check_w1_Q7   # Branch if equal, if value held in register $t8 i
s equal to value in $t5 go to check_w1_Q7
5991:       j _check_w1_Q7      # Else jump to _check_w1_Q7
5992:
5993:   _check_z1_Q7:
5994:       bne $t8, $t4, _check_w1_Q7  # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t4 go to _check_w1_Q7
5995:       bne $t8, $t5, _check_w1_Q7  # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t5 go to _check_w1_Q7
5996:       bne $t8, $t6, _check_w1_Q7  # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t6 go to _check_w1_Q7
5997:       bne $t8, $t7, _check_w1_Q7  # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t7 go to _check_w1_Q7
5998:       j check_w1_Q7   # Else jump to check_w1_Q7
5999:
6000:   check_w1_Q7:
6001:       beq $t9, $t4, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t4 go to Correct_Q7
6002:       j incorrect_Q7      # Else jump to incorrect_Q7
6003:
6004:   check_w2_Q7:
6005:       beq $t9, $t5, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t5 go to Correct_Q7
6006:       j incorrect_Q7      # Else jump to incorrect_Q7
6007:
6008:   check_w3_Q7:
6009:       beq $t9, $t6, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t6 go to Correct_Q7
6010:       j incorrect_Q7      # Else jump to incorrect_Q7
```

```
6011:
6012:   check_w4_Q7:
6013:       beq $t9, $t7, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t7 go to Correct_Q7
6014:       j incorrect_Q7      # Else jump to incorrect_Q7
6015:
6016:   _check_w1_Q7:
6017:       bne $t9, $t4, incorrect_Q7  # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t4 go to incorrect_Q7
6018:       bne $t9, $t5, incorrect_Q7  # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t5 go to incorrect_Q7
6019:       bne $t9, $t6, incorrect_Q7  # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t6 go to incorrect_Q7
6020:       bne $t9, $t7, incorrect_Q7  # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t7 go to incorrect_Q7
6021:       j incorrect_Q7      # Else jump to incorrect_Q7
6022:
6023:   Correct_Q7:
6024:       la $a0, sound5      # Load the address of "sound5" into $a0
6025:       la $a1, duration5   # Load the address of "duration5" into $a1
6026:       la $a2, instrument5     # Load the address of "instrument5" into $a2
6027:       la $a3, volume5     # Load the address of "volume5" into $a3
6028:
6029:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
6030:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
6031:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
6032:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
6033:
6034:       # Make the system call to play the sound
6035:           li $v0, 31   # Use the "play note" system call
6036:           syscall     # Print out the output on screen
6037:
6038:       li $v0, 32  # Load value 32 to register $v0 to wait for a few seconds
6039:       li $a0, 300 # wait for 300 millisecond
6040:       syscall     # Print out the output on screen
6041:
6042:       la $a0, sound5      # Load the address of "sound5" into $a0
6043:       la $a1, duration5   # Load the address of "duration5" into $a1
6044:       la $a2, instrument5     # Load the address of "instrument5" into $a2
6045:       la $a3, volume5     # Load the address of "volume5" into $a3
6046:
6047:       lb $a0, 0($a0)      # Load the value of "sound5" into $a0
6048:       lb $a1, 0($a1)      # Load the value of "duration5" into $a1
6049:       lb $a2, 0($a2)      # Load the value of "instrument5" into $a2
6050:       lb $a3, 0($a3)      # Load the value of "volume5" into $a3
6051:
6052:       # Make the system call to play the sound
6053:           li $v0, 31   # Use the "play note" system call
6054:           syscall     # Print out the output on screen
6055:
6056:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
6057:       la $a0, correct   # Load correct into $a0 using la (load Address)
6058:       syscall   # Print out the output on screen
6059:
```

```
6060:       j exit_Q7   # Jump go to exit_Q7
6061:
6062:   incorrect_Q7:
6063:       la $a0, beep6       # Load the address of "beep6" into $a0
6064:       la $a1, duration6   # Load the address of "duration6" into $a1
6065:       la $a2, instrument6 # Load the address of "instrument6" into $a2
6066:       la $a3, volume6     # Load the address of "volume6" into $a3
6067:
6068:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
6069:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
6070:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
6071:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
6072:
6073:       # Make the system call to play the sound
6074:           li $v0, 31  # Use the "play note" system call
6075:           syscall     # Print out the output on screen
6076:
6077:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
6078:       la $a0, incorrect   # Load incorrect into $a0 using la (load Address)
6079:       syscall   # Print out the output on screen
6080:
6081:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
6082:       syscall     # Print out the output on screen
6083:       move $t1, $v0   # move value held in $v0 to register $t1
6084:
6085:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
6086:       syscall     # Print out the output on screen
6087:       move $t2, $v0   # move value held in $v0 to register $t2
6088:
6089:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
6090:       syscall     # Print out the output on screen
6091:       move $t8, $v0   # move value held in $v0 to register $t8
6092:
6093:       li $v0, 5   # Telling the system to get an INTEGER from the user by putting va
lue 5 into register $v0 (scanf)
6094:       syscall     # Print out the output on screen
6095:       move $t9, $v0   # move value held in $v0 to register $t9
6096:
6097:       lw $t4, ArrayQ7Ans1($zero)    # Loading value held at index 4 in ArrayQ7Int Qu
estion 7 to register $t4
6098:       li $t3, 4       # Loading register $t3 with the index 4
6099:       lw $t5, ArrayQ7Ans1($t3)    # Loading value held at index 8 in ArrayQ7Int Ques
tion 7 to register $t5
6100:       addi $t3, $t3, 4    # Updating register $t0 to index 12 by adding 4 to it
6101:       lw $t6, ArrayQ7Ans1($t3)    # Loading value held at index 12 in ArrayQ7Int Que
stion 7 to to register $t6
6102:       addi $t3, $t3, 4    # Updating register $t0 to index 16 by adding 4 to it
6103:       lw $t7, ArrayQ7Ans1($t3)    # Loading value held at index 16 in ArrayQ7Int Que
stion 7 to register $t7
6104:
6105:       beq $t1, $t4, check_y1_Q7_  # Branch if equal, if value held in register $t1 i
```

```
s equal to value in $t4 go to check_y1_Q7_
6106:       beq $t1, $t5, check_y2_Q7_  # Branch if equal, if value held in register $t1 i
s equal to value in $t5 go to check_y2_Q7_
6107:       beq $t1, $t6, check_y3_Q7_  # Branch if equal, if value held in register $t1 i
s equal to value in $t6 go to check_y3_Q7_
6108:       beq $t1, $t7, check_y4_Q7_  # Branch if equal, if value held in register $t1 i
s equal to value in $t7 go to check_y4_Q7_
6109:       j _check_y1_Q7_         # Else, jump to _check_y1_Q7_
6110:
6111:   check_y1_Q7_:
6112:       beq $t2, $t5, check_z1_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t5 go to check_z1_Q7_
6113:       beq $t2, $t6, check_z2_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t6 go to check_z2_Q7_
6114:       beq $t2, $t7, check_z3_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t7 go to check_z3_Q7_
6115:       j _check_z1_Q7_     # Else jump to _check_z1_Q7_
6116:
6117:   check_y2_Q7_:
6118:       beq $t2, $t4, check_z1_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t4 go to check_z1_Q7_
6119:       beq $t2, $t6, check_z4_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t6 go to check_z4_Q7_
6120:       beq $t2, $t7, check_z5_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t7 go to check_z5_Q7_
6121:       j _check_z1_Q7_     # Else jump to _check_z1_Q7_
6122:
6123:   check_y3_Q7_:
6124:       beq $t2, $t4, check_z2_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t4 go to check_z2_Q7_
6125:       beq $t2, $t5, check_z4_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t5 go to check_z4_Q7_
6126:       beq $t2, $t7, check_z6_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t7 go to check_z6_Q7_
6127:       j _check_z1_Q7_     # Else jump to _check_z1_Q7_
6128:
6129:   check_y4_Q7_:
6130:       beq $t2, $t4, check_z3_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t4 go to check_z3_Q7_
6131:       beq $t2, $t5, check_z5_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t5 go to check_z5_Q7_
6132:       beq $t2, $t6, check_z6_Q7_  # Branch if equal, if value held in register $t2 i
s equal to value in $t6 go to check_z6_Q7_
6133:       j _check_z1_Q7_     # Else jump to _check_z1_Q7_
6134:
6135:   _check_y1_Q7_:
6136:       bne $t2, $t4, _check_z1_Q7_ # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t4 go to _check_z1_Q7_
6137:       bne $t2, $t5, _check_z1_Q7_ # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t5 go to _check_z1_Q7_
6138:       bne $t2, $t6, _check_z1_Q7_ # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t6 go to _check_z1_Q7_
6139:       bne $t2, $t7, _check_z1_Q7_ # Branch not if equal, if value held in register $
t2 is NOT equal to value in $t7 go to _check_z1_Q7_
6140:       j check_z1_Q7_      # Else jump to check_z1_Q7_
```

```
6141:
6142:   check_z1_Q7_:
6143:       beq $t8, $t6, check_w4_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t6 go to check_w4_Q7_
6144:       beq $t8, $t7, check_w3_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t7 go to check_w3_Q7_
6145:       j _check_w1_Q7_     # Else jump to _check_w1_Q7_
6146:
6147:   check_z2_Q7_:
6148:       beq $t8, $t5, check_w4_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t5 go to check_w4_Q7_
6149:       beq $t8, $t7, check_w2_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t7 go to check_w2_Q7_
6150:       j _check_w1_Q7_     # Else jump to _check_w1_Q7_
6151:
6152:   check_z3_Q7_:
6153:       beq $t8, $t5, check_w3_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t5 go to check_w3_Q7
6154:       beq $t8, $t6, check_w2_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t6 go to check_w2_Q7
6155:       j _check_w1_Q7_     # Else jump to _check_w1_Q7
6156:
6157:   check_z4_Q7_:
6158:       beq $t8, $t4, check_w4_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t4 go to check_w4_Q7_
6159:       beq $t8, $t7, check_w1_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t7 go to check_w1_Q7_
6160:       j _check_w1_Q7_     # Else jump to _check_w1_Q7_
6161:
6162:   check_z5_Q7_:
6163:       beq $t8, $t4, check_w3_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t4 go to check_w3_Q7_
6164:       beq $t8, $t6, check_w1_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t6 go to check_w1_Q7_
6165:       j _check_w1_Q7_     # Else jump to _check_w1_Q7_
6166:
6167:   check_z6_Q7_:
6168:       beq $t8, $t4, check_w2_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t4 go to check_w2_Q7_
6169:       beq $t8, $t5, check_w1_Q7_  # Branch if equal, if value held in register $t8 i
s equal to value in $t5 go to check_w1_Q7_
6170:       j _check_w1_Q7_     # Else jump to _check_w1_Q7_
6171:
6172:   _check_z1_Q7_:
6173:       bne $t8, $t4, _check_w1_Q7_ # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t4 go to _check_w1_Q7_
6174:       bne $t8, $t5, _check_w1_Q7_ # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t5 go to _check_w1_Q7_
6175:       bne $t8, $t6, _check_w1_Q7_ # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t6 go to _check_w1_Q7_
6176:       bne $t8, $t7, _check_w1_Q7_ # Branch if not equal, if value held in register $
t8 is NOT equal to value in $t7 go to _check_w1_Q7_
6177:       j check_w1_Q7_      # Else jump to check_w1_Q7_
6178:
6179:   check_w1_Q7_:
```

```
6180:       beq $t9, $t4, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t4 go to Correct_Q7
6181:       j lost_Q7       # Else jump to lost_Q7
6182:
6183:   check_w2_Q7_:
6184:       beq $t9, $t5, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t5 go to Correct_Q7
6185:       j lost_Q7       # Else jump to lost_Q7
6186:
6187:   check_w3_Q7_:
6188:       beq $t9, $t6, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t6 go to Correct_Q7
6189:       j lost_Q7       # Else jump to lost_Q7
6190:
6191:   check_w4_Q7_:
6192:       beq $t9, $t7, Correct_Q7    # Branch if equal, if value held in register $t9 i
s equal to value in $t7 go to Correct_Q7
6193:       j lost_Q7       # Else jump to lost_Q7
6194:
6195:   _check_w1_Q7_:
6196:       bne $t9, $t4, lost_Q7       # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t4 go to lost_Q7
6197:       bne $t9, $t5, lost_Q7       # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t5 go to lost_Q7
6198:       bne $t9, $t6, lost_Q7       # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t6 go to lost_Q7
6199:       bne $t9, $t7, lost_Q7       # Branch if not equal, if value held in register $
t9 is NOT equal to value in $t7 go to lost_Q7
6200:       j lost_Q7       # Else jump to lost_Q7
6201:
6202:   lost_Q7:
6203:       la $a0, beep6       # Load the address of "beep6" into $a0
6204:       la $a1, duration6   # Load the address of "duration6" into $a1
6205:       la $a2, instrument6 # Load the address of "instrument6" into $a2
6206:       la $a3, volume6     # Load the address of "volume6" into $a3
6207:
6208:       lb $a0, 0($a0)      # Load the value of "beep6" into $a0
6209:       lb $a1, 0($a1)      # Load the value of "duration6" into $a1
6210:       lb $a2, 0($a2)      # Load the value of "instrument6" into $a2
6211:       lb $a3, 0($a3)      # Load the value of "volume6" into $a3
6212:
6213:       # Make the system call to play the sound
6214:           li $v0, 31  # Use the "play note" system call
6215:           syscall     # Print out the output on screen
6216:
6217:       li $v0, 4   # Telling the system to print a TEXT by putting value 4 into regis
ter $v0
6218:       la $a0, lost   # Load lost into $a0 using la (load Address)
6219:       syscall   # Print out the output on screen
6220:
6221:       # Telling the system to stop executing the program
6222:       li $v0, 10     # Telling the system to stop executing by putting value 10 into
 register $v0
6223:       syscall     # Print out the output on screen
6224:
```

```
6225:         exit_Q7:
6226:
6227:       jr $ra      # Finish and return to main and continue executing
6228:
6229:
```