VE370 Introduction to Computer Organization Project 1 Report

Xinhao Liao 516370910037

October 2018

1 Introduction

MIPS (as an acronym for Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture developed by MIPS Technologies. The MIPS instruction set is studied in VE370 for a better knowledge of the organization and design of computer. In project 1, an MIPS assembly program that operates on a data segment consisting of an array of 32-bit unsigned integers is asked to be developed based on some given codes in C (which can be found in Appendix A). The program is assembled, simulated, and carefully commented with PCSpim.

2 Background

The program, with its C code in Appendix A, can count the number of given marks that pass and fail.

Initially in the program, 20 elements of marks in the range of [0,100] are given. They are orderly 55, 83, 55, 76, 45, 98, 77, 21, 90, 61, 82, 49, 73, 22, 86, 60, 59, 0, 100, 11. Among them, 11 elements (including 83, 76, 98, 77, 90, 61, 82, 73, 86, 60, 100) are greater than or equal to 60, which pass. And other 9 elements (including 55, 55, 45, 21, 49, 22, 59, 0, 11) are less than 60, which fail.

For the result of the program, we notice that in the C program given in Appendix A, in the main function there are variables *size, testArray, PassCnt,* and *FailCnt*. *size* represents the size of the array, which should be 20_2, which is 14_{16}. *testArray* represents the address of the array of 20 elements, and the value of it is uncertain. *PassCnt* represents the number of marks that pass, which should be 11_2, which is B_{16}. *FailCnt* represents the number of marks that pass, which should be 9_2, which is 9_{16}.

3 Simulation result and analysis

```
 PC      = 00400000   EPC     = 00000000   Cause   = 00000000   BadVAddr= 00000000
 Status  = 3000ff10   HI      = 00000000   LO      = 00000000
                        General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000000  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000000  R17 (s1) = 00000000  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 00000000  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 00000000  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000000  R13 (t5) = 00000000  R21 (s5) = 00000000  R29 (sp) = 7fffeffc
R6  (a2) = 00000000  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 00000000

 FIR     = 00009800    FCSR    = 00000000    FCCR    = 00000000   FEXR    = 00000000
<
[0x00400000]    0x24100014  addiu $16, $0, 20              ; 6: addiu  $s0, $0, 20     # set size to be 20 and save it in $s0
[0x00400004]    0x23bdffb0  addi $29, $29, -80             ; 7: addi   $sp, $sp,-80    # initialize the array with 20 elements, make room in stack
[0x00400008]    0x001d8820  add $17, $0, $29               ; 8: add    $s1, $0, $sp    # the base address of the array
[0x0040000c]    0x24080037  addiu $8, $0, 55               ; 9: addiu  $t0, $0, 55     # initialize the 20 elements and respectively save them in the room in stack
[0x00400010]    0xae280000  sw $8, 0($17)                  ; 10: sw    $t0, 0($s1)
[0x00400014]    0x24090053  addiu $9, $0, 83               ; 11: addiu $t1, $0, 83
[0x00400018]    0xae290004  sw $9, 4($17)                  ; 12: sw    $t1, 4($s1)
[0x0040001c]    0x240a0037  addiu $10, $0, 55              ; 13: addiu $t2, $0, 55
[0x00400020]    0xae2a0008  sw $10, 8($17)                 ; 14: sw    $t2, 8($s1)
[0x00400024]    0x240b004c  addiu $11, $0, 76              ; 15: addiu         $t3, $0, 76
[0x00400028]    0xae2b000c  sw $11, 12($17)                ; 16: sw    $t3, 12($s1)
[0x0040002c]    0x240c002d  addiu $12, $0, 45              ; 17: addiu         $t4, $0, 45
[0x00400030]    0xae2c0010  sw $12, 16($17)                ; 18: sw    $t4, 16($s1)
<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000
[0x7fff8000]...[0x80000000]     0x00000000

        KERNEL DATA
[0x90000000]...[0x90010000]     0x00000000
```

Figure 1: Initial state.

The assembly file *main.s* which implements the function *main()* and its subroutines are shown in Appendix B. The program is simulated with PCSpim.

The initial state of the program is shown in Figure 1. As we can see, there are no data stored on Stack, or in Data area. And for the registers, the value in $\$sp$ is $7fffeffc_{16}$ which denotes the address for next data on stack. And all the other values of registers are initially 0.

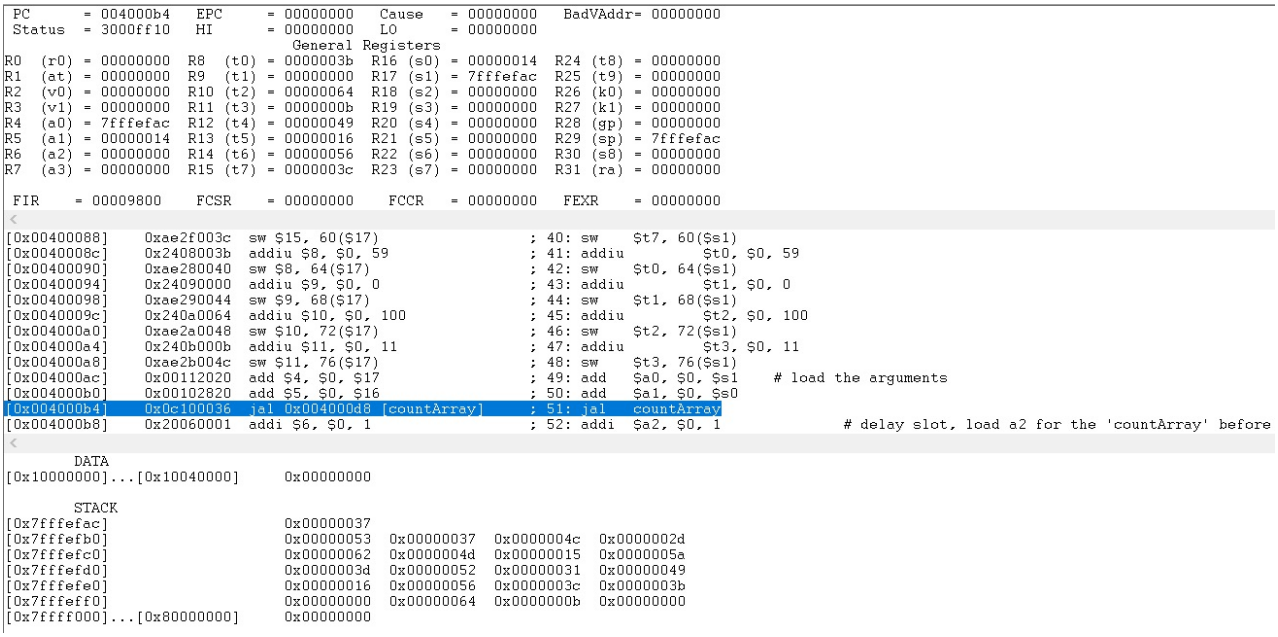

Figure 2: Before the first calling of *countArray*.

Then after initializing *size* and the array *testArray* with 20 elements and before *countArray* is called, the state is screenshot and shown in Figure 2. As we can see in Figure 2, there are 20 values on stack corresponding to the 20 elements of array. $\$sp$ has been updated to $7fffefac_{16}$ which is $50_{16} = 80_2$ smaller than $7fffeffc_{16}$ shown in Figure 1. This means that $80/4 = 20$ elements are stored on stack. And $\$s0$ has been updated to $14_{16} = 20_2$ representing the value of *size*. $\$s1$ has been updated to $7fffeffc_{16}$ representing the value of the base address of *testArray*. $\$a0$ and $\$a1$ has been updated as the arguments for the first calling of *countArray*. And $\$a2$ as another argument will be updated just before the calling since the instruction "addi $a2, $0, 1" is in the delay slot following "jal countArray".

The state before entering the loop in *countArray(testArray, size, 1)* is screenshot and shown in Figure 3. As shown in the figure, before the loop, $\$t0$ is initialized to be 0, representing *cnt*, $\$t1$ is initialized to be $numElements - 1 = 20 - 1 = 19_2 = 13_{16}$. Then we compare i with 0 using "slt $t2, $t1, $0" and have $\$t2 = 0$ since $i >= 0$. This will lead to jumping to the label *Loop* with "beq $t2, $0, Loop". Also notice that $a2 has already been updated to 1, resulted from the instruction in the delay slot mentioned above. And $ra has been updated to $004000bc_{16}$ resulted from the "jal countArray" instruction mentioned above. Observing Figure 2, we can see that the address of the instruction "jal countArray" is $004000b4_{16}$ and the following delay slot is at the address $004000b8_{16}$, and so the value of $ra, $004000bc_{16}$, is the address of the instruction following the delay slot. After the procedure *countArray* finishes, the program can jump back to that address. All the other values remain the same.

The state before calling *Pass(A[19])* in the first calling of *countArray* is screenshot and shown in Figure 4. At this state, $t3 has been changed to be *A[i=19]*, and the vale in $a0 has been stored on stack, $sp has been updated to be $7fffefa8_{16}$, which means the room for the newly stored value of $a0 has been made. And $t2 has been set to be 1 to compare with $a2, representing *cntType*. Since $\$a2 = \$t2 = 1$, the program will go to label *L2*, representing the procedure *Pass*. But before that, the instruction in the delay slot following the "beq" instruction will be executed first, which will update $a0 to be *A[i=19]*.

The state before entering the second iteratrion in the first calling of *countArray* is screenshot and shown in Figure 5. At this state, $t4 has been updated to 1 since the last element of the array is 11, which is smaller than 60. Since $\$t4 \neq 0$, the program doesn't directly jump to label *L3*. The result to be returned, $v0, is

```
 PC      = 004000e4   EPC     = 00000000   Cause   = 00000000   BadVAddr= 00000000
 Status  = 3000ff10   HI      = 00000000   LO      = 00000000
                          General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000014  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000013  R17 (s1) = 7fffefac  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 0000000b  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7fffefac  R12 (t4) = 00000049  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000014  R13 (t5) = 00000016  R21 (s5) = 00000000  R29 (sp) = 7fffefac
R6  (a2) = 00000001  R14 (t6) = 00000056  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 0000003c  R23 (s7) = 00000000  R31 (ra) = 004000bc

 FIR    = 00009800    FCSR    = 00000000    FCCR    = 00000000   FEXR    = 00000000
<
[0x004000c8]    0x0c100036  jal 0x004000d8 [countArray]     ; 56: jal   countArray
[0x004000cc]    0x2006ffff  addi $6, $0, -1                 ; 57: addi  $a2, $0, -1     # delay slot, load a2 for the 'countArray' before
[0x004000d0]    0x00029820  add $19, $0, $2                 ; 58: add   $s3, $0, $v0    # save the result of countArray in $s3, this is 'FailCnt'
[0x004000d4]    0x08100052  j 0x00400148 [Exit]             ; 59: j     Exit
[0x004000d8]    0x00004020  add $8, $0, $0                  ; 61: add   $t0, $0, $0     # initialize cnt=0
[0x004000dc]    0x20a9ffff  addi $9, $5, -1                 ; 62: add   $t1, $a1, -1    # initialize i=numElements-1
[0x004000e0]    0x0120502a  slt $10, $9, $0                 ; 64: slt   $t2, $t1, $0    # $t2=1 if i<0
[0x004000e4]    0x11400002  beq $10, $0, 8 [Loop-0x004000e4]; 65: beq   $t2, $0, Loop   # go to Loop if i>=0
[0x004000e8]    0x00081020  add $2, $0, $8                  ; 66: add   $v0, $0, $t0    # set cnt as the result to be returned
[0x004000ec]    0x03e00008  jr $31                          ; 67: jr    $ra             # return to calling routine
[0x004000f0]    0x00095880  sll $11, $9, 2                  ; 69: sll   $t3, $t1, 2     # $t3 = i * 4
[0x004000f4]    0x01645820  add $11, $11, $4                ; 70: add   $t3, $t3, $a0   # $t3 = A + i * 4, the address of A[i]
[0x004000f8]    0x8d6b0000  lw $11, 0($11)                  ; 71: lw    $t3, 0($t3)     # $t3 = A[i]
<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffefac]                    0x00000037
[0x7fffefb0]                    0x00000053  0x00000037  0x0000004c  0x0000002d
[0x7fffefc0]                    0x00000062  0x0000004d  0x00000015  0x0000005a
[0x7fffefd0]                    0x0000003d  0x00000052  0x00000031  0x00000049
[0x7fffefe0]                    0x00000016  0x00000056  0x0000003c  0x0000003b
[0x7fffeff0]                    0x00000000  0x00000064  0x0000000b  0x00000000
[0x7ffff000]...[0x80000000]     0x00000000
```

Figure 3: Before the loop of the first calling of *countArray*.

```
 PC      = 00400108   EPC     = 00000000   Cause   = 00000000   BadVAddr= 00000000
 Status  = 3000ff10   HI      = 00000000   LO      = 00000000
                          General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000014  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000013  R17 (s1) = 7fffefac  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000001  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 0000000b  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7fffefac  R12 (t4) = 00000049  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000014  R13 (t5) = 00000016  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000001  R14 (t6) = 00000056  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 0000003c  R23 (s7) = 00000000  R31 (ra) = 004000bc

 FIR    = 00009800    FCSR    = 00000000    FCCR    = 00000000   FEXR    = 00000000
<
[0x004000e0]    0x0120502a  slt $10, $9, $0                 ; 64: slt   $t2, $t1, $0    # $t2=1 if i<0
[0x004000e4]    0x11400002  beq $10, $0, 8 [Loop-0x004000e4]; 65: beq   $t2, $0, Loop   # go to Loop if i>=0
[0x004000e8]    0x00081020  add $2, $0, $8                  ; 66: add   $v0, $0, $t0    # set cnt as the result to be returned
[0x004000ec]    0x03e00008  jr $31                          ; 67: jr    $ra             # return to calling routine
[0x004000f0]    0x00095880  sll $11, $9, 2                  ; 69: sll   $t3, $t1, 2     # $t3 = i * 4
[0x004000f4]    0x01645820  add $11, $11, $4                ; 70: add   $t3, $t3, $a0   # $t3 = A + i * 4, the address of A[i]
[0x004000f8]    0x8d6b0000  lw $11, 0($11)                  ; 71: lw    $t3, 0($t3)     # $t3 = A[i]
[0x004000fc]    0x23bdfffc  addi $29, $29, -4               ; 72: addi  $sp, $sp, -4    # make room on stack for 1 register
[0x00400100]    0xafa40000  sw $4, 0($29)                   ; 73: sw    $a0, 0($sp)     # save $a0 on stack
[0x00400104]    0x200a0001  addi $10, $0, 1                 ; 74: addi  $t2, $0, 1              # $t2 = 1
[0x00400108]    0x10ca0006  beq $6, $10, 24 [L2-0x00400108] ; 75: beq   $a2, $t2, L2    # if cntType==1, go to L2(Pass)
[0x0040010c]    0x01602020  add $4, $11, $0                 ; 76: add   $a0, $t3, $0    # delay slot, load $a0 for 'Pass'
[0x00400110]    0x288c003c  slti $12, $4, 60                ; 78: slti  $t4, $a0, 60    # if x<60, $t4=1, else $t4=0
<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffefa8]                    0x7fffefac  0x00000037
[0x7fffefb0]                    0x00000053  0x00000037  0x0000004c  0x0000002d
[0x7fffefc0]                    0x00000062  0x0000004d  0x00000015  0x0000005a
[0x7fffefd0]                    0x0000003d  0x00000052  0x00000031  0x00000049
[0x7fffefe0]                    0x00000016  0x00000056  0x0000003c  0x0000003b
[0x7fffeff0]                    0x00000000  0x00000064  0x0000000b  0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
```

Figure 4: Before *Pass(A[19])* in the first calling of *countArray*.

finally updated to be 0. Then $a0 is restored from the stack, $t1, representing i, decreases by 1, and $t0, representing *cnt*, increases by the result in $v0, which is 0 in label *L3*. Before jumping back to label *L1* for next iteration, $sp wiil firstly increase by 4, which means the room for previously restored value of $a0 has been freed. This instruction is in the delay slot.

In the following iterations, similar instructions will loop again and again. The state after the first calling of

```
 PC       = 00400140   EPC      = 00000000   Cause    = 00000000   BadVAddr= 00000000
 Status   = 3000ff10   HI       = 00000000   LO       = 00000000
                          General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000014  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000012  R17 (s1) = 7fffefac  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000001  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 0000000b  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7fffefac  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000014  R13 (t5) = 00000016  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000001  R14 (t6) = 00000056  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 0000003c  R23 (s7) = 00000000  R31 (ra) = 004000bc

 FIR    = 00009800    FCSR    = 00000000    FCCR    = 00000000    FEXR    = 00000000
<
[0x00400118]    0x20020000  addi $2, $0, 0                ; 80: addi  $v0, $0, 0              # delay slot, if x>=60, the result to be returned is 0
[0x0040011c]    0x20020001  addi $2, $0, 1                ; 81: addi  $v0, $0, 1              # if x<60, the result to be returned is 1
[0x00400120]    0x0810004d  j 0x00400134 [L3]             ; 82: j     L3              # jump to L3 to return
[0x00400124]    0x288c003c  slti $12, $4, 60              ; 85: slti  $t4, $a0, 60    # if x<60, $t4=1, else $t4=0
[0x00400128]    0x11800002  beq $12, $0, 8 [L3-0x00400128]  ; 86: beq   $t4, $0, L3     # if x>=60, go to L3 to return
[0x0040012c]    0x20020001  addi $2, $0, 1                ; 87: addi  $v0, $0, 1              # delay slot, if x>=60, the result to be returned is 1
[0x00400130]    0x20020000  addi $2, $0, 0                ; 88: addi  $v0, $0, 0              # if x<60, the result to be returned is 0
[0x00400134]    0x8fa40000  lw $4, 0($29)                 ; 90: lw    $a0, 0($sp)     # restore $a0 from stack
[0x00400138]    0x2129ffff  addi $9, $9, -1               ; 91: add   $t1, $t1, -1    # i--
[0x0040013c]    0x01024020  add $8, $8, $2                ; 92: add   $t0, $t0, $v0   # cnt += $v0, where $v0 is the result of Pass(A[i]) or Fail(A[i])
[0x00400140]    0x08100038  j 0x004000e0 [L1]             ; 93: j     L1              # jump back to L1 to Loop again
[0x00400144]    0x23bd0004  addi $29, $29, 4              ; 94: addi  $sp, $sp, 4     # delay slot, adjust stack to delete 1 item, completed before j L1
[0x00400148]    0x24020000  addiu $2, $0, 0               ; 96: addiu         $v0, $0, 0              # Prepare to exit (system call 0)
<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffefa8]                    0x7fffefac  0x00000037
[0x7fffefb0]                    0x00000053  0x00000037  0x0000004c  0x0000002d
[0x7fffefc0]                    0x00000062  0x0000004d  0x00000015  0x0000005a
[0x7fffefd0]                    0x0000003d  0x00000052  0x00000031  0x00000049
[0x7fffefe0]                    0x00000016  0x00000056  0x0000003c  0x0000003b
[0x7fffeff0]                    0x00000000  0x00000064  0x0000000b  0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
```

Figure 5: Before entering the second iteration of the loop.

```
 PC       = 004000ec   EPC      = 00000000   Cause    = 00000000   BadVAddr= 00000000
 Status   = 3000ff10   HI       = 00000000   LO       = 00000000
                          General Registers
R0  (r0) = 00000000  R8  (t0) = 0000000b  R16 (s0) = 00000014  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = ffffffff  R17 (s1) = 7fffefac  R25 (t9) = 00000000
R2  (v0) = 0000000b  R10 (t2) = 00000001  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 00000037  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7fffefac  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000014  R13 (t5) = 00000016  R21 (s5) = 00000000  R29 (sp) = 7fffefac
R6  (a2) = 00000001  R14 (t6) = 00000056  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 0000003c  R23 (s7) = 00000000  R31 (ra) = 004000bc

 FIR    = 00009800    FCSR    = 00000000    FCCR    = 00000000    FEXR    = 00000000
<
[0x004000e0]    0x0120502a  slt $10, $9, $0                 ; 64: slt   $t2, $t1, $0    # $t2=1 if i<0
[0x004000e4]    0x11400002  beq $10, $0, 8 [Loop-0x004000e4]; 65: beq   $t2, $0, Loop   # go to Loop if i>=0
[0x004000e8]    0x00081020  add $2, $0, $8                  ; 66: add   $v0, $0, $t0    # set cnt as the result to be returned
[0x004000ec]    0x03e00008  jr $31                          ; 67: jr    $ra             # return to calling routine
[0x004000f0]    0x00095880  sll $11, $9, 2                  ; 69: sll   $t3, $t1, 2     # $t3 = i * 4
[0x004000f4]    0x01645820  add $11, $11, $4                ; 70: add   $t3, $t3, $a0   # $t3 = A + i * 4, the address of A[i]
[0x004000f8]    0x8d6b0000  lw $11, 0($11)                  ; 71: lw    $t3, 0($t3)     # $t3 = A[i]
[0x004000fc]    0x23bdfffc  addi $29, $29, -4               ; 72: addi  $sp, $sp, -4    # make room on stack for 1 register
[0x00400100]    0xafa40000  sw $4, 0($29)                   ; 73: sw    $a0, 0($sp)     # save $a0 on stack
[0x00400104]    0x200a0001  addi $10, $0, 1                 ; 74: addi  $t2, $0, 1              # $t2 = 1
[0x00400108]    0x10ca0006  beq $6, $10, 24 [L2-0x00400108] ; 75: beq   $a2, $t2, L2    # if cntType==1, go to L2(Pass)
[0x0040010c]    0x01602020  add $4, $11, $0                 ; 76: add   $a0, $t3, $0    # delay slot, load $a0 for 'Pass'
[0x00400110]    0x288c003c  slti $12, $4, 60                ; 78: slti  $t4, $a0, 60    # if x<60, $t4=1, else $t4=0
<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffefac]                    0x00000037
[0x7fffefb0]                    0x00000053  0x00000037  0x0000004c  0x0000002d
[0x7fffefc0]                    0x00000062  0x0000004d  0x00000015  0x0000005a
[0x7fffefd0]                    0x0000003d  0x00000052  0x00000031  0x00000049
[0x7fffefe0]                    0x00000016  0x00000056  0x0000003c  0x0000003b
[0x7fffeff0]                    0x00000000  0x00000064  0x0000000b  0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
```

Figure 6: Before exiting the first calling of *countArray* and jumping back.

countArray and just before jumping back is screen shot and shown in Figure 6. As we can see, $t0, representing *cnt* has been updated to be $b_{16} = 11_2$ and its value has been copied to $v0 as the result. $t1, representing i, has reached $ffffffff_{16} = -1_2$. Next, the program will jump back to the address stored in $ra.

Then the result in $v0 will be copied to $s3 as the value of *PassCnt*. And *countArray* will be called second time with $cntType = -1$ to obtain *FailCnt* in a similar way. After that, the program will reach its end and jump to label *Exit*. The screen shot for the final state is shown in Figure 7. As we can see, $s0, representing

```
 PC       = 00400150   EPC      = 00000000   Cause    = 00000000   BadVAddr= 00000000
 Status   = 3000ff10   HI       = 00000000   LO       = 00000000
                          General Registers
R0  (r0) = 00000000  R8  (t0) = 00000009  R16 (s0) = 00000014  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = ffffffff  R17 (s1) = 7fffefac  R25 (t9) = 00000000
R2  (v0) = 0000000a  R10 (t2) = 00000001  R18 (s2) = 0000000b  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = fffffffc  R19 (s3) = 00000009  R27 (k1) = 00000000
R4  (a0) = 7fffefac  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000014  R13 (t5) = 00000016  R21 (s5) = 00000000  R29 (sp) = 7fffeffc
R6  (a2) = ffffffff  R14 (t6) = 00000056  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 0000003c  R23 (s7) = 00000000  R31 (ra) = 004000d0

 FIR     = 00009800    FCSR     = 00000000    FCCR    = 00000000    FEXR     = 00000000
<
[0x00400120]    0x20020001  addi $2, $0, 1                ; 82: addi  $v0, $0, 1              # if x<60, the result to be returned is 1
[0x00400124]    0x0810004e  j 0x00400138 [L3]             ; 83: j     L3              # jump to L3 to return
[0x00400128]    0x288c003c  slti $12, $4, 60              ; 86: slti  $t4, $a0, 60    # if x<60, $t4=1, else $t4=0
[0x0040012c]    0x11800002  beq $12, $0, 8 [L3-0x0040012c]  ; 87: beq   $t4, $0, L3     # if x>=60, go to L3 to return
[0x00400130]    0x20020001  addi $2, $0, 1                ; 88: addi  $v0, $0, 1              # delay slot, if x>=60, the result to be returned is 1
[0x00400134]    0x20020000  addi $2, $0, 0                ; 89: addi  $v0, $0, 0              # if x<60, the result to be returned is 0
[0x00400138]    0x8fa40000  lw $4, 0($29)                 ; 91: lw    $a0, 0($sp)     # restore $a0 from stack
[0x0040013c]    0x2129ffff  addi $9, $9, -1               ; 92: add   $t1, $t1, -1    # i--
[0x00400140]    0x01024020  add $8, $8, $2                ; 93: add   $t0, $t0, $v0   # cnt += $v0, where $v0 is the result of Pass(A[i]) or Fail(A[i])
[0x00400144]    0x08100039  j 0x004000e4 [L1]             ; 94: j     L1              # jump back to L1 to Loop again
[0x00400148]    0x23bd0004  addi $29, $29, 4              ; 95: addi  $sp, $sp, 4     # delay slot, adjust stack to delete 1 item, completed before j L1
[0x0040014c]    0x2402000a  addiu $2, $0, 10              ; 97: addiu         $v0, $0, 10     # Prepare to exit (system call 0)
[0x00400150]    0x0000000c  syscall                       ; 98: syscall                       # Exit
<
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
[0x90000000]...[0x90010000]     0x00000000
```

Figure 7: Final state.

size is $14_{16} = 20_2$. And $s1 represent the base address of *testArray*, but actually the array has been released, and there are now no values on stack. And $sp has been restored to its original value. $s2 = $b_{16} = 11_2$. This represents *PassCnt*. And $s3 = $9_{16} = 9_2$. This represents *FailCnt*.

4 Appendix A. C code of the program

main.c

```
main() {
int size = 20;  //determine the size of the array here
int PassCnt, FailCnt;
int testArray[size] = { 55, 83,
    55, 76, 45, 98, 77, 21, 90, 61, 82, 49, 73, 22, 86, 60, 59, 0, 100, 11
    //compose 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;
}
```

5 Appendix B. Assembly code of the program

main.s

```
#main.s
 .text
 .globl __start
__start:
main:
addiu $s0, $0, 20     # set size to be 20 and save it in $s0
addi  $sp, $sp,-80    # initialize the array with 20 elements, make room in stack
add $s1, $0, $sp      # the base address of the array
addiu $t0, $0, 55     # initialize the 20 elements and respectively save them on stack
sw  $t0, 0($s1)
addiu $t1, $0, 83
sw  $t1, 4($s1)
addiu $t2, $0, 55
sw  $t2, 8($s1)
addiu   $t3, $0, 76
sw  $t3, 12($s1)
addiu   $t4, $0, 45
sw  $t4, 16($s1)
addiu   $t5, $0, 98
sw  $t5, 20($s1)
addiu   $t6, $0, 77
sw  $t6, 24($s1)
addiu   $t7, $0, 21
sw  $t7, 28($s1)
addiu   $t0, $0, 90
sw  $t0, 32($s1)
addiu   $t1, $0, 61
sw  $t1, 36($s1)
addiu   $t2, $0, 82
sw  $t2, 40($s1)
addiu   $t3, $0, 49
sw  $t3, 44($s1)
addiu   $t4, $0, 73
sw  $t4, 48($s1)
addiu   $t5, $0, 22
sw  $t5, 52($s1)
addiu   $t6, $0, 86
sw  $t6, 56($s1)
addiu   $t7, $0, 60
sw  $t7, 60($s1)
addiu   $t0, $0, 59
sw  $t0, 64($s1)
addiu   $t1, $0, 0
sw  $t1, 68($s1)
addiu   $t2, $0, 100
sw  $t2, 72($s1)
addiu   $t3, $0, 11
sw  $t3, 76($s1)
add $a0, $0, $s1  # load the arguments
add $a1, $0, $s0
jal countArray
addi  $a2, $0, 1  # delay slot, load a2 for the 'countArray' before
add $a1, $0, $s0  # load the arguments
add $s2, $0, $v0  # save the result of countArray in $s2, this is 'PassCnt'
add $a0, $0, $s1
jal countArray
```

```
57 addi  $a2, $0, -1 # delay slot, load a2 for the 'countArray' before
58 add $s3, $0, $v0  # save the result of countArray in $s3, this is 'FailCnt'
59 j Exit
60 addi  $sp, $sp, 80  # delay slot, release the room for the array stored on stack
61 countArray:
62 add $t0, $0, $0   # initialize cnt=0
63 add $t1, $a1, -1  # initialize i=numElements-1
64 L1:
65 slt $t2, $t1, $0  # $t2=1 if i<0
66 beq $t2, $0, Loop # go to Loop if i>=0
67 add $v0, $0, $t0  # set cnt as the result to be returned
68 jr  $ra           # return to calling routine
69 Loop:
70 sll $t3, $t1, 2   # $t3 = i * 4
71 add $t3, $t3, $a0 # $t3 = A + i * 4, the address of A[i]
72 lw  $t3, 0($t3)   # $t3 = A[i]
73 addi $sp, $sp, -4 # make room on stack for 1 register
74 sw  $a0, 0($sp)   # save $a0 on stack
75 addi  $t2, $0, 1  # $t2 = 1
76 beq $a2, $t2, L2  # if cntType==1, go to L2(Pass)
77 add $a0, $t3, $0  # delay slot, load $a0 for 'Pass'
78 Fail:
79 slti  $t4, $a0, 60  # if x<60, $t4=1, else $t4=0
80 beq $t4, $0, L3     # if x>=60, go to L3 to return
81 addi  $v0, $0, 0    # delay slot, if x>=60, the result to be returned is 0
82 addi  $v0, $0, 1    # if x<60, the result to be returned is 1
83 j L3                # jump to L3 to return
84 L2:
85 Pass:
86 slti  $t4, $a0, 60  # if x<60, $t4=1, else $t4=0
87 beq $t4, $0, L3     # if x>=60, go to L3 to return
88 addi  $v0, $0, 1    # delay slot, if x>=60, the result to be returned is 1
89 addi  $v0, $0, 0    # if x<60, the result to be returned is 0
90 L3:
91 lw  $a0, 0($sp)     # restore $a0 from stack
92 add $t1, $t1, -1    # i--
93 add $t0, $t0, $v0   # cnt += $v0, where $v0 is the result of Pass(A[i]) or Fail(A[i])
94 j L1                # jump back to L1 to Loop again
95 addi  $sp, $sp, 4   # delay slot, adjust stack to delete 1 item, completed before j L1
96 Exit:
97 addiu   $v0, $0, 10 # Prepare to exit (system call 10)
98 syscall             # Exit
```