VE370 Project 1

Liu Yihao 515370910207

1 Objectives

Develop a MIPS assembly program that operates on a data segment consisting of an array of 32-bit unsigned integers. In the text (program) segment of memory, write a procedure called main that implements the main() function and other subroutines described below. Assemble, simulate, and carefully comment the file. Screen print your simulation results and explain the results by annotating the screen prints. You should compose an array whose size is determined by you in the main function and is not less than 20 elements.

```
main() {
    int size = ...;  //determine the size of the array here
    int PassCnt, FailCnt;
    int testArray[size] = { 55, 83,
        ... //compose your own array here
    };
    PassCnt = countArray(testArray, size, 1);
    FailCnt = countArray(testArray, size, -1);
}

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 60;  *
* When cntType = -1, count the elements less than 60;                *
*************************************************************/
    int i, cnt = 0;
    for(i = numElements - 1; i > -1; i--) {
        switch (cntType) {
            case '1' : cnt += Pass(A[i]); break;
            otherwise: cnt += Fail(A[i]);
        }
    }
    return cnt;
}

int Pass(int x) {
    if(x >= 60) return 1;
    else return 0;
```

```
}

int Fail(int x) {
    if (x < 60) return 1;
    else return 0;
}
```

2 Procedure

The effects of all MIPS statements are carefully commented, so I won't show many details here.

2.1 main function

In the main function, I use the stack to save the scores. I choose 50 as the size, and I write a python script to generate the data, uniformly distributed in $[0, 100]$.

```
import random

size = 50

arr = [55, 83]
for i in range(2, size):
    arr.append(random.randint(0, 100))
print(arr)

fid = open('testArray.s', 'w')
for i in range(size):
    fid.write('        addi $t0, $0, %2d    # $t0 = %d\n' % (arr[i], arr[i]))
    fid.write('        sw $t0, %3d($s4)    # testArray[%d] = $t0\n' % (i * 4, i))
```

Then I called the function countArray for two times to count the pass and fail number. I also use the stack the save ascii strings "Pass: ", "Fail: " and "\n" so that you can see a user friendly result on the console.

2.2 countArray function

In the countArray function, I use saved registers $s0, $s1, $s2, $s3, $s4, and I call other functions, so these registers and $ra should be saved into stack in the begin and load from the stack in the end. Then I write a for loop and a condition statement, so some jump tags is added.

2.3 Pass and Fail function

Theses two function are very easy, no stack space is needed, only a condition statement can give the correct result.

3 Result

I use the data [55, 83, 21, 20, 40, 49, 42, 35, 92, 8, 65, 88, 25, 100, 43, 9, 98, 10, 81, 63, 83, 27, 42, 81, 94, 2, 40, 49, 75, 46, 67, 46, 89, 27, 39, 12, 19, 41, 86, 3, 14, 64, 22, 64, 8, 38, 32, 26, 64, 5], there

are 50 in total, 18 pass, 22 fail.

The simulation result is shown in Figure 1.

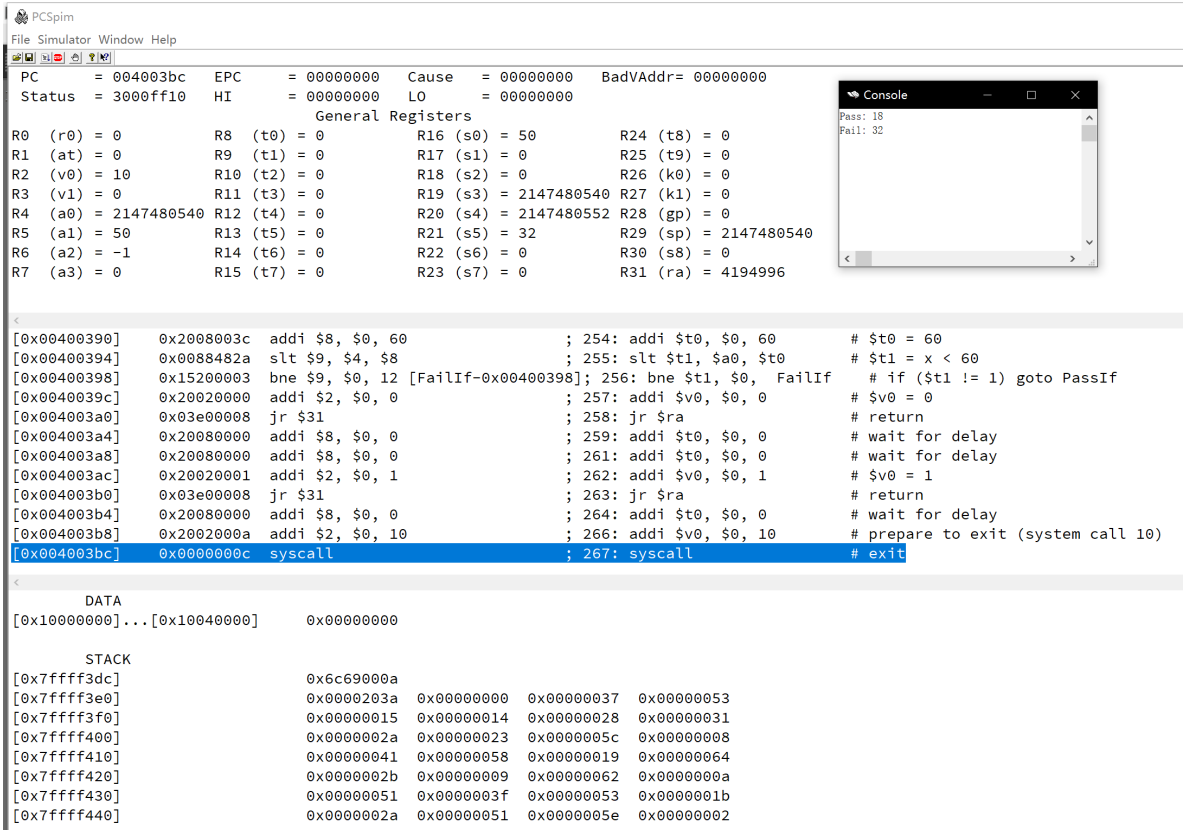

Figure 1: Simulation result

4 Conclusion

The things I want to mention are mainly in two aspects.

First, programming in pseudo instructions is much more efficient then without them. I made a mistake that I didn't switch off this option, and I found many pseudo instructions on the Internet. I used the .data scope to save ascii and word, with the usage of la instruction. However, pseudo instructions are not permitted, so I have to change them into silly assignment on stack (I know a c compiler will put these static data on the global variable area).

Second, the delay in branches and loads is quite confusing. I added many no-effect statement after jr, jal and lw to avoid the skip of those useful statements. I wondered whether the delay should be considered in a real assembly environment.

5 Appendix

```
        .text
        .globl __start
__start:
        addi $sp, $sp, -212     # adjust stack for 50*4+12 item

        addi $s0, $0, 50        # int size = 50
        addu $s1, $0, $0        # int PassCnt = 0
        addu $s2, $0, $0        # int FailCnt = 0
        addu $s3, $0, $sp       # char buffer[10]
        addiu $s4, $s3, 12      # int testArray[size]

        addi $t0, $0, 55        # $t0 = 55
        sw $t0,   0($s4)        # testArray[0] = $t0
        addi $t0, $0, 83        # $t0 = 83
        sw $t0,   4($s4)        # testArray[1] = $t0
        addi $t0, $0, 21        # $t0 = 21
        sw $t0,   8($s4)        # testArray[2] = $t0
        addi $t0, $0, 20        # $t0 = 20
        sw $t0,  12($s4)        # testArray[3] = $t0
        addi $t0, $0, 40        # $t0 = 40
        sw $t0,  16($s4)        # testArray[4] = $t0
        addi $t0, $0, 49        # $t0 = 49
        sw $t0,  20($s4)        # testArray[5] = $t0
        addi $t0, $0, 42        # $t0 = 42
        sw $t0,  24($s4)        # testArray[6] = $t0
        addi $t0, $0, 35        # $t0 = 35
        sw $t0,  28($s4)        # testArray[7] = $t0
        addi $t0, $0, 92        # $t0 = 92
        sw $t0,  32($s4)        # testArray[8] = $t0
        addi $t0, $0,  8        # $t0 = 8
        sw $t0,  36($s4)        # testArray[9] = $t0
        addi $t0, $0, 65        # $t0 = 65
        sw $t0,  40($s4)        # testArray[10] = $t0
        addi $t0, $0, 88        # $t0 = 88
        sw $t0,  44($s4)        # testArray[11] = $t0
        addi $t0, $0, 25        # $t0 = 25
        sw $t0,  48($s4)        # testArray[12] = $t0
        addi $t0, $0, 100       # $t0 = 100
        sw $t0,  52($s4)        # testArray[13] = $t0
        addi $t0, $0, 43        # $t0 = 43
        sw $t0,  56($s4)        # testArray[14] = $t0
        addi $t0, $0,  9        # $t0 = 9
        sw $t0,  60($s4)        # testArray[15] = $t0
        addi $t0, $0, 98        # $t0 = 98
        sw $t0,  64($s4)        # testArray[16] = $t0
        addi $t0, $0, 10        # $t0 = 10
        sw $t0,  68($s4)        # testArray[17] = $t0
```

```
48          addi $t0, $0, 81        # £t0 = 81
49          sw $t0,  72($s4)        # testArray[18] = £t0
50          addi $t0, $0, 63        # £t0 = 63
51          sw $t0,  76($s4)        # testArray[19] = £t0
52          addi $t0, $0, 83        # £t0 = 83
53          sw $t0,  80($s4)        # testArray[20] = £t0
54          addi $t0, $0, 27        # £t0 = 27
55          sw $t0,  84($s4)        # testArray[21] = £t0
56          addi $t0, $0, 42        # £t0 = 42
57          sw $t0,  88($s4)        # testArray[22] = £t0
58          addi $t0, $0, 81        # £t0 = 81
59          sw $t0,  92($s4)        # testArray[23] = £t0
60          addi $t0, $0, 94        # £t0 = 94
61          sw $t0,  96($s4)        # testArray[24] = £t0
62          addi $t0, $0,  2        # £t0 = 2
63          sw $t0, 100($s4)        # testArray[25] = £t0
64          addi $t0, $0, 40        # £t0 = 40
65          sw $t0, 104($s4)        # testArray[26] = £t0
66          addi $t0, $0, 49        # £t0 = 49
67          sw $t0, 108($s4)        # testArray[27] = £t0
68          addi $t0, $0, 75        # £t0 = 75
69          sw $t0, 112($s4)        # testArray[28] = £t0
70          addi $t0, $0, 46        # £t0 = 46
71          sw $t0, 116($s4)        # testArray[29] = £t0
72          addi $t0, $0, 67        # £t0 = 67
73          sw $t0, 120($s4)        # testArray[30] = £t0
74          addi $t0, $0, 46        # £t0 = 46
75          sw $t0, 124($s4)        # testArray[31] = £t0
76          addi $t0, $0, 89        # £t0 = 89
77          sw $t0, 128($s4)        # testArray[32] = £t0
78          addi $t0, $0, 27        # £t0 = 27
79          sw $t0, 132($s4)        # testArray[33] = £t0
80          addi $t0, $0, 39        # £t0 = 39
81          sw $t0, 136($s4)        # testArray[34] = £t0
82          addi $t0, $0, 12        # £t0 = 12
83          sw $t0, 140($s4)        # testArray[35] = £t0
84          addi $t0, $0, 19        # £t0 = 19
85          sw $t0, 144($s4)        # testArray[36] = £t0
86          addi $t0, $0, 41        # £t0 = 41
87          sw $t0, 148($s4)        # testArray[37] = £t0
88          addi $t0, $0, 86        # £t0 = 86
89          sw $t0, 152($s4)        # testArray[38] = £t0
90          addi $t0, $0,  3        # £t0 = 3
91          sw $t0, 156($s4)        # testArray[39] = £t0
92          addi $t0, $0, 14        # £t0 = 14
93          sw $t0, 160($s4)        # testArray[40] = £t0
94          addi $t0, $0, 64        # £t0 = 64
95          sw $t0, 164($s4)        # testArray[41] = £t0
96          addi $t0, $0, 22        # £t0 = 22
```

```
97          sw $t0, 168($s4)        # testArray[42] = £t0
98          addi $t0, $0, 64        # £t0 = 64
99          sw $t0, 172($s4)        # testArray[43] = £t0
100         addi $t0, $0,  8        # £t0 = 8
101         sw $t0, 176($s4)        # testArray[44] = £t0
102         addi $t0, $0, 38        # £t0 = 38
103         sw $t0, 180($s4)        # testArray[45] = £t0
104         addi $t0, $0, 32        # £t0 = 32
105         sw $t0, 184($s4)        # testArray[46] = £t0
106         addi $t0, $0, 26        # £t0 = 26
107         sw $t0, 188($s4)        # testArray[47] = £t0
108         addi $t0, $0, 64        # £t0 = 64
109         sw $t0, 192($s4)        # testArray[48] = £t0
110         addi $t0, $0,  5        # £t0 = 5
111         sw $t0, 196($s4)        # testArray[49] = £t0
112
113         addu $a0, $0, $s4       # £a0 = testArray
114         addu $a1, $0, $s0       # £a1 = size
115         addi $a2, $0, 1         # £a2 = 1
116         jal countArray          # £v0 = countArray(testArray, size, 1)
117         addi $t1, $0, 1         # wait for delay
118         addu $s5, $0, $v0       # save the result into £s4
119
120         addi $t0, $0, 80        # Init the string "Pass: "
121         sb $t0, 0($s3)
122         addi $t0, $0, 97
123         sb $t0, 1($s3)
124         addi $t0, $0, 115
125         sb $t0, 2($s3)
126         sb $t0, 3($s3)
127         addi $t0, $0, 58
128         sb $t0, 4($s3)
129         addi $t0, $0, 32
130         sb $t0, 5($s3)
131         addi $t0, $0, 0
132         sb $t0, 6($s3)
133         addiu $a0, $s3, 0       # £a0 = £s3 ("Pass: ")
134         addi $v0, $0, 4         # prepare to string output (system call 4)
135         syscall                 # string output
136
137         addu $a0, $0, $s5       # £a0 = £s4
138         addi $v0, $0, 1         # prepare to int output (system call 1)
139         syscall                 # int output
140
141         addi $t0, $0, 10        # Init the string "\n"
142         sb $t0, 0($s3)
143         addi $t0, $0, 0
144         sb $t0, 1($s3)
145         addiu $a0, $s3, 0       # £a0 = £s3 ("\n")
```

```
146         addi $v0, $0, 4         # prepare to string output (system call 4)
147         syscall                 # string output
148
149         addu $a0, $0, $s4       # £a0 = testArray
150         addu $a1, $0, $s0       # £a1 = size
151         addi $a2, $0, -1        # £a2 = -1
152         jal countArray          # £v0 = countArray(testArray, size, -1)
153         addi $t1, $0, 1         # wait for delay
154         addu $s5, $0, $v0       # save the result into £s4
155
156         addi $t0, $0, 70        # Init the string "Fail: "
157         sb $t0, 0($s3)
158         addi $t0, $0, 97
159         sb $t0, 1($s3)
160         addi $t0, $0, 105
161         sb $t0, 2($s3)
162         addi $t0, $0, 108
163         sb $t0, 3($s3)
164         addi $t0, $0, 58
165         sb $t0, 4($s3)
166         addi $t0, $0, 32
167         sb $t0, 5($s3)
168         addi $t0, $0, 0
169         sb $t0, 6($s3)
170         addiu $a0, $s3, 0       # £a0 = £s3 ("Fail: ")
171         addi $v0, $0, 4         # prepare to string output (system call 4)
172         syscall                 # string output
173         addu $a0, $0, $s5       # £a0 = £v0
174         addi $v0, $0, 1         # prepare to int output (system call 1)
175         syscall                 # int output
176
177         addi $t0, $0, 10        # Init the string "\n"
178         sb $t0, 0($s3)
179         addi $t0, $0, 0
180         sb $t0, 1($s3)
181         addiu $a0, $s3, 0       # £a0 = £s3 ("\n")
182         addi $v0, $0, 4         # prepare to string output (system call 4)
183         syscall                 # string output
184
185         jal exit
186         addi $t0, $0, 0
187 countArray:
188         addi $sp, $sp, -24      # adjust stack for 6 items
189         sw $ra, 20($sp)         # save £ra on stack
190         sw $s4, 16($sp)         # save £s4 on stack
191         sw $s3, 12($sp)         # save £s3 on stack
192         sw $s2, 8($sp)          # save £s2 on stack
193         sw $s1, 4($sp)          # save £s1 on stack
194         sw $s0, 0($sp)          # save £s0 on stack
```

```
195
196         addu $s0, $0, $a0       # save £a0(int A[]) into £s0
197         addu $s1, $0, $a1       # save £a1(int numElements) into £s1
198         addu $s2, $0, $a2       # save £a2(int cntType) into £s2
199
200         addi $s3, $s1, -1       # £s3(i) = numElements - 1
201         addi $s4, $0, 0         # £s4(cnt) = 0
202 countArrayFor:
203         addi $t0, $0, 0         # wait for delay
204         slt $t0, $s3, $0        # £t0 = i < 0
205         bne $t0, $0, countArrayEndFor
206                                 # if (£t0 != 0) goto countArrayEndFor
207         sll $t0, $s3 ,2         # £t0 = i * 4
208         add $t0, $s0, $t0       # £t0 = A + £t0
209         lw $a0, 0($t0)          # £a0 = A[i]
210         addi $t1, $0, 1         # £t1 = 1
211         addi $t1, $0, 1         # wait for delay
212         bne $s2, $t1, countArrayElse
213                                 # if (cntType != 1) goto countArrayElse
214         jal Pass                # £v0 = Pass(A[i])
215         addi $t1, $0, 1         # wait for delay
216         j countArrayEndIf       # jump to endif
217         addi $t0, $0, 0         # wait for delay
218 countArrayElse:
219         addi $t0, $0, 0         # wait for delay
220         jal Fail                # £v0 = Fail(A[i])
221         addi $t1, $0, 1         # wait for delay
222 countArrayEndIf:
223         addi $t0, $0, 0         # wait for delay
224         addu $s4, $s4, $v0      # cnt += £v0
225         addi $s3, $s3, -1       # i--
226         j countArrayFor         # jump to for begin
227         addi $t0, $0, 0         # wait for delay
228 countArrayEndFor:
229         addi $t0, $0, 0         # wait for delay
230         addu $v0, $0, $s4       # £v0 = cnt
231         lw $s0, 0($sp)          # restore £s0 from stack
232         lw $s1, 4($sp)          # restore £s1 from stack
233         lw $s2, 8($sp)          # restore £s2 from stack
234         lw $s3, 12($sp)         # restore £s3 from stack
235         lw $s4, 16($sp)         # restore £s4 from stack
236         lw $ra, 20($sp)         # restore £ra from stack
237         addi $sp, $sp, 24       # recover the stack
238         addi $t1, $0, 0         # wait for delay
239         jr $ra                  # return
240         addi $t0, $0, 0         # wait for delay
241 Pass:
242         addi $t0, $0, 60        # £t0 = 60
243         slt $t1, $a0, $t0       # £t1 = x < 60
```

```
244         beq $t1, $0, PassIf     # if (£t1 == 1) goto PassIf
245         addi $v0, $0, 0         # £v0 = 0
246         jr $ra                  # return
247         addi $t0, $0, 0         # wait for delay
248 PassIf:
249         addi $t0, $0, 0         # wait for delay
250         addi $v0, $0, 1         # £v0 = 1
251         jr $ra                  # return
252         addi $t0, $0, 0         # wait for delay
253 Fail:
254         addi $t0, $0, 60        # £t0 = 60
255         slt $t1, $a0, $t0       # £t1 = x < 60
256         bne $t1, $0,  FailIf    # if (£t1 != 1) goto PassIf
257         addi $v0, $0, 0         # £v0 = 0
258         jr $ra                  # return
259         addi $t0, $0, 0         # wait for delay
260 FailIf:
261         addi $t0, $0, 0         # wait for delay
262         addi $v0, $0, 1         # £v0 = 1
263         jr $ra                  # return
264         addi $t0, $0, 0         # wait for delay
265 exit:
266         addi $v0, $0, 10        # prepare to exit (system call 10)
267         syscall                 # exit
```