UM-SJTU JOINT INSTITURE

Intro to Computer Organization
(VE370)

Project 1

prof. Zheng

XINMIAO YU
518021910792

DATE: 28 Sept. 2020

1 Introduction

In this project, an array of 32-bits signed integer has a predefined **size 32**. These 32 numbers are assigned randomly. The C program is shown below.

```c
int main() {
    int size = 32; //determine the size of the array here
    int hotDay, coldDay, comfortDay;
    int tempArray[32] = {36, 9, -8, 40, 25, 20, 18, 19, 15, 16, 17, 16, 15,
    ↪  14,
    13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -3, 30, -19, 33};
    hotDay = countArray (tempArray, size, 1);
    coldDay = countArray (tempArray, size, -1);
    comfortDay = countArray (tempArray, size, 0);
}
int countArray(int A[], int numElements, int cntType) {
    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;
            default: 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;
}
```

A MIPS program with same functions as the C program is developed. In addition, no pseudo-instruction used.

2 Procedures

The program is developed following the logic of C program.

- As use `la` to load in the address of array is not allowed, the address used to store the numbers is predefined, as $0x10001000$. Then `main` function, where `int tempArray[32]=...` is needed, the base address ($0x10001000$) is load in $a0 through the command `lui` and `ori`.

- In `main` function, other variables needed is defined as

```
$a1 = size              // as numElements = 32
$s0 = hotDay           // as 0
$s1 = coldDay           // as 0
$s2 = comfortDay       // as 0
```

  function `countArray` is called three times. Before each call, make $a2 as 1/-1/0 and the **stack pointer** is adjusted for 6 items. The function arguments ($a0, $a1), return address ($ra) and saved register ($s0, $s1, $s2) are all saved into the stack before call `countArray` and restored after the function call. According to the return value in $v0, the number of hotDay/coldDay/comfortDay are all determined.

- In function `countArray`, define

```
$s0 = 0                 // as numElements - 1 = 31
$s1 = cnt               // as 0
```

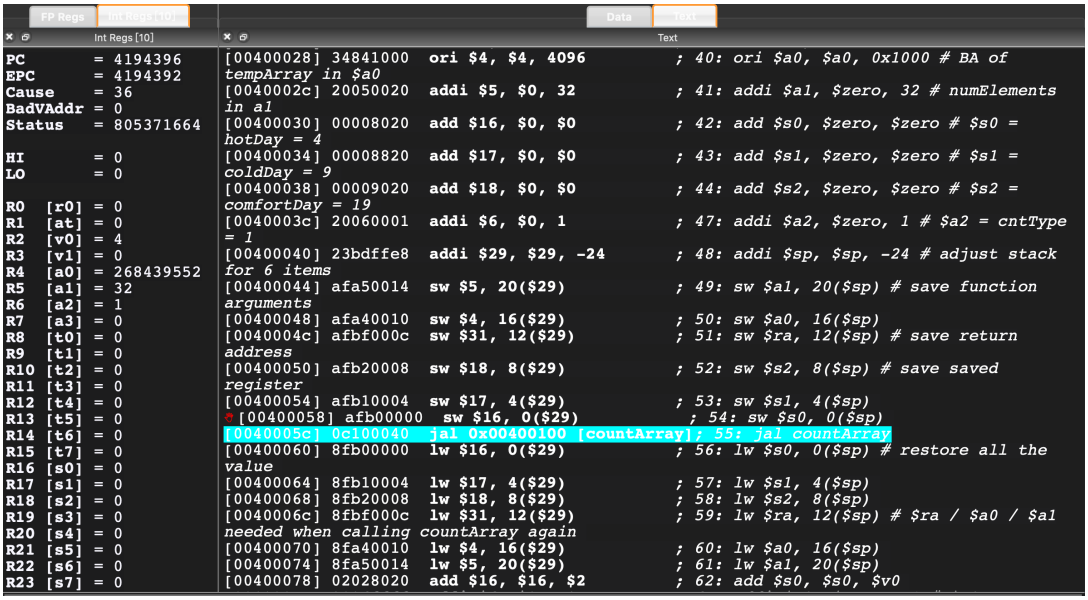
  Same as before, before function call adjust stack pointer for saving $a0, $a1, $a2, $ra, $s0, $s1. Set $a0 = A[i], where the address of A[i] is obtained through $a0 + $s0 *4. Restore all the value after each function call and change $s1 according to the value in $v0.

- For leaf function `hot/cold/comfort`, return the right value, stored in $v0.

3 Simulation Result

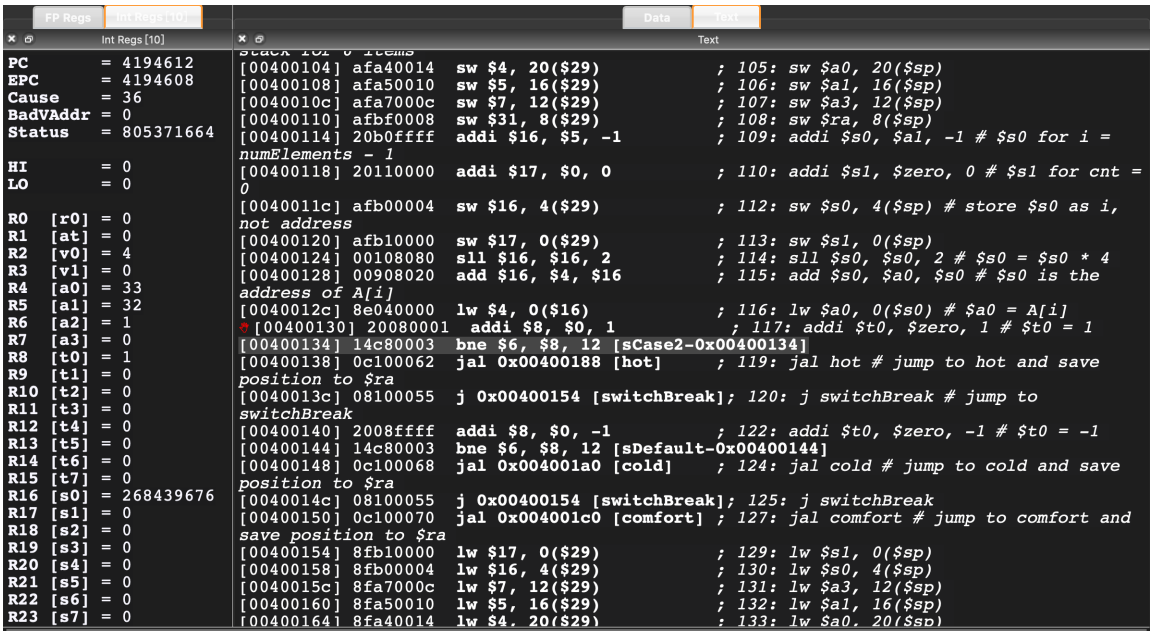
The simulation result is obtained through *QtSpim*. All the register show decimal value for simpler explanation.

1. **Before first call** `countArray`



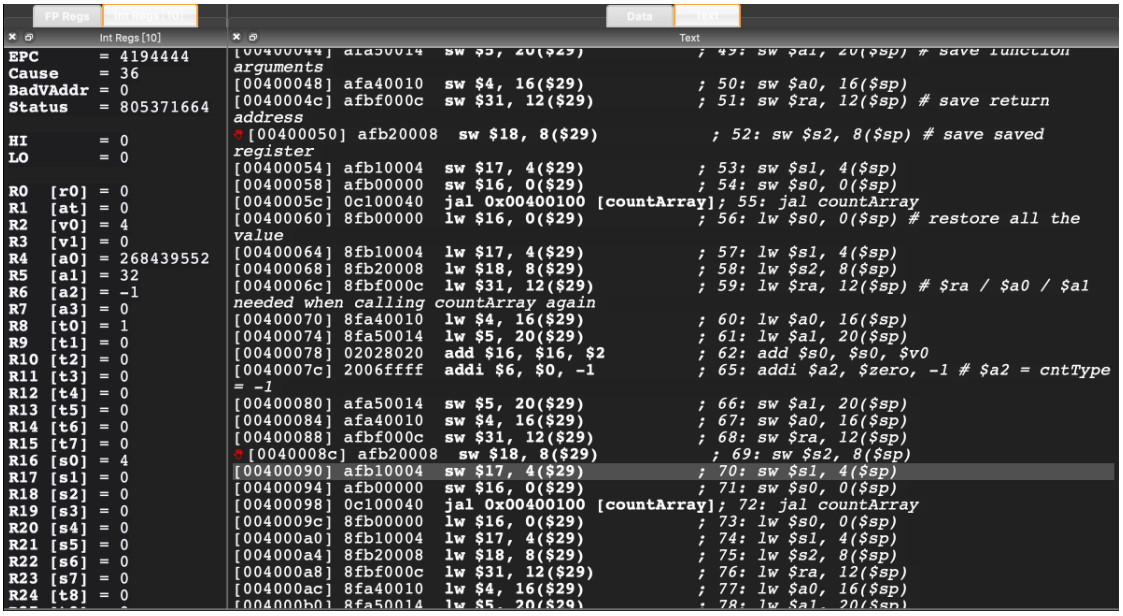
we could see that $a0 = $(268439552)_{10}$ = 0x10001000, $a1 = 32, $a2 = 1, as expected.

2. **Before first call** `hot/cold/comfort`



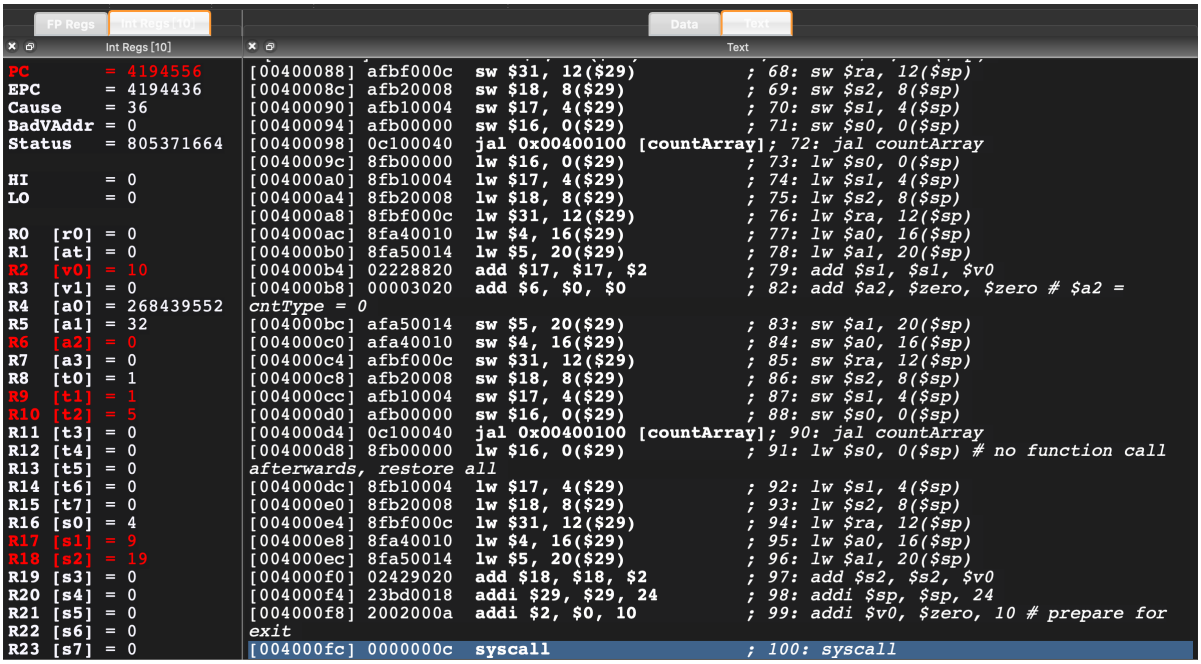
the value of A[i], which is A[31] = 33, is load in $a0, as function argument to pass to the leaf function. Now $s0 has the address of A[31] = $(268439676)_{10}$ = 0x1000107c = 0x10001000 + $(32 * 4)_{10}$. Then the loop will continue until all the value in the array are passed.

3. **After first call** `countArray`



The value of hotDay is stored in $s0, which is 4. $a2 is set as -1 for count coldDay. The value of $a0 and $a1 should not change as restored from stack. Then the procedure is similar, only differ in the leaf function called.

4. **Finish**



When the program finishes, $s0 = 4 (the number of hotDays), $s1 = 9 (the number of coldDays), $s2 = 19 (the number of comfortDays), as expected.

4 Conclusion

In conclusion, the program written through MIPS successfully finished the expected tasks and output the correct answer. Potential error might caused by

- use pseudo-instruction, caused syntax error.
- forget to adjust the stack pointer to save/restore values before/after each function call.
- the existed delay. Add some meaningless command will help.

A Program

```
1       .data 0x10001000
2   tempArrary:
3       .word    36
4       .word    9
5       .word    -8
6       .word    40
7       .word    25
8       .word    20
9       .word    18
10      .word    19
11      .word    15
12      .word    16
13      .word    17
14      .word    16
15      .word    15
16      .word    14
17      .word    13
18      .word    12
19      .word    11
20      .word    10
21      .word    9
22      .word    8
23      .word    7
24      .word    6
25      .word    5
26      .word    4
27      .word    3
28      .word    2
29      .word    1
30      .word    0
31      .word    -3
32      .word    30
33      .word    -19
34      .word    33
35      .text
36      .align    2
37      .globl    main
38  main:
39      lui     $a0, 0x1000
40      ori     $a0, $a0, 0x1000    # BA of tempArray in £a0
41      addi    $a1, $zero, 32      # numElements in a1
42      add     $s0, $zero, $zero   # £s0 = hotDay = 4
43      add     $s1, $zero, $zero   # £s1 = coldDay = 9
44      add     $s2, $zero, $zero   # £s2 = comfortDay = 19
45
46      ## First call: hotDay = countArray (tempArray, size, 1);
47      addi    $a2, $zero, 1       # £a2 = cntType = 1
```

```
48      addi    $sp, $sp, -24       # adjust stack for 6 items
49      sw      $a1, 20($sp)        # save function arguments
50      sw      $a0, 16($sp)
51      sw      $ra, 12($sp)        # save return address
52      sw      $s2, 8($sp)         # save saved register
53      sw      $s1, 4($sp)
54      sw      $s0, 0($sp)
55      add     $t0, $t0, $0        # meaningless
56      jal     countArray
57      lw      $s0, 0($sp)          # restore all the value
58      lw      $s1, 4($sp)
59      lw      $s2, 8($sp)
60      lw      $ra, 12($sp)         # £ra / £a0 / £a1 needed when calling
        ↪  countArray again
61      lw      $a0, 16($sp)
62      lw      $a1, 20($sp)
63      add     $s0, $s0, $v0
64
65      ## Second call: coldDay = countArray (tempArray, size, -1);
66      addi    $a2, $zero, -1      # £a2 = cntType = -1
67      sw      $a1, 20($sp)
68      sw      $a0, 16($sp)
69      sw      $ra, 12($sp)
70      sw      $s2, 8($sp)
71      sw      $s1, 4($sp)
72      sw      $s0, 0($sp)
73      add     $t0, $t0, $0        # meaningless
74      jal     countArray
75      lw      $s0, 0($sp)
76      lw      $s1, 4($sp)
77      lw      $s2, 8($sp)
78      lw      $ra, 12($sp)
79      lw      $a0, 16($sp)
80      lw      $a1, 20($sp)
81      add     $s1, $s1, $v0
82
83
84      ## comfortDay = countArray (tempArray, size, 0);
85      add     $a2, $zero, $zero   # £a2 = cntType = 0
86      sw      $a1, 20($sp)
87      sw      $a0, 16($sp)
88      sw      $ra, 12($sp)
89      sw      $s2, 8($sp)
90      sw      $s1, 4($sp)
91      sw      $s0, 0($sp)
92      ## only £s1 has chanegd, store this only
93      add     $t0, $t0, $0        # meaningless
94       add     $t0, $t0, $0        # meaningless
95      jal     countArray
```

```
96      lw      $s0, 0($sp)          # no function call afterwards, restore all
97      lw      $s1, 4($sp)
98      lw      $s2, 8($sp)
99      lw      $ra, 12($sp)
100     lw      $a0, 16($sp)
101     lw      $a1, 20($sp)
102     add     $s2, $s2, $v0
103     addi    $sp, $sp, 24
104     addi    $v0, $zero, 10      # for exit
105     syscall
106
107 ### Function countArray ###
108 countArray:
109     addi    $sp, $sp, -24       # adjust the stack for 6 items
110     sw      $a0, 20($sp)        # save function arguments
111     sw      $a1, 16($sp)
112     sw      $a3, 12($sp)
113     sw      $ra, 8($sp)         # save return address
114     addi    $s0, $a1, -1        # $s0 for i = numElements - 1
115     addi    $s1, $zero, 0       # $s1 for cnt = 0
116 cntLoop:
117     sw      $s0, 4($sp)         # store $s0 as i, not address
118     sw      $s1, 0($sp)
119     sll     $s0, $s0, 2         # $s0 = $s0 * 4
120     add     $s0, $a0, $s0       # $s0 is the address of A[i]
121     lw      $a0, 0($s0)         # $a0 = A[i]
122     addi    $t0, $zero, 1       # $t0 = 1
123     bne        $a2, $t0, sCase2    # if $a2 != 1, then not hot
124     jal        hot                    # jump to hot and save position to $ra
125     j        switchBreak            # jump to switchBreak
126 sCase2:
127     addi    $t0, $zero, -1      # $t0 = -1
128     bne     $a2, $t0, sDefault  # if $a2 != -1, then not cold
129     jal        cold                # jump to cold and save position to $ra
130     j       switchBreak
131 sDefault:
132     add     $t0, $t0, $0        # meaningless
133     jal        comfort                # jump to comfort and save position to
    ↪  $ra
134 switchBreak:
135     lw      $s1, 0($sp)
136     lw      $s0, 4($sp)
137     lw      $a3, 12($sp)
138     lw      $a1, 16($sp)
139     lw      $a0, 20($sp)
140     add     $s1, $s1, $v0       # cnt += $v0
141     addi    $s0, $s0, -1        # i--
142     slti    $t0, $s0, 0         # if $s0 < 0, $t0 = 0
143     beq        $t0, $zero, cntLoop    # if $t0 != 0 then continue the loop
```

```
144     lw      $ra, 8($sp)         # else, exit the loop, restore $ra
145     add     $v0, $s1, $zero     # $v0 = cnt
146     addi    $sp, $sp, 24        # destroy spaces on stack
147     jr      $ra
148
149
150 ### Function hot ###
151 hot:
152     slti    $v0, $a0, 30
153     beq     $v0, $zero, hotTrue
154     add     $v0, $zero, $zero   # A[i] < 30, $v0 = 0
155     jr      $ra
156 hotTrue:
157     addi    $v0, $zero, 1       # A[i] >= 30, $v0 = 1
158     jr      $ra
159
160 ### Function cold ###
161 cold:
162     slti    $t0, $a0, 5         # A[i] < 5, $t0 = 1
163     beq     $t0, $zero, coldFalse
164 coldTrue:
165     addi    $v0, $zero, 1       # $v0 = 1
166     jr      $ra
167 coldFalse:
168     addi    $t1, $zero, 5
169     beq        $a0, $t1, coldTrue    # if $a0 == $t1 == 5 then coldTrue
170     add        $v0, $zero, $zero     # $v0 = $zero + $zero = 0
171     jr      $ra
172
173 ### Function comfort ###
174 comfort:
175     slti    $t0, $a0, 30
176     addi    $t1, $zero, 1
177     beq        $t0, $t1, comfortTrue    # if $t0 == 1, $a0 < 30, then
        ↪ comfortTrue
178 comfortFalse:
179     add     $v0, $zero, $zero
180     jr      $ra
181 comfortTrue:
182     slti    $t0, $a0, 5
183     beq        $t0, $t1, comfortFalse    # if $t0 == 1, $a0 < 5 then
        ↪ comfortFalse
184     addi    $t2, $zero, 5
185     beq        $a0, $t2, comfortFalse    # if $a0 == 5 comfortFalse
186     addi    $v0, $zero, 1
187     jr      $ra
188
```