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 is        *
 * numElements and return the following:                                 *
 *                                                                       *
 * When cntType = 1, count the elements greater than or equal to 30      *
 * When cntType = -1, count the elements less than or equal to 5;        *
 * 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;
}
```

# 2. Design Process

## 2.1. `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.

## 2.2. `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.

## 2.3. `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`.

## 2.4. `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`.

## 2.5. `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`.

# 3. 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).

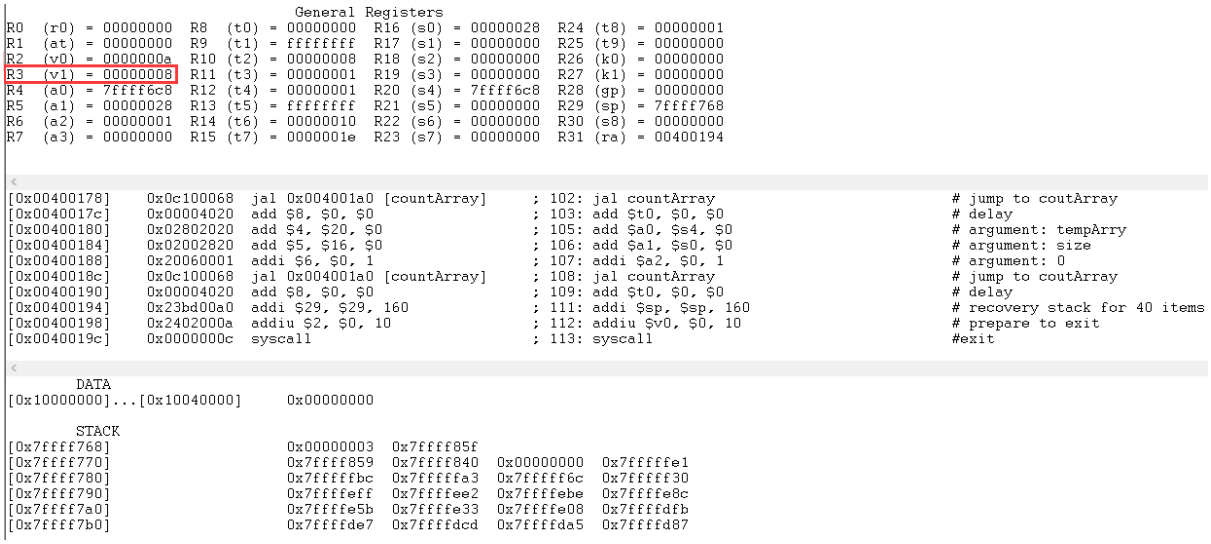


Figure 1. Result for `cntType` = 1

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).

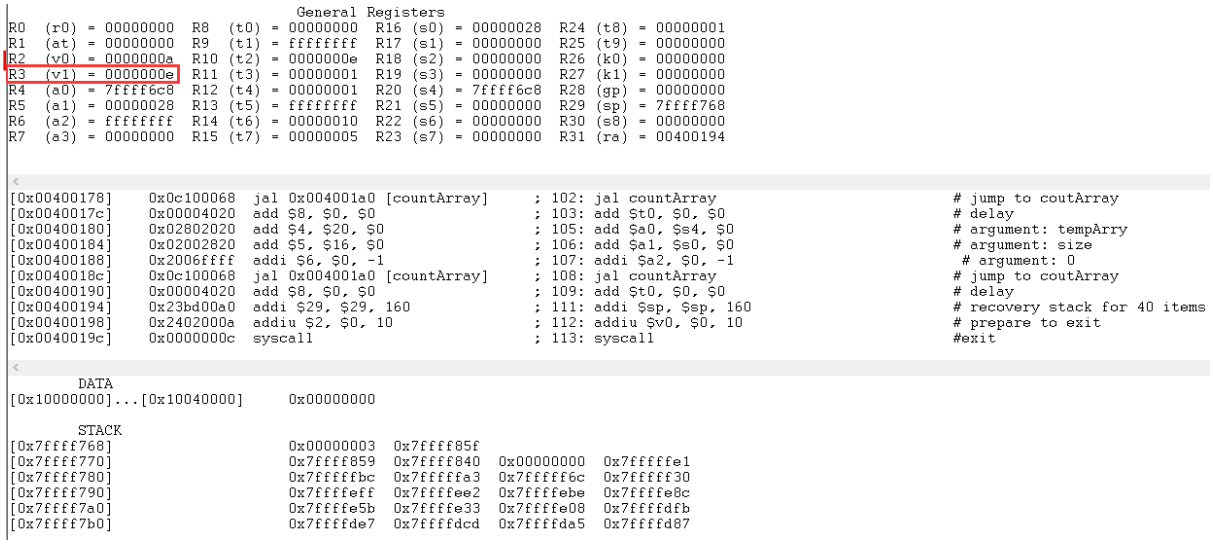


Figure 2. Result for `cntType` = -1

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).

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000028  R24 (t8) = 00000001
R1  (at) = 00000000  R9  (t1) = ffffffff  R17 (s1) = 00000000  R25 (t9) = 00000000
R2  (v0) = 0000000a  R10 (t2) = 00000012  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000012  R11 (t3) = 00000001  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7ffff6c8  R12 (t4) = 00000001  R20 (s4) = 7ffff6c8  R28 (gp) = 00000000
R5  (a1) = 00000028  R13 (t5) = ffffffff  R21 (s5) = 00000000  R29 (sp) = 7ffff768
R6  (a2) = 00000000  R14 (t6) = 00000010  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000005  R23 (s7) = 00000000  R31 (ra) = 00400194

<
[0x00400180]    0x02802020  add $4, $20, $0                ; 105: add $a0, $s4, $0                 # argument: tempArry
[0x00400184]    0x02002820  add $5, $16, $0                ; 106: add $a1, $s0, $0                 # argument: size
[0x00400188]    0x20060000  addi $6, $0, 0                 ; 107: addi $a2, $0, 0                  # argument: 0
[0x0040018c]    0x0c100068  jal 0x004001a0 [countArray]    ; 108: jal countArray                   # jump to coutArray
[0x00400190]    0x00004020  add $8, $0, $0                 ; 109: add $t0, $0, $0                  # delay
[0x00400194]    0x23bd00a0  addi $29, $29, 160             ; 111: addi $sp, $sp, 160               # recovery stack for 40 items
[0x00400198]    0x2402000a  addiu $2, $0, 10               ; 112: addiu $v0, $0, 10                # prepare to exit
[0x0040019c]    0x0000000c  syscall                        ; 113: syscall                          #exit
[0x004001a0]    0x23bdffdc  addi $29, $29, -36             ; 117: addi $sp, $sp, -36               # adjust stack for 9 items
[0x004001a4]    0xafbf0020  sw $31, 32($29)                ; 118: sw $ra, 32($sp)                  # store $ra

<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7ffff768]                    0x00000003  0x7ffff85f
[0x7ffff770]                    0x7ffff859  0x7ffff840  0x00000000  0x7fffffe1
[0x7ffff780]                    0x7ffffffbc 0x7ffffa3  0x7fffff6c  0x7fffff30
[0x7ffff790]                    0x7fffffeff 0x7ffffee2 0x7ffffebe  0x7ffffe8c
[0x7ffff7a0]                    0x7ffffe5b  0x7ffffe33  0x7ffffe08  0x7ffffdfb
[0x7ffff7b0]                    0x7ffffde7  0x7ffffdcd  0x7ffffda5  0x7ffffd87
```

Figure 3. Result for `cntType` = 0

From these results, we know our code runs correctly.

# 4. 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.

# 5. Reference

[1] Project1.pdf

# 6. Appendix

```
1.  .text
2.  .globl __start
3.  __start:
4.      addi $sp, $sp, -160                 # adjust stack for 40 items
5.
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]
11.
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.
93.     add $a0, $s4, $0                    # argument: tempArry
94.     add $a1, $s0, $0                    # argument: size
95.     addi $a2, $0, 1                     # argument: 1
96.     jal countArray                      # jump to coutArray
97.     add $t0, $0, $0                     # delay
98.
99.     add $a0, $s4, $0                    # argument: tempArry
100.        add $a1, $s0, $0                    # argument: size
101.        addi $a2, $0, -1                    # argument: -1
102.        jal countArray                      # jump to coutArray
103.        add $t0, $0, $0                     # delay
104.
105.        add $a0, $s4, $0                    # argument: tempArry
106.        add $a1, $s0, $0                    # argument: size
107.        addi $a2, $0, 0                     # argument: 0
108.        jal countArray                      # jump to coutArray
109.        add $t0, $0, $0                     # delay
110.
111.        addi $sp, $sp, 160                  # recovery stack for 40 items
112.        addiu $v0, $0, 10                   # prepare to exit
113.        syscall                             # exit
114.
115.    countArray:
116.        addi $sp, $sp, -36                  # adjust stack for 9 items
117.        sw $ra, 32($sp)                     # store $ra
118.        sw $s4, 28($sp)                     # store $s4
119.        sw $s3, 24($sp)                     # store $s3
120.        sw $s2, 20($sp)                     # store $s2
121.        sw $s1, 16($sp)                     # store $s1
122.        sw $s0, 12($sp)                     # store $s0
123.        sw $a2, 8($sp)                      # store $a2
124.        sw $a1, 4($sp)                      # store $a1
125.        sw $a0, 0($sp)                      # store $a0
```

```
126.
127.        addi $t1, $a1, -1                    # int i = numElements - 1
128.        add $t2, $0, $0                      # int cnt =0
129.        addi $t4, $0, 1                      # $t4 = 1
130.        addi $t5, $0, -1                     # $t5 = -1
131.
132.    ForLoop:
133.        slt $t3, $t1, $0                     # if $t1(i) < 0, $t3 = 1
134.        bne $t3, $0, ExitForLoop             # if $t3 != 0, jump to exit
135.        add $t0, $0, $0                      # delay
136.        beq $t4, $a2, Hot                    # if $t4 == $a2 == 1, jump to
     Hot
137.        add $t0, $0, $0                      # delay
138.        beq $t5, $a2, Cold                   # if $t5 == $a2 == -
    1, jump to Cold
139.        add $t0, $0, $0                      # delay
140.        j Comfort                            # Otherwise, jump to Comfort
141.        add $t0, $0, $0                      # delay
142.
143.    Hot:
144.        addi $sp, $sp, -4                    # adjust stack for 1 items
145.        sw $a0, 0($sp)                       # store $a0
146.        sll $t6, $t1, 2                      # $t1 * 4
147.        add $t6, $t6, $a0                    # $a0 + $t1 * 4
148.        lw $t6, 0($t6)                       # load value
149.        addi $t7, $0, 30
150.        slt $t8, $t6, $t7                    # if $t6 < 30, $t8 = 1
151.        bne $t8, $0, HotOtherwise            # if $t8 != 0, jump to HotOth
    erwise
152.        add $t0, $0, $0                      # delay
153.        addi $t2, $t2, 1                     # cnt++
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.
160.    HotOtherwise:
161.        addi $t2, $t2, 0                     # cnt + 0
162.        addi $t1, $t1, -1                    # i--
163.        lw $a0, 0($sp)                       # load $a0
164.        addi $sp, $sp, 4                     # recover stack for 1 items
```

```
165.        j ForLoop                               # jump to ForLoop
166.        add $t0, $0, $0                         # delay
167.
168.    Cold:
169.        addi $sp, $sp, -4                       # adjust stack for 1 items
170.        sw $a0, 0($sp)                          # store $a0
171.        sll $t6, $t1, 2                         # $t1 * 4
172.        add $t6, $t6, $a0                       # $a0 + $t1 * 4
173.        lw $t6, 0($t6)                          # load value
174.        addi $t7, $0, 5                         # $t7 = 5
175.        slt $t8, $t7, $t6                       # if $t7 < $t6, $t8 = 1
176.        bne $t8, $0, ColdOtherwise              # if $t8 != 0, jump to ColdOt
    herwise
177.        add $t0, $0, $0                         # delay
178.        addi $t2, $t2, 1                        # cnt++
179.        addi $t1, $t1, -1                       # i--
180.        lw $a0, 0($sp)                          # load $a0
181.        addi $sp, $sp, 4                        # recover stack for 1 items
182.        j ForLoop                               # jump to ForLoop
183.        add $t0, $0, $0                         # delay
184.
185.    ColdOtherwise:
186.        addi $t2, $t2, 0                        # cnt + 0
187.        addi $t1, $t1, -1                       # i--
188.        lw $a0, 0($sp)                          # load $a0
189.        addi $sp, $sp, 4                        # recover stack for 1 items
190.        j ForLoop                               # jump to ForLoop
191.        add $t0, $0, $0                         # delay
192.
193.    Comfort:
194.        addi $sp, $sp, -4                       # adjust stack for 1 items
195.        sw $a0, 0($sp)                          # store $a0
196.        sll $t6, $t1, 2                         # $t1 * 4
197.        add $t6, $t6, $a0                       # $a0 + $t1 * 4
198.        lw $t6, 0($t6)                          # load value
199.        addi $t7, $0, 30                        # $t7 = 30
200.        slt $t8, $t6, $t7                       # if $t6 < 30, $t8 = 1
201.        bne $t8, $0, Judge                      # if $t8 != 0, jump to Judge
202.        add $t0, $0, $0                         # delay
203.        addi $t2, $t2, 0                        # cnt + 0
204.        addi $t1, $t1, -1                       # i--
205.        lw $a0, 0($sp)                          # load $a0
```

```
206.        addi $sp, $sp, 4                    # recover stack for 1 items
207.        j ForLoop                           # jump to ForLoop
208.        add $t0, $0, $0                     # delay
209.
210.    Judge:
211.        addi $t1, $t1, -1                   # i--
212.        addi $t7, $0, 5                     # $t7 = 5
213.        slt $t8, $t7, $t6                   # if $t7(5) < $t6, $t8 = 1
214.        lw $a0, 0($sp)                      # load $a0
215.        addi $sp, $sp, 4                    # recover stack for 1 items
216.        beq $t8, $0, ForLoop                # if $t8 == 0 (x <= 5), jump
    to ForLoop
217.        add $t0, $0, $0                     # delay
218.        addi $t2, $t2, 1                    # cnt++
219.        j ForLoop                           # jump to ForLoop
220.        add $t0, $0, $0                     #delay
221.
222.    ExitForLoop:
223.        add $v1, $t2, $0                    # return cnt
224.        lw $a0, 0($sp)                      # load $a0
225.        lw $a1, 4($sp)                      # load $a1
226.        lw $a2, 8($sp)                      # load $a2
227.        lw $s0, 12($sp)                     # load $s0
228.        lw $s1, 16($sp)                     # load $s1
229.        lw $s2, 20($sp)                     # load $s2
230.        lw $s3, 24($sp)                     # load $s3
231.        lw $s4, 28($sp)                     # load $s4
232.        lw $ra, 32($sp)                     # load $ra
233.        addi $sp, $sp, 36                   # recover stack for 9 items
234.        jr $ra                              # return
235.        add $t0, $0, $0                     # delay
```