

UM-SJTU JOINT INSTITUTE
Introduction to Computer Organization
(VE370)

Project1

Name: Pan Chongdan

ID: 516370910121

Date: October 2, 2018

1 Introduction

The project asks me to develop a MIPS assembly program that operates on a data segment consisting of an array of 32-bit unsigned integers. In the text segment of memory, I'll write a procedure called main that implements the main() function and other subroutines described below. Assemble, simulate, and carefully comment the file. I'll also screen print my simulation results and explain the results by annotating the screen prints. I compose an array whose size 30.

```
1 main() {
2     int size = ...; //determine the size of the array here
3     int PassCnt, FailCnt;
4     int testArray[size] = { 55, 83,
5         ... //compose your own array here
6     };
7     PassCnt = countArray(testArray, size, 1);
8     FailCnt = countArray(testArray, size, -1);
9 }
10 int countArray(int A[], int numElements, int cntType) {
11     /******
12     * Count specific elements in the integer array A[] whose size is
13     * numElements and return the following:
14     *
15     * When cntType = 1, count the elements greater than or equal to 60;
16     * When cntType = -1, count the elements less than 60;
17     *****/
18     int i, cnt = 0;
19     for(i=numElements-1, i>0, i--) {
20         switch (cntType) {
21             case '1' : cnt += Pass(A[i]); break;
22             otherwise: cnt += Fail(A[i]);
23         }
24     }
25     return cnt;
26 }
27 int Pass(int x) {
28     if(x>=60) return 1;
29     else return 0;
30 }
31 int Fail(int x) {
32     if (x<60) return 1;
33     else return 0;
34 }
```

Figure 1: Original C++ code

The program will count the numbers of elements less than 60 or bigger than 60 in my array.

2 Procedure

2.1 Generate the Array for simulation

First, I write a CPP code to generate an array whose size is 30.

```
main() {  
    srand(time(NULL));  
    int size = 31; //determine the size of the array here  
    int PassCnt, FailCnt;  
    int testArray[size];  
    for(int i=0;i<size;i++){  
        testArray[i]=rand()%100;  
        cout<<testArray[i]<<" ";  
    }  
    PassCnt = countArray(testArray, size, 1);  
    FailCnt = countArray(testArray, size, -1);  
    cout<<endl<<"PassCnt= "<<PassCnt<<endl<<"FailCnt= "<<FailCnt;  
}
```

Figure 2: C++ code to generate the array

For simulation, I choose the array with elements :

55 84 13 48 29 75 53 42 97 2 81 36 19 69 55 0 55 94 68 1 62 76 41 66 9 10 12 28 65 62
where 12 passed and 18 failed

3 Conclusion

It takes me a long time to finish my project, and here is my simulation result and my conclusion.

PC = 00400150 EPC = 00000000 Cause = 00000000 BadVAddr = 00000000
Status = 3000fff10 HI = 00000000 LO = 00000000

General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 0000001e R24 (t8) = 00000000
R1 (a0) = 00000000 R9 (t1) = 00000001 R17 (s1) = 0000000c R25 (t9) = 00000000
R2 (v0) = 0000000e R10 (t2) = 00000000 R18 (s2) = 00000011 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 7ffff588 R27 (k1) = 00000000
R4 (a0) = 00000011 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 00000000
R5 (a1) = 0000001e R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7ffff588
R6 (a2) = ffffffff R14 (t6) = 00000000 R22 (s6) = 00000000 R30 (s8) = 00000000
R7 (a3) = 00000000 R15 (t7) = 00000000 R23 (s7) = 00000000 R31 (s9) = 0040013e

FIR = 00009800 FCSR = 00000000 FCCR = 0
FENR = 00000000

01217

[0x0040011c] 0x00112020 add \$4, \$0, \$17
[0x00400120] 0x20020001 addi \$2, \$0, 1
[0x00400124] 0x0000000c syscall
[0x00400128] 0x00132020 add \$4, \$0, \$19
[0x0040012c] 0x00102820 add \$5, \$0, \$16
[0x00400130] 0x2006ffff addi \$6, \$0, -1
[0x00400134] 0xc0c10055 jal 0x00400154 [countArray
[0x00400138] 0x00004020 add \$8, \$0, \$0
[0x0040013c] 0x00409020 add \$18, \$2, \$0
[0x00400140] 0x00122020 add \$4, \$0, \$18
[0x00400144] 0x20020001 addi \$2, \$0, 1
[0x00400148] 0x0000000c syscall
[0x0040014c] 0x2402000a addiu \$2, \$0, 10
[0x00400150] 0x0000000c syscall

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

STACK
[0x7ffff588] 0x00000037 0x00000054
[0x7ffff590] 0x00000004 0x00000030
[0x7ffff5a0] 0x00000035 0x0000002a
[0x7ffff5b0] 0x00000051 0x00000024
[0x7ffff5c0] 0x00000037 0x00000000
[0x7ffff5d0] 0x00000044 0x00000001
[0x7ffff5e0] 0x00000029 0x00000042
[0x7ffff5f0] 0x0000000c 0x0000001c 0x00000041 0x0000003e
[0x7ffff600] 0x00000004 0x7ffff72c 0x7ffff726 0x7ffff70d
[0x7ffff610] 0x7ffff6d9 0x00000000 0x7ffff7e0 0x7ffff7c1

SPIM Version 9.1.4 of September 4, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
Memory and registers cleared and the simulator reinitialized.

SPIM Version 9.1.4 of September 4, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
D:\PANDA-Study\VE370\Project\Project1\main\main.s successfully loaded

1. From the console we can see the number for pass is 12 and for failure is 17, it's obvious to see their sum is 29, which is smaller than the size. I think it's because in the statement of the for loop given from the project introduction, the `i<0` should be changed to `i<=0` because we haven't visited element 0 yet. So we miss a element smaller than 60, which is 55 in my array.
2. Write meaningless code for delay is very important. At first, I can't run my code but I can't find where is the problem. Then, it took me a long time to debug through breakpoints, and in the end I found I need to place some delay code at the `jr` and `jal` instruction, otherwise mips will do another instruction and change the value in registers.
3. As you can see from my result, I can't separate 12 from 17 because I don't know how to output a string. At first, I find some way on the internet by using `la` and `.asciiz` code, but it doesn't work afterwards. I think it's because the setting of PCSpin and we'll learn some other way later.

4 Appendix

```
■      #main.s
.text
.globl __start
__start:
    addi $sp, $sp, -120 # create 30*4 spaces on the stack
    addi $s0, $0, 30 # int size=30
    add  $s1, $0, $0 # int passcnt=0
    add  $s2, $0, $0 # int failcnt=0
    addu $s3, $0, $sp
    addi $t0, $0, 55 # testarray[0]=55
    sw $t0, 0($s3)
    addi $t0, $0, 84
    sw $t0, 4($s3)
    addi $t0, $0, 13
    sw $t0, 8($s3)
    addi $t0, $0, 48
    sw $t0, 12($s3)
    addi $t0, $0, 29
    sw $t0, 16($s3)
    addi $t0, $0, 75
    sw $t0, 20($s3)
    addi $t0, $0, 53
    sw $t0, 24($s3)
    addi $t0, $0, 42
    sw $t0, 28($s3)
    addi $t0, $0, 97
    sw $t0, 32($s3)
    addi $t0, $0, 2
    sw $t0, 36($s3)
    addi $t0, $0, 81
    sw $t0, 40($s3)
    addi $t0, $0, 36
    sw $t0, 44($s3)
    addi $t0, $0, 19
    sw $t0, 48($s3)
    addi $t0, $0, 69
    sw $t0, 52($s3)
    addi $t0, $0, 55
    sw $t0, 56($s3)
    addi $t0, $0, 0
    sw $t0, 60($s3)
    addi $t0, $0, 55
    sw $t0, 64($s3)
    addi $t0, $0, 94
    sw $t0, 68($s3)
    addi $t0, $0, 68
    sw $t0, 72($s3)
    addi $t0, $0, 1
    sw $t0, 76($s3)
    addi $t0, $0, 62
    sw $t0, 80($s3)
    addi $t0, $0, 76
    sw $t0, 84($s3)
    addi $t0, $0, 41
    sw $t0, 88($s3)
    addi $t0, $0, 66
    sw $t0, 92($s3)
    addi $t0, $0, 9
    sw $t0, 96($s3)
    addi $t0, $0, 10
```

```

sw $t0, 100($s3)
addi $t0, $0, 12
sw $t0, 104($s3)
addi $t0, $0, 28
sw $t0, 108($s3)
addi $t0, $0, 65
sw $t0, 112($s3)
addi $t0, $0, 62
sw $t0, 116($s3) #the above process is to initialize the testarray

add $a0, $0, $s3 #input argument A[]
add $a1, $0, $s0 #input argument numelements
addi $a2, $0, 1 #input argument l
jal countArray
add $t0, $0, $0 #delay
add $s1, $v0, $0 #save the result from the first countArray function
add $a0, $0, $s1
addi $v0, $0, 1 #standby for integer output
syscall #output the pass number

add $a0, $0, $s3 #input argument A[]
add $a1, $0, $s0 #input argument numelements
addi $a2, $0, -1 #input argument l
jal countArray
add $t0, $0, $0 #delay
add $s2, $v0, $0 #save the result from the first countArray function

add $a0, $0, $s2
addi $v0, $0, 1 #standby for integer output
syscall #output the pass number

addiu $v0, $0, 10 # Prepare to exit (system call 10)
syscall # Exit

countArray:
addi $sp, $sp, -24 #create 6 spaces for stack points for countarray
sw $ra, 20($sp) #save the address
sw $s0, 16($sp) #save the first argument testarray
add $s0, $0, $a0
sw $s1, 12($sp) #save the second argument numelements
add $s1, $0, $a1
sw $s2, 8($sp) #save the third argument cntType 1 or -1
add $s2, $0, $a2
sw $s3, 4($sp) #space for i
addi $s3, $s1, -1 #i=numelements-1
sw $s4, 0($sp) #space for cnt
addi $s4, $0, 0 #cnt=0

forloop:
slti $t0, $s3, 1 #break condition i<1
bne $t0, $0, break1 # break
sll $t0, $s3, 2 #t0=4*i
add $t0, $t0, $s0
lw $a0, 0($t0) #the above process is going to load A[i]
addi $t0, $0, 1 #t0=1
bne $t0, $s2, otherwise #whether cntType=1
jal Pass #call pass function
add $t0, $t0, $0 #delay
j endswitch

otherwise:

```

```

        jal Fail #call fail function
        add $t0, $t0,$0 #delay
endswitch:
        add $t0, $t0 $0          #delay
        add $s4, $s4, $v0 #change cnt
        addi $s3, $s3, -1 #i--
        j forloop
break1:
        add $v0,$0,$s4
        lw $s4, 0($sp)
        lw $s3, 4($sp)
        lw $s2, 8($sp)
        lw $s1, 12($sp)
        lw $s0, 16($sp)
        lw $ra, 20($sp)
        addi $sp, $sp, 24 #restore the stack
        jr $ra #return
        add $t0, $t0,$0 #delay
Pass:
        addi $sp, $sp, -12 #create 3 spaces for stack for pass function
        sw $ra, 8($sp)
        sw $s0, 4($sp) #save x
        add $s1,$a0,$0
        sw $s1, 0($sp)
        addi $t1,$0,1 #t1=1
        addi $s1,$0,0 #s1=0
        add $s0, $0, $a0
        slti $t0, $s0, 60 #x<60
        beq $t0,$t1 passreturn #return s1=0
        add $t0, $t0,$0 #delay
        addi $s1,$0,1 #return s1=1
passreturn:
        add $v0, $0,$s1
        lw $s1, 0($sp)
        lw $s0, 4($sp)
        lw $ra, 8($sp)
        addi $sp, $sp, 12 #restore the stack
        jr $ra #return
        add $t0, $t0,$0 #delay
Fail:
        addi $sp, $sp, -12 #create 3 spaces for stack for fail function
        sw $ra, 8($sp)
        sw $s0, 4($sp) #save x
        sw $s1, 0($sp)
        add $s1,$a0,$0
        addi $t1,$0,1 #t1=0
        addi $s1,$0,0 #s1=0
        add $s0, $0, $a0
        slti $t0, $s0, 60 #x<60
        bne $t0,$t1 failreturn #return s1=0
        add $t0, $t0,$0 #delay
        addi $s1,$0,1 #return s1=1
failreturn:
        add $v0, $0,$s1
        lw $s1, 0($sp)
        lw $s0, 4($sp)
        lw $ra, 8($sp)
        addi $sp, $sp, 12 #restore the stack

```

```
jr $ra #return  
add $t0, $t0,$0 #delay
```