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.

```
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>0, 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;
}
```

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.

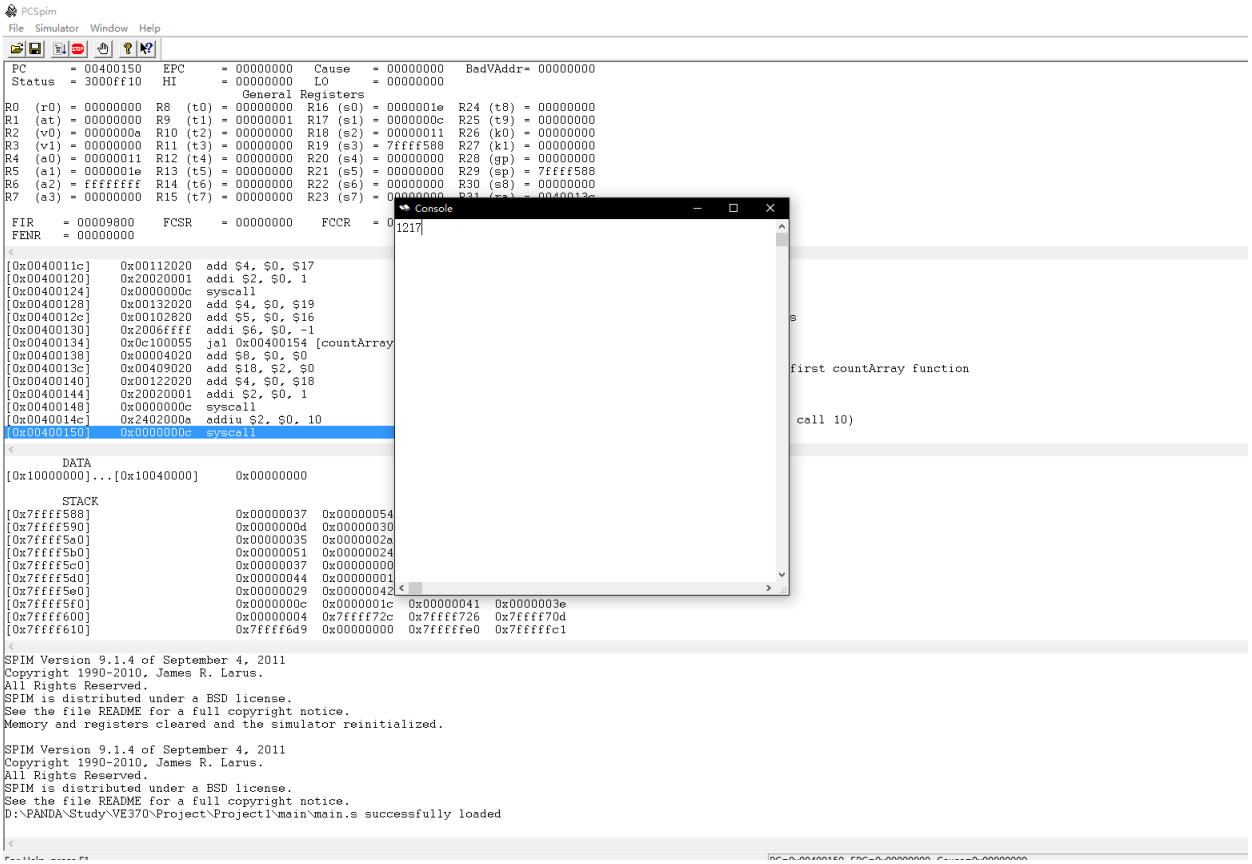
```
main() {
    srand(time(NULL));
    int size = 31;  //determine the size of the array here
    int PassCnt, FailCnt;
    int testArray[size];
    for(int i=0;i<size;i++){
        testArray[i]=rand()%100;
        cout<<testArray[i]<<" ";
    }
    PassCnt = countArray(testArray, size, 1);
    FailCnt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



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 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 PCSpim 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 1
        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 1
        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
breakl:
        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
```