VE370 Project 1

Weikai Zhou 518021911039

1. **Objectives[1]**

Develop a MIPS assembly program that operates on a data segment consisting of an array of 32-bit signed integers. In the text (program) segment of memory, write a procedure called main that implements the main() function and as well as procedures for other subroutines described below. Assemble, simulate, and carefully comment the file. Screen print the simulation results and explain the results by annotating the screen prints. We should compose an array whose size is determined by you in the main function and is not less than 30 elements.

main() {

int size = ...; //determine the size of the array here

int hotDay, coldDay, comfortDay;

int tempArray[size] = {36, 25, -6,

... //compose your own array here

};

hotDay = countArray (tempArray, size, 1);

coldDay = countArray (tempArray, size, -1);

comfortDay = countArray (tempArray, size, 0);

}

int countArray(int A[], int numElements, int cntType) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Count specific elements in the integer array A[] whose size isaaaaaaaa\*

\* numElements and return the following: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\*

\* aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\*

\* When cntType = 1, count the elements greater than or equal to 30aaaaaa\*

\* When cntType = -1, count the elements less than or equal to 5;aaaaaaaa\*

\* When cntType = 0, count the elements greater than 5 and less than 30. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int i, cnt = 0;

for(i=numElements-1, i>=0, i--) {

switch (cntType) {

case ‘1’ : cnt += hot(A[i]); break;

case ‘-1’: cnt += cold(A[i]); break;

otherwise: cnt += comfort(A[i]);

}

}

return cnt;

}

int hot(int x) {

if(x>=30) return 1;

else return 0;

}

int cold(int x) {

if (x<=5) return 1;

else return 0;

}

int comfort(int x) {

if (x>5 && x<30) return 1;

else return 0;

}

1. **Design Process**
   1. **main()**

In the main() function, we choose the size of the array to be 40, so we need to adjust the stack for 40 items and use $s0, $s1, $s2, $s3 and $s4 to represent size, hotDay, coldDay, comfortDay and tempArray[size] respectively. To initialize the array, we use $t0 as the temporary register to contain the integer and save it to the array. The array is [16, 7, 21, 6, 5, 29, -2, 37, 8, -1, 7, 11, 35, 39, 1, 30, -10, 39, -7, -15, 34, 29, 38, 6, 29, -4, 28, 29, -11, -5, 6, 19, -4, 23, -15, -1, 9, 37, -9, 24], which includes the boundary condition, e.g. 5 and 30. There are 8 hot days, 14 clod days and 18 comfortable days. Then, we use $a0, $a1 and $a2 to represent the arguments A[],numElements and cntType passed to the function countArray(int A[], int numElements, int cntType) respectively. After that, we need to recover the stack for the 40 items.

* 1. **countArray(int A[], int numElements, int cntType)**

To follow the function call convention, we first need to adjust the stack to store $a0, $a1, $a2, $s0, $s1, $s2, $s3, $s4 and $ra. Then we use $t1 and $t2 to represent i and cnt respectively. We should label the ForLoop for later jump. We should check the value of i, stored in $t1. If it is smaller than zero, we will jump to ExitForLoop, which will return the value of cnt($t2) to $v1, which stores the final answer we desired, and recover the stack we adjusted. Otherwise, we will compare cntType($a2) with $t4(representing 1) and $t5(representing -1) respectively to judge which subfunction we will jump to later.

* 1. **hot(int x)**

To follow the function call convention, we first need to adjust the stack to store $a0. Then we access the value using $t1(i) and $a0(A[]) and compare it with $t7(representing 30). If it is smaller than $t7(representing 30), we will jump to HotOtherwise. Otherwise, we will increment cnt($t2), decrement i($t1), recover stack and jump back to ForLoop. In HotOtherwise, we will decrement i($t1), recover stack and jump back to ForLoop.

* 1. **cold(int x)**

To follow the function call convention, we first need to adjust the stack to store $a0. Then we access the value using $t1(i) and $a0(A[]) and compare it with $t7(representing 5). If it is larger than $t7(representing 5), we will jump to ColdOtherwise. Otherwise, we will increment cnt($t2), decrement i($t1), recover stack and jump back to ForLoop. In ColdOtherwise, we will decrement i($t1), recover stack and jump back to ForLoop.

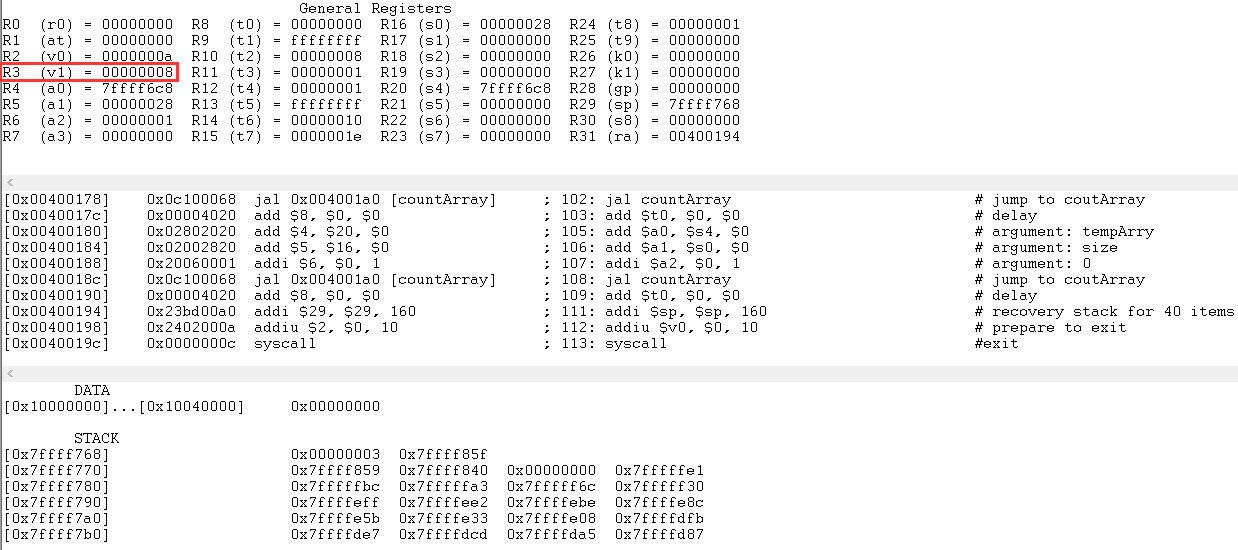
* 1. **comfort(int x)**

To follow the function call convention, we first need to adjust the stack to store $a0. Then we access the value using $t1(i) and $a0(A[]) and compare it with $t7(representing 30). If it is smaller than $t7(representing 30), we will jump to Judge. Otherwise, we will decrement i($t1), recover stack and jump back to ForLoop. In Judge, we will decrement i($t1), then compare the value with $t7(representing 5) and recover stack. If it is smaller or equal to 5, we will jump back to ForLoop. Otherwise, we will increment i($t1) and jump back to ForLoop.

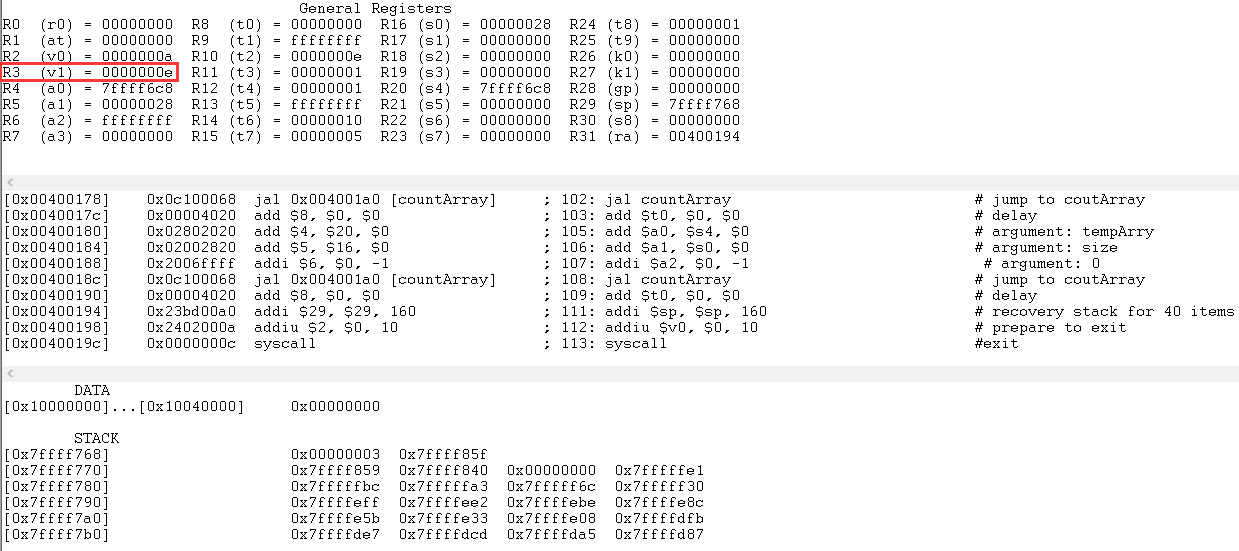
1. **Result**

The array we have is [16, 7, 21, 6, 5, 29, -2, 37, 8, -1, 7, 11, 35, 39, 1, 30, -10, 39, -7, -15, 34, 29, 38, 6, 29, -4, 28, 29, -11, -5, 6, 19, -4, 23, -15, -1, 9, 37, -9, 24], which includes the boundary condition, e.g. 5 and 30. There are 8 hot days, 14 clod days and 18 comfortable days.

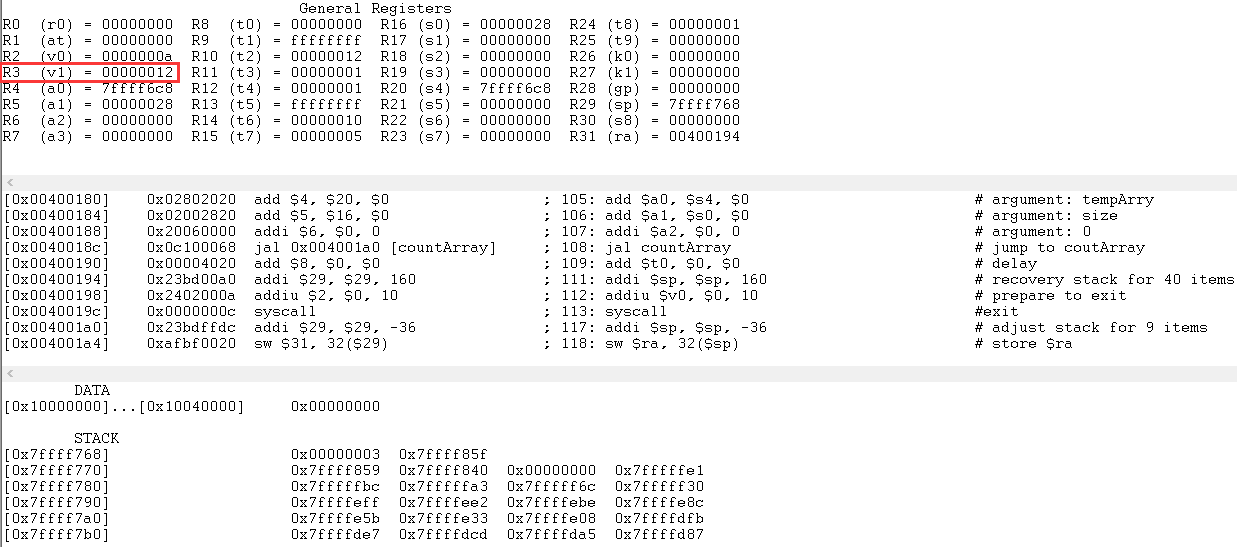
When the cntType = 1, we should count hot days, which is larger or equal to 30. And we should get 8(0x8) in $v1. The result is shown below (Figure 1).

 Figure 1. Result for cntType = 1

When the cntType = -1, we should count cold days, which is smaller or equal to 5. And we should get 14(0xE) in $v1. The result is shown below (Figure 2).

 Figure 2. Result for cntType = -1

When the cntType is neither 1 or -1, e.g. 0, we should count comfortable days, which is larger than 5 and smaller than 30. And we should get 18(0x12) in $v1. The result is shown below (Figure 3).

 Figure 3. Result for cntType = 0

From these results, we know our code runs correctly.

1. **Conclusion**

In this project, we transform the c code into MIPS to run a program. We should split it into different subfunctions and obey the function call convention. One thing I want to highlight is that when to recover the stack is important, especially there is j in the MIPS. Also, we should pay attention to the delay in MIPS. Generally, after j or jal, we should add delay. If we are not sure about the delay, we can always choose to add it.

1. **Reference**

[1] Project1.pdf

1. **Appendix**
2. .text
3. .globl \_\_start
4. \_\_start:
5. addi $sp, $sp, -160                      # adjust stack for 40 items
6. addi $s0, $0, 40                         # int size = 40
7. add $s1, $0, $0                          # int hotDay = 0
8. add $s2, $0, $0                          # int coldDay = 0
9. add $s3, $0, $0                          # int comfortDay = 0
10. add $s4, $0, $sp                         # int tempArray[size]
12. addi $t0, $0, 16
13. sw $t0, 0($s4)                           # tempArray[0] = 16
14. addi $t0, $0, 7
15. sw $t0, 4($s4)                           # tempArray[1] = 7
16. addi $t0, $0, 21
17. sw $t0, 8($s4)                           # tempArray[2] = 21
18. addi $t0, $0, 6
19. sw $t0, 12($s4)                          # tempArray[3] = 6
20. addi $t0, $0, 5
21. sw $t0, 16($s4)                          # tempArray[4] = 5
22. addi $t0, $0, 29
23. sw $t0, 20($s4)                          # tempArray[5] = 29
24. addi $t0, $0, -2
25. sw $t0, 24($s4)                          # tempArray[6] = -2
26. addi $t0, $0, 37
27. sw $t0, 28($s4)                          # tempArray[7] = 37
28. addi $t0, $0, 8
29. sw $t0, 32($s4)                          # tempArray[8] = 8
30. addi $t0, $0, -1
31. sw $t0, 36($s4)                          # tempArray[9] = -1
32. addi $t0, $0, 7
33. sw $t0, 40($s4)                          # tempArray[10] = 7
34. addi $t0, $0, 11
35. sw $t0, 44($s4)                          # tempArray[11] = 11
36. addi $t0, $0, 35
37. sw $t0, 48($s4)                          # tempArray[12] = 35
38. addi $t0, $0, 39
39. sw $t0, 52($s4)                          # tempArray[13] = 39
40. addi $t0, $0, 1
41. sw $t0, 56($s4)                          # tempArray[14] = 1
42. addi $t0, $0, 30
43. sw $t0, 60($s4)                          # tempArray[15] = 30
44. addi $t0, $0, -10
45. sw $t0, 64($s4)                          # tempArray[16] = -10
46. addi $t0, $0, 39
47. sw $t0, 68($s4)                          # tempArray[17] = 39
48. addi $t0, $0, -7
49. sw $t0, 72($s4)                          # tempArray[18] = -7
50. addi $t0, $0, -15
51. sw $t0, 76($s4)                          # tempArray[19] = -15
52. addi $t0, $0, 34
53. sw $t0, 80($s4)                          # tempArray[20] = 34
54. addi $t0, $0, 29
55. sw $t0, 84($s4)                          # tempArray[21] = 29
56. addi $t0, $0, 38
57. sw $t0, 88($s4)                          # tempArray[22] = 38
58. addi $t0, $0, 6
59. sw $t0, 92($s4)                          # tempArray[23] = 6
60. addi $t0, $0, 29
61. sw $t0, 96($s4)                          # tempArray[24] = 29
62. addi $t0, $0, -4
63. sw $t0, 100($s4)                         # tempArray[25] = -4
64. addi $t0, $0, 28
65. sw $t0, 104($s4)                         # tempArray[26] = 28
66. addi $t0, $0, 29
67. sw $t0, 108($s4)                         # tempArray[27] = 29
68. addi $t0, $0, -11
69. sw $t0, 112($s4)                         # tempArray[28] = -11
70. addi $t0, $0, -5
71. sw $t0, 116($s4)                         # tempArray[29] = -5
72. addi $t0, $0, 6
73. sw $t0, 120($s4)                         # tempArray[30] = 6
74. addi $t0, $0, 19
75. sw $t0, 124($s4)                         # tempArray[31] = 19
76. addi $t0, $0, -4
77. sw $t0, 128($s4)                         # tempArray[32] = -4
78. addi $t0, $0, 23
79. sw $t0, 132($s4)                         # tempArray[33] = 23
80. addi $t0, $0, -15
81. sw $t0, 136($s4)                         # tempArray[34] = -15
82. addi $t0, $0, -1
83. sw $t0, 140($s4)                         # tempArray[35] = -1
84. addi $t0, $0, 9
85. sw $t0, 144($s4)                         # tempArray[36] = 9
86. addi $t0, $0, 37
87. sw $t0, 148($s4)                         # tempArray[37] = 37
88. addi $t0, $0, -9
89. sw $t0, 152($s4)                         # tempArray[38] = -9
90. addi $t0, $0, 24
91. sw $t0, 156($s4)                         # tempArray[39] = 24
92. add $a0, $s4, $0                         # argument: tempArry
93. add $a1, $s0, $0                         # argument: size
94. addi $a2, $0, 1                          # argument: 1
95. jal countArray                           # jump to coutArray
96. add $t0, $0, $0                          # delay
97. add $a0, $s4, $0                         # argument: tempArry
98. add $a1, $s0, $0                         # argument: size
99. addi $a2, $0, -1                         # argument: -1
100. jal countArray                           # jump to coutArray
101. add $t0, $0, $0                          # delay
102. add $a0, $s4, $0                         # argument: tempArry
103. add $a1, $s0, $0                         # argument: size
104. addi $a2, $0, 0                          # argument: 0
105. jal countArray                           # jump to coutArray
106. add $t0, $0, $0                          # delay
107. addi $sp, $sp, 160                       # recovery stack for 40 items
108. addiu $v0, $0, 10                        # prepare to exit
109. syscall                                  # exit
110. countArray:
111. addi $sp, $sp, -36                       # adjust stack for 9 items
112. sw $ra, 32($sp)                          # store $ra
113. sw $s4, 28($sp)                          # store $s4
114. sw $s3, 24($sp)                          # store $s3
115. sw $s2, 20($sp)                          # store $s2
116. sw $s1, 16($sp)                          # store $s1
117. sw $s0, 12($sp)                          # store $s0
118. sw $a2, 8($sp)                           # store $a2
119. sw $a1, 4($sp)                           # store $a1
120. sw $a0, 0($sp)                           # store $a0
121. addi $t1, $a1, -1                        # int i = numElements - 1
122. add $t2, $0, $0                          # int cnt =0
123. addi $t4, $0, 1                          # $t4 = 1
124. addi $t5, $0, -1                         # $t5 = -1
125. ForLoop:
126. slt $t3, $t1, $0                         # if $t1(i) < 0, $t3 = 1
127. bne $t3, $0, ExitForLoop                 # if $t3 != 0, jump to exit
128. add $t0, $0, $0                          # delay
129. beq $t4, $a2, Hot                        # if $t4 == $a2 == 1, jump to Hot
130. add $t0, $0, $0                          # delay
131. beq $t5, $a2, Cold                       # if $t5 == $a2 == -1, jump to Cold
132. add $t0, $0, $0                          # delay
133. j Comfort                                # Otherwise, jump to Comfort
134. add $t0, $0, $0                          # delay
136. Hot:
137. addi $sp, $sp, -4                        # adjust stack for 1 items
138. sw $a0, 0($sp)                           # store $a0
139. sll $t6, $t1, 2                          # $t1 \* 4
140. add $t6, $t6, $a0                        # $a0 + $t1 \* 4
141. lw $t6, 0($t6)                           # load value
142. addi $t7, $0, 30
143. slt $t8, $t6, $t7                        # if $t6 < 30, $t8 = 1
144. bne $t8, $0, HotOtherwise                # if $t8 != 0, jump to HotOtherwise
145. add $t0, $0, $0                          # delay
146. addi $t2, $t2, 1                         # cnt++
147. addi $t1, $t1, -1                        # i--
148. lw $a0, 0($sp)                           # load $a0
149. addi $sp, $sp, 4                         # recover stack for 1 items
150. j ForLoop                                # jump to ForLoop
151. add $t0, $0, $0                          # delay
152. HotOtherwise:
153. addi $t2, $t2, 0                         # cnt + 0
154. addi $t1, $t1, -1                        # i--
155. lw $a0, 0($sp)                           # load $a0
156. addi $sp, $sp, 4                         # recover stack for 1 items
157. j ForLoop                                # jump to ForLoop
158. add $t0, $0, $0                          # delay
159. Cold:
160. addi $sp, $sp, -4                        # adjust stack for 1 items
161. sw $a0, 0($sp)                           # store $a0
162. sll $t6, $t1, 2                          # $t1 \* 4
163. add $t6, $t6, $a0                        # $a0 + $t1 \* 4
164. lw $t6, 0($t6)                           # load value
165. addi $t7, $0, 5                          # $t7 = 5
166. slt $t8, $t7, $t6                        # if $t7 < $t6, $t8 = 1
167. bne $t8, $0, ColdOtherwise               # if $t8 != 0, jump to ColdOtherwise
168. add $t0, $0, $0                          # delay
169. addi $t2, $t2, 1                         # cnt++
170. addi $t1, $t1, -1                        # i--
171. lw $a0, 0($sp)                           # load $a0
172. addi $sp, $sp, 4                         # recover stack for 1 items
173. j ForLoop                                # jump to ForLoop
174. add $t0, $0, $0                          # delay
175. ColdOtherwise:
176. addi $t2, $t2, 0                         # cnt + 0
177. addi $t1, $t1, -1                        # i--
178. lw $a0, 0($sp)                           # load $a0
179. addi $sp, $sp, 4                         # recover stack for 1 items
180. j ForLoop                                # jump to ForLoop
181. add $t0, $0, $0                          # delay
182. Comfort:
183. addi $sp, $sp, -4                        # adjust stack for 1 items
184. sw $a0, 0($sp)                           # store $a0
185. sll $t6, $t1, 2                          # $t1 \* 4
186. add $t6, $t6, $a0                        # $a0 + $t1 \* 4
187. lw $t6, 0($t6)                           # load value
188. addi $t7, $0, 30                         # $t7 = 30
189. slt $t8, $t6, $t7                        # if $t6 < 30, $t8 = 1
190. bne $t8, $0, Judge                       # if $t8 != 0, jump to Judge
191. add $t0, $0, $0                          # delay
192. addi $t2, $t2, 0                         # cnt + 0
193. addi $t1, $t1, -1                        # i--
194. lw $a0, 0($sp)                           # load $a0
195. addi $sp, $sp, 4                         # recover stack for 1 items
196. j ForLoop                                # jump to ForLoop
197. add $t0, $0, $0                          # delay
198. Judge:
199. addi $t1, $t1, -1                        # i--
200. addi $t7, $0, 5                          # $t7 = 5
201. slt $t8, $t7, $t6                        # if $t7(5) < $t6, $t8 = 1
202. lw $a0, 0($sp)                           # load $a0
203. addi $sp, $sp, 4                         # recover stack for 1 items
204. beq $t8, $0, ForLoop                     # if $t8 == 0 (x <= 5), jump to ForLoop
205. add $t0, $0, $0                          # delay
206. addi $t2, $t2, 1                         # cnt++
207. j ForLoop                                # jump to ForLoop
208. add $t0, $0, $0                          #delay
209. ExitForLoop:
210. add $v1, $t2, $0                         # return cnt
211. lw $a0, 0($sp)                           # load $a0
212. lw $a1, 4($sp)                           # load $a1
213. lw $a2, 8($sp)                           # load $a2
214. lw $s0, 12($sp)                          # load $s0
215. lw $s1, 16($sp)                          # load $s1
216. lw $s2, 20($sp)                          # load $s2
217. lw $s3, 24($sp)                          # load $s3
218. lw $s4, 28($sp)                          # load $s4
219. lw $ra, 32($sp)                          # load $ra
220. addi $sp, $sp, 36                        # recover stack for 9 items
221. jr $ra                                   # return
222. add $t0, $0, $0                          # delay