VE370 Introduction to Computer Organization

Project 1

Wu Jiayao,吴佳遥,517370910257

1. Obejective

Develop a MIPS assembly program that operates on a data segment consisting of an array of 32-bit signed integers.

For this project, the MIPS program is developed from a C program that counts the number of positive integers, negative integers and 0 from an customized array whose size is more than 20.

2. Arrangement of the registers

Name	Usage	Name	Usage
t0	register for delay operations	a0	The first arg of a function: numArray; A[i]
t1	CntType	a1	The second arg of a function: numElements
t2	register for temporarily storage	a2	The third arg of a function:cntType
t3	i in countArray(int*,int,int)	s0	int size = 21 for this work
t4	register for storing bool temps in Pos(int),Neg(int),Zero(int)	s1	int PosCnt
t5	numArray[i]	s2	int NegCnt
t9	cnt in countArray(int*,int,int)	s3	int ZeroCnt

3. Project Customization and Comments

- Detail comments follows the source code in Appendix.
- The array for this project is generated by Python. Given as [14 -15 0 -13 -8 16 30 -38 24 0 18 14 -30 -13 -2 17 0 -27 11 -30 25]. Source code is provided in Appendix.
- For this project:

```
int size =21;
//Result
PosCnt = 9;
NegCnt = 9;
ZeroCnt = 3;
```

4. Before calling countArray(int*,int,int)

```
int main()
{
    int size = 21; //determine the size of the array here
    int PosCnt, NegCnt, ZeroCnt;
    int testArray[21] = {
14,-15,0,-13,-8,16,30,-38,24,0,18,14,-30,-13,-2,17,0,-27,11,-30,25};
    ....
}

```

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000019  R16 (s0) = 00000015  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) = 7fffefa8  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = 00000000  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000001  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
```

```
[0x0040009c]    0x2008ffe5  addi $8, $0, -27          ; 45: addi $t0, $0, -27 # store 18th temporarily in $t0
[0x004000a0]    0xafa80044  sw $8, 68($29)             ; 46: sw $t0, 68($sp) #store in testArray[17]
[0x004000a4]    0x2008000b  addi $8, $0, 11           ; 47: addi $t0, $0, 11 # store 19th temporarily in $t0
[0x004000a8]    0xafa80048  sw $8, 72($29)             ; 48: sw $t0, 72($sp) #store in testArray[18]
[0x004000ac]    0x2008ffe2  addi $8, $0, -30          ; 49: addi $t0, $0, -30 # store 20th temporarily in $t0
[0x004000b0]    0xafa8004c  sw $8, 76($29)             ; 50: sw $t0, 76($sp) #store in testArray[19]
[0x004000b4]    0x20080019  addi $8, $0, 25           ; 51: addi $t0, $0, 25 # store 21st temporarily in $t0
[0x004000b8]    0xafa80050  sw $8, 80($29)             ; 52: sw $t0, 80($sp) #store in testArray[20]
[0x004000bc]    0x001d2021  addu $4, $0, $29           ; 54: addu $a0,$0,$sp # a0 = testArray
[0x004000c0]    0x00102821  addu $5, $0, $16           ; 55: addu $a1,$0,$s0 # a1 = size
[0x004000c4]    0x20060001  addi $6, $0, 1            ; 56: addi $a2,$0,1 # a2 = 1
*[0x004000c8]    0x0c100042  jal 0x00400108 [countArray]    ; 57: jal countArray # $v0 = countArray(testArray, size, 1)
[0x004000cc]    0x20080001  addi $8, $0, 1            ; 58: addi $t0, $0, 1  # wait for delay
```

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

          STACK
[0x7fffefa8]                         0x0000000e  0xffffff1
[0x7fffefb0]                         0x00000000  0xffffff3  0xffffff8  0x00000010
[0x7fffefc0]                         0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffefd0]                         0x00000012  0x0000000e  0xffffffe2  0xffffff3
[0x7fffefe0]                         0xffffffe  0x00000011  0x00000000  0xffffffe5
[0x7fffeff0]                         0x0000000b  0xffffffe2  0x00000019  0x00000000
[0x7ffff000]...[0x80000000]    0x00000000
```

Line 4 to Line 52 in project1.s.

The testArray is stored in 0-80th address of the stack $$sp$$.

size is stored in $$s0$$.

PosCnt is stored in $$s1$$.

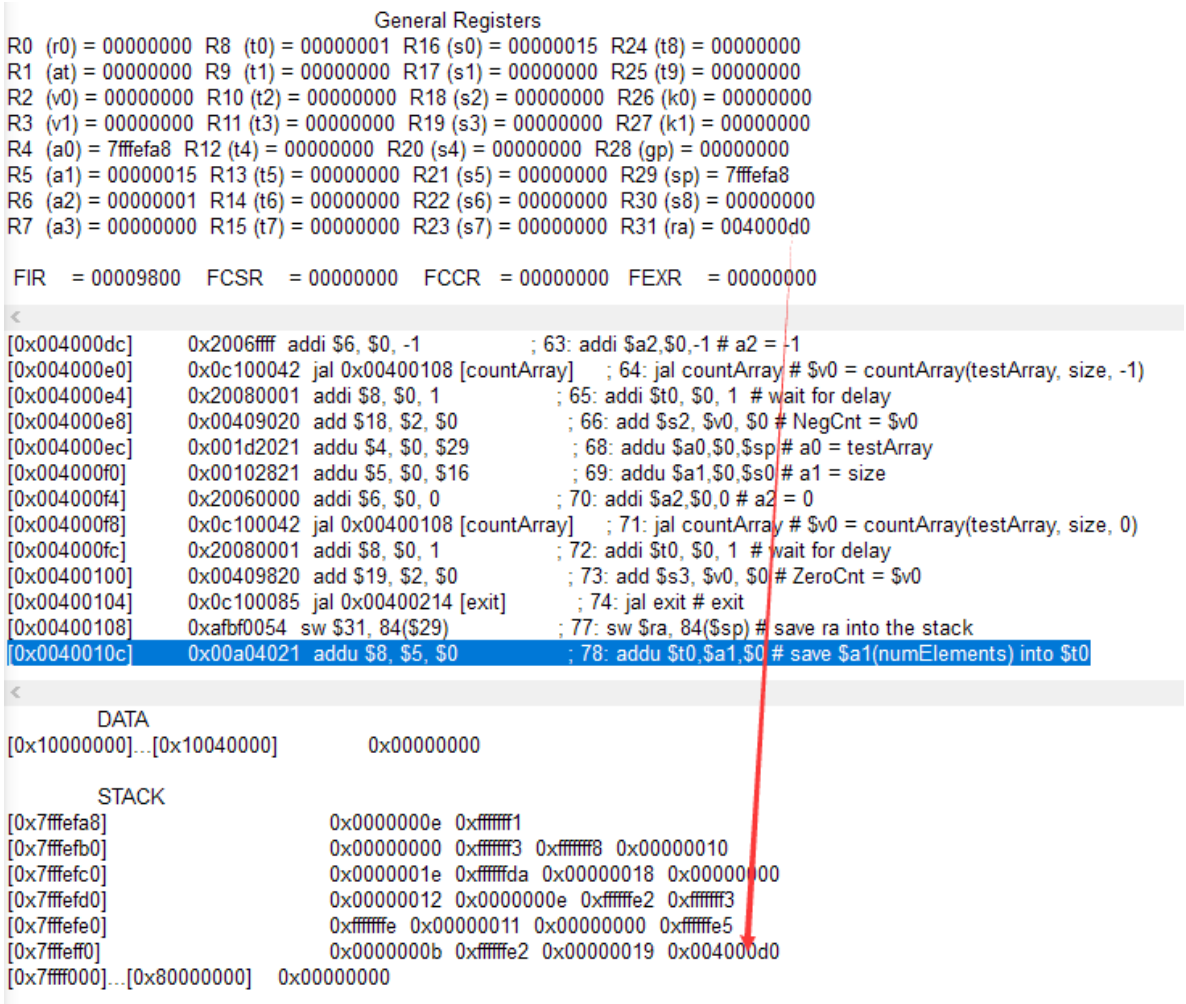
NegInt is stored in $$s2$$.

ZeroCnt is stored in $s3

5. Inside countArray(int*,int,int)

5.1 Function and For loop initialization

Save the return address $ra into the stack to make sure a correct return.




```
int i, cnt = 0;
for (i = numElements - 1; i >= 0; i--)
{
    ...
}
```

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000001  R16 (s0) = 00000015  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000001  R17 (s1) = 00000000  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 00000014  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7fffefa8  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = 00000000  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000001  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 004000d0

 FIR    = 00009800    FCSR    = 00000000    FCCR   = 00000000   FEXR    = 00000000
<
[0x004000f4]        0x20060000  addi $6, $0, 0               ; 70: addi $a2,$0,0 # a2 = 0
[0x004000f8]        0x0c100042  jal 0x00400108 [countArray]     ; 71: jal countArray # $v0 = countArray(testArray, size, 0)
[0x004000fc]        0x20080001  addi $8, $0, 1               ; 72: addi $t0, $0, 1  # wait for delay
[0x00400100]        0x00409820  add $19, $2, $0               ; 73: add $s3, $v0, $0 # ZeroCnt = $v0
[0x00400104]        0x0c100085  jal 0x00400214 [exit]          ; 74: jal exit # exit
[0x00400108]        0xafbf0054  sw $31, 84($29)              ; 77: sw $ra, 84($sp) # save ra into the stack
[0x0040010c]        0x00a04021  addu $8, $5, $0               ; 78: addu $t0,$a1,$0 # save $a1(numElements) into $t0
[0x00400110]        0x00c04821  addu $9, $6, $0               ; 79: addu $t1,$a2,$0 # save $a2(cntType) into $t1
[0x00400114]        0x210bffff  addi $11, $8, -1             ; 80: addi $t3,$t0,-1 # t3(i) = numElements - 1
[0x00400118]        0x20190000  addi $25, $0, 0               ; 81: add $t9, $0,0 # cnt = 0
[0x0040011c]        0x20080001  addi $8, $0, 1               ; 84: addi $t0, $0, 1  # wait for delay
[0x00400120]        0x0160602a  slt $12, $11, $0             ; 85: slt $t4,$t3,$0 # t4 = i < 0
[0x00400124]        0x1580001e  bne $12, $0, 120 [countArrayForEnd-0x00400124]; 86: bne $t4,$0,countArrayForEnd # if (i< 0 == 1), go to ForEnd
<
            DATA
[0x10000000]...[0x10040000]            0x00000000

            STACK
[0x7fffefa8]                           0x0000000e  0xffffff1
[0x7fffefb0]                           0x00000000  0xffffff3  0xffffff8  0x00000010
[0x7fffefc0]                           0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffefd0]                           0x00000012  0x0000000e  0xffffffe2  0xffffff3
[0x7fffefe0]                           0xffffffe  0x00000011  0x00000000  0xffffffe5
[0x7fffeff0]                           0x0000000b  0xffffffe2  0x00000019  0x004000d0
[0x7ffff000]...[0x80000000]    0x00000000
```

Line 77 to Line 81 in project1.s..

Save the return address into the stack, the 84 address.

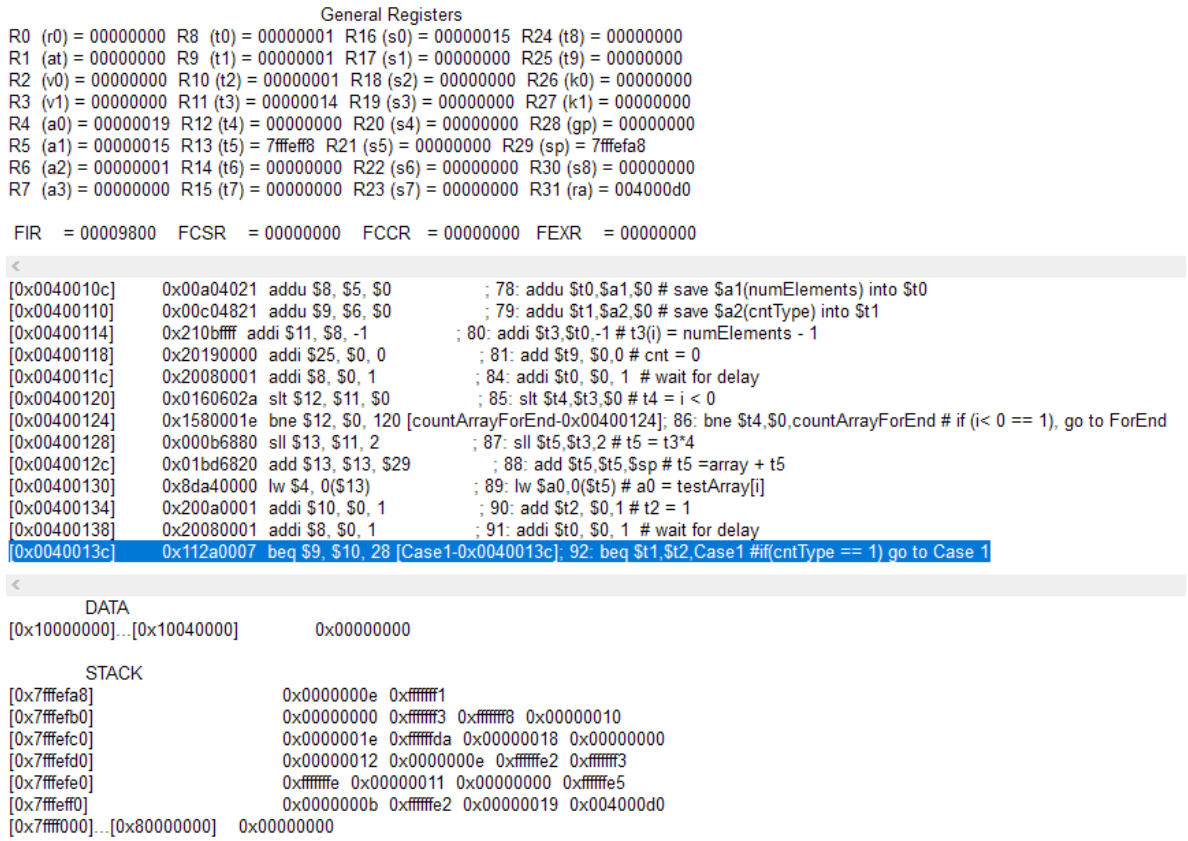
Set from arguments $numElements$ in $\$t0$ and $cntType$ in $\$t1$.

Set the iterator of the for loop in $\$t3$.

Set the result variable in register $\$t9$.

At the beginning of each loop, if i<0, it will jump to the end of loop also end of the function (see 5.5).

5.2 Operation inside loop



i = 0x15 for the above screen shot.

Visit and get value from numArray[i] by adding the $4i$ and the base address of array. The address of the value is $\$t5$. Load the value into $\$a0$ for further execution.

5.3 switch case

Regard three cases as three blocks. If the case condition is met by the current CntType, jump to the relevant block. Otherwise, it will continue to judge whether the program can jump to the next "case" block. Line 92-117 in project1.s

After executing function described in 5.4 respectively, whatever block jump to, it will return to the for loop block $countArrayFor$. Add the result of the function in 5.4 into cnt. Then continues the next loop. Line 120-124 in project1.s. In this screenshot, i = 0x25, cntType = 1.

```
switch (cntType) {
            case 1 : cnt += Pos(A[i]); break;
            case -1: cnt += Neg(A[i]); break;
            default:
                cnt += Zero(A[i]);
                break;
        }
```

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000001  R16 (s0) = 00000015  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000001  R17 (s1) = 00000000  R25 (t9) = 00000001
R2  (v0) = 00000000  R10 (t2) = 00000001  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 00000013  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 00000019  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = 00000001  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000001  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 00400164

 FIR    = 00009800    FCSR    = 00000000    FCCR    = 00000000    FEXR    = 00000000

<
[0x00400168]      0x20080001  addi $8, $0, 1              ; 105: addi $t0, $0, 1  # wait for delay
[0x0040016c]      0x0c100076  jal 0x004001d8 [Neg]          ; 108: jal Neg #v0 = Neg(array[i])
[0x00400170]      0x20080001  addi $8, $0, 1              ; 109: addi $t0, $0, 1  # wait for delay
[0x00400174]      0x08100063  j 0x0040018c [countArrayEsac]   ; 110: j countArrayEsac # break; jump to end of case
[0x00400178]      0x20080001  addi $8, $0, 1              ; 111: addi $t0, $0, 1  # wait for delay
[0x0040017c]      0x0c10007e  jal 0x004001f8 [Zero]          ; 114: jal Zero #v0 = Zero(array[i])
[0x00400180]      0x20080001  addi $8, $0, 1              ; 115: addi $t0, $0, 1  # wait for delay
[0x00400184]      0x08100063  j 0x0040018c [countArrayEsac]   ; 116: j countArrayEsac # break; jump to end of case
[0x00400188]      0x20080001  addi $8, $0, 1              ; 117: addi $t0, $0, 1  # wait for delay
[0x0040018c]      0x0322c821  addu $25, $25, $2             ; 120: addu $t9,$t9,$v0 # cnt+=v0
[0x00400190]      0x00001020  add $2, $0, $0              ; 121: add $v0,$0,$0
[0x00400194]      0x216bffff  addi $11, $11, -1           ; 122: addi $t3,$t3,-1 # i--
[0x00400198]      0x08100047  j 0x0040011c [countArrayFor]    ; 123: j countArrayFor # next loop

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

          STACK
[0x7fffefa8]                         0x0000000e  0xffffff1
[0x7fffefb0]                         0x00000000  0xffffff3  0xffffff8  0x00000010
[0x7fffefc0]                         0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffefd0]                         0x00000012  0x0000000e  0xffffffe2  0xffffff3
[0x7fffefe0]                         0xffffffe  0x00000011  0x00000000  0xffffffe5
[0x7fffeff0]                         0x0000000b  0xffffffe2  0x00000019  0x004000d0
[0x7ffff000]...[0x80000000]    0x00000000
```

5.4 Pos(int), Neg(int), Zero(int)

Refer to code block, Line 134- 162 in source code for details.

```
int Pos(int x) {
    if(x>0) return 1;
    else return 0;
}

int Neg(int x) {
    if (x<0) return 1;
    else return 0;
}

int Zero(int x) {
    if (x==0) return 1;
    else return 0;
}
```

5.5 End of for loop then end of the function

Line 126-132 in project1.s.

Set the result register $\$v0$ from result variable stored in $\$t9$.

Load the return address saved at the beginning of the function and return to main.

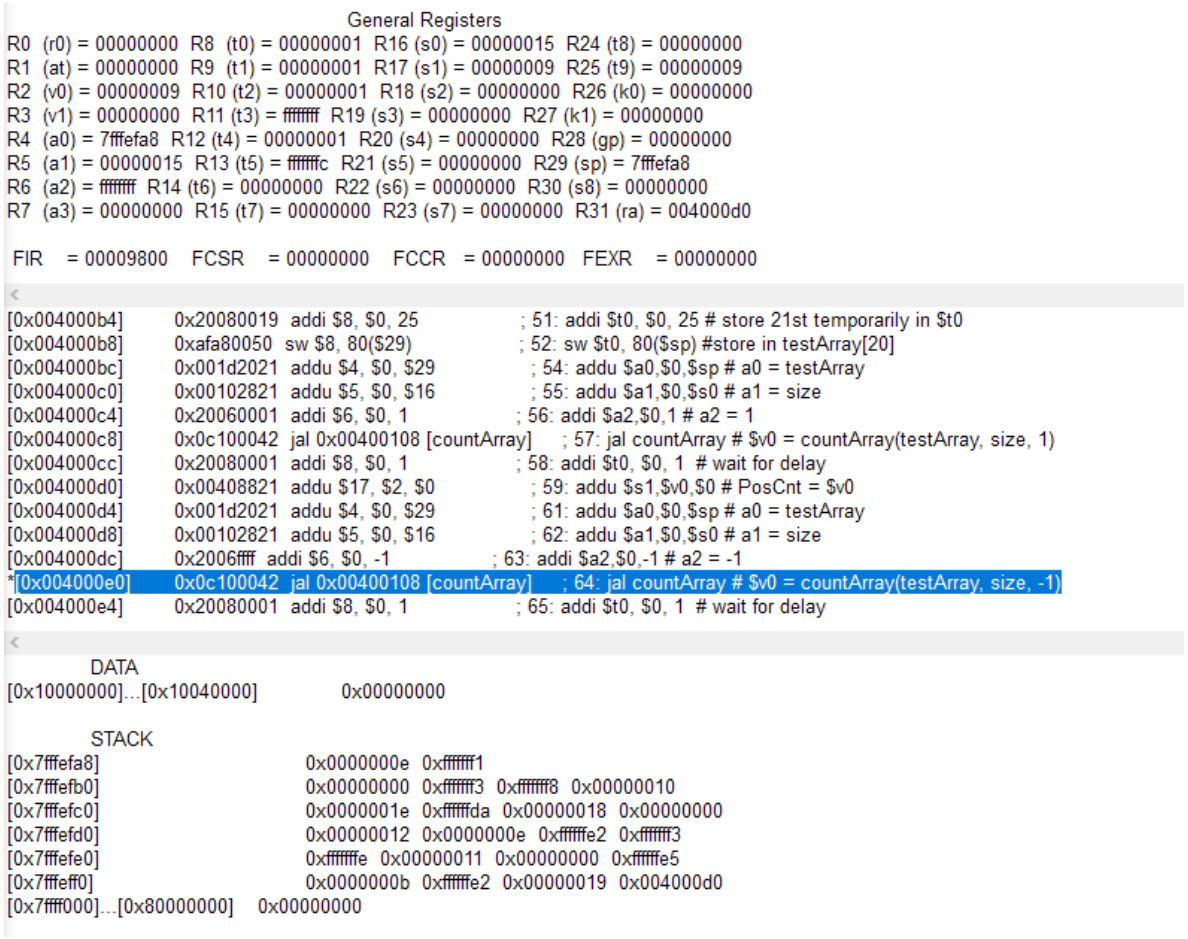
```
return cnt;
```

```
                                    General Registers
R0  (r0) = 00000000  R8  (t0) = 00000001  R16 (s0) = 00000015  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000001  R17 (s1) = 00000000  R25 (t9) = 00000009
R2  (v0) = 00000009  R10 (t2) = 00000001  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = ffffffff  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 0000000e  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = fffffffc  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000001  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 00400168

 FIR   = 00009800   FCSR   = 00000000   FCCR  = 00000000  FEXR   = 00000000

[0x0040018c]      0x20080001  addi $8, $0, 1              ; 118: addi $t0, $0, 1  # wait for delay
[0x00400190]      0x0322c821  addu $25, $25, $2           ; 121: addu $t9,$t9,$v0 # cnt+=v0
[0x00400194]      0x00001020  add $2, $0, $0              ; 122: add $v0,$0,$0 # reset v0 to 0
[0x00400198]      0x216bffff  addi $11, $11, -1           ; 123: addi $t3,$t3,-1 # i--
[0x0040019c]      0x08100048  j 0x00400120 [countArrayFor]   ; 124: j countArrayFor # next loop
[0x004001a0]      0x20080001  addi $8, $0, 1              ; 125: addi $t0, $0, 1  # wait for delay
*[0x004001a4]     0x20080001  addi $8, $0, 1              ; 128: addi $t0, $0, 1  # wait for delay
[0x004001a8]      0x00191021  addu $2, $0, $25            ; 129: addu $v0,$0,$t9 # v0 = cnt
[0x004001ac]      0x8fbf0054  lw $31, 84($29)             ; 130: lw $ra,84($sp) #recover ra from the stack
[0x004001b0]      0x20080001  addi $8, $0, 1              ; 131: addi $t0, $0, 1  # wait for delay
[0x004001b4]      0x03e00008  jr $31                      ; 132: jr $ra # return
[0x004001b8]      0x20080001  addi $8, $0, 1              ; 133: addi $t0, $0, 1  # wait for delay
[0x004001bc]      0x0004682a  slt $13, $0, $4             ; 136: slt $t5, $0, $a0      # $t5 = 0 < x

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

          STACK
[0x7fffefa8]                         0x0000000e  0xffffff1
[0x7fffefb0]                         0x00000000  0xffffff3  0xffffff8  0x00000010
[0x7fffefc0]                         0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffefd0]                         0x00000012  0x0000000e  0xfffffe2  0xffffff3
[0x7fffefe0]                         0xfffffffe  0x00000011  0x00000000  0xfffffe5
[0x7ffeff0]                          0x0000000b  0xfffffe2  0x00000019  0x004000d0
[0x7ffff000]...[0x80000000]    0x00000000
```

6. Screenshot after the first, second, third call of countArray(int*,int,int)

```
PosCnt = countArray(testArray, size, 1);
NegCnt = countArray(testArray, size, -1);
ZeroCnt = countArray(testArray, size, 0);
```

6.1 After the first call



Notice that $\$s1$=9. PosCnt = 9.It is the correct answer.

6.2 After the second call

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000001  R16 (s0) = 00000015  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = ffffffff  R17 (s1) = 00000009  R25 (t9) = 00000009
R2  (v0) = 00000009  R10 (t2) = ffffffff  R18 (s2) = 00000009  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = ffffffff  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 7fffefa8  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = fffffffc  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000000  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 004000e8

 FIR    = 00009800    FCSR    = 00000000    FCCR   = 00000000   FEXR    = 00000000

<
*[0x004000c8]      0x0c100042  jal 0x00400108 [countArray]     ; 57: jal countArray # $v0 = countArray(testArray, size, 1)
[0x004000cc]      0x20080001  addi $8, $0, 1                  ; 58: addi $t0, $0, 1  # wait for delay
[0x004000d0]      0x00408821  addu $17, $2, $0                 ; 59: addu $s1,$v0,$0 # PosCnt = $v0
[0x004000d4]      0x001d2021  addu $4, $0, $29                 ; 61: addu $a0,$0,$sp # a0 = testArray
[0x004000d8]      0x00102821  addu $5, $0, $16                 ; 62: addu $a1,$0,$s0 # a1 = size
[0x004000dc]      0x2006ffff  addi $6, $0, -1                ; 63: addi $a2,$0,-1 # a2 = -1
[0x004000e0]      0x0c100042  jal 0x00400108 [countArray]     ; 64: jal countArray # $v0 = countArray(testArray, size, -1)
[0x004000e4]      0x20080001  addi $8, $0, 1                  ; 65: addi $t0, $0, 1  # wait for delay
[0x004000e8]      0x00409020  add $18, $2, $0                  ; 66: add $s2, $v0, $0 # NegCnt = $v0
[0x004000ec]      0x001d2021  addu $4, $0, $29                 ; 68: addu $a0,$0,$sp # a0 = testArray
[0x004000f0]      0x00102821  addu $5, $0, $16                 ; 69: addu $a1,$0,$s0 # a1 = size
[0x004000f4]      0x20060000  addi $6, $0, 0                  ; 70: addi $a2,$0,0 # a2 = -1
*[0x004000f8]      0x0c100042  jal 0x00400108 [countArray]     ; 71: jal countArray # $v0 = countArray(testArray, size, -1)
<
          DATA
[0x10000000]...[0x10040000]          0x00000000

          STACK
[0x7fffefa8]                         0x0000000e  0xffffff1
[0x7fffefb0]                         0x00000000  0xfffffff3  0xfffffff8  0x00000010
[0x7fffefc0]                         0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffefd0]                         0x00000012  0x0000000e  0xffffffe2  0xfffffff3
[0x7fffefe0]                         0xfffffffe  0x00000011  0x00000000  0xffffffe5
[0x7fffeff0]                         0x0000000b  0xffffffe2  0x00000019  0x004000e8
[0x7ffff000]...[0x80000000]    0x00000000
```

Notice that $\$s2$=9. NegCnt = 9. It is the correct answer.

6.3 After the third call

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000001  R16 (s0) = 00000015  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000000  R17 (s1) = 00000009  R25 (t9) = 00000003
R2  (v0) = 00000003  R10 (t2) = ffffffff  R18 (s2) = 00000009  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = ffffffff  R19 (s3) = 00000003  R27 (k1) = 00000000
R4  (a0) = 0000000e  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = fffffffc  R21 (s5) = 00000000  R29 (sp) = 7fffefa8
R6  (a2) = 00000000  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 00400100

 FIR    = 00009800    FCSR    = 00000000    FCCR   = 00000000   FEXR    = 00000000

<
[0x004000d8]       0x00102821  addu $5, $0, $16              ; 62: addu $a1,$0,$s0 # a1 = size
[0x004000dc]       0x2006ffff  addi $6, $0, -1             ; 63: addi $a2,$0,-1 # a2 = -1
[0x004000e0]       0x0c100042  jal 0x00400108 [countArray]     ; 64: jal countArray # $v0 = countArray(testArray, size, -1)
[0x004000e4]       0x20080001  addi $8, $0, 1               ; 65: addi $t0, $0, 1  # wait for delay
[0x004000e8]       0x00409020  add $18, $2, $0              ; 66: add $s2, $v0, $0 # NegCnt = $v0
[0x004000ec]       0x001d2021  addu $4, $0, $29             ; 68: addu $a0,$0,$sp # a0 = testArray
[0x004000f0]       0x00102821  addu $5, $0, $16             ; 69: addu $a1,$0,$s0 # a1 = size
[0x004000f4]       0x20060000  addi $6, $0, 0               ; 70: addi $a2,$0,0 # a2 = 0
[0x004000f8]       0x0c100042  jal 0x00400108 [countArray]     ; 71: jal countArray # $v0 = countArray(testArray, size, 0)
[0x004000fc]       0x20080001  addi $8, $0, 1               ; 72: addi $t0, $0, 1  # wait for delay
[0x00400100]       0x00409820  add $19, $2, $0              ; 73: add $s3, $v0, $0 # ZeroCnt = $v0
*[0x00400104]      0x0c100085  jal 0x00400214 [exit]           ; 74: jal exit # exit
[0x00400108]       0xafbf0054  sw $31, 84($29)              ; 77: sw $ra, 84($sp) # save ra into the stack

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

            STACK
[0x7fffefa8]                           0x0000000e  0xffffff1
[0x7fffefb0]                           0x00000000  0xfffffff3  0xfffffff8  0x00000010
[0x7fffefc0]                           0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffefd0]                           0x00000012  0x0000000e  0xffffffe2  0xfffffff3
[0x7fffefe0]                           0xfffffffe  0x00000011  0x00000000  0xffffffe5
[0x7fffeff0]                           0x0000000b  0xffffffe2  0x00000019  0x00400100
[0x7ffff000]...[0x80000000]    0x00000000
```

Notice that $\$s3$=3. ZeroCnt = 3. It is the correct answer.

7. Exit

```
                                General Registers
R0  (r0) = 00000000  R8  (t0) = 00000001  R16 (s0) = 00000015  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000000  R17 (s1) = 00000009  R25 (t9) = 00000003
R2  (v0) = 0000000a  R10 (t2) = ffffffff  R18 (s2) = 00000009  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = ffffffff  R19 (s3) = 00000003  R27 (k1) = 00000000
R4  (a0) = 0000000e  R12 (t4) = 00000001  R20 (s4) = 00000000  R28 (gp) = 00000000
R5  (a1) = 00000015  R13 (t5) = fffffffc  R21 (s5) = 00000000  R29 (sp) = 7fffee78
R6  (a2) = 00000000  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 0040010c

 FIR    = 00009800    FCSR    = 00000000    FCCR   = 00000000   FEXR    = 00000000
```

```
[0x004001e8]    0x20080001  addi $8, $0, 1              ; 150: addi $t0, $0, 1  # wait for delay
[0x004001ec]    0x20020001  addi $2, $0, 1              ; 152: addi $v0, $0, 1        # $v0 = 1
[0x004001f0]    0x03e00008  jr $31                    ; 153: jr $ra                 # return
[0x004001f4]    0x20080001  addi $8, $0, 1              ; 154: addi $t0, $0, 1  # wait for delay
[0x004001f8]    0x10800003  beq $4, $0, 12 [Zerolf-0x004001f8]; 156: beq $a0, $0, Zerolf     # if ($a0 == 0) go to Zerolf
[0x004001fc]    0x20020000  addi $2, $0, 0              ; 157: addi $v0, $0, 0        # $v0 = 0
[0x00400200]    0x03e00008  jr $31                    ; 158: jr $ra                 # return
[0x00400204]    0x20080001  addi $8, $0, 1              ; 159: addi $t0, $0, 1  # wait for delay
[0x00400208]    0x20020001  addi $2, $0, 1              ; 161: addi $v0, $0, 1        # $v0 = 1
[0x0040020c]    0x03e00008  jr $31                    ; 162: jr $ra                 # return
[0x00400210]    0x20080001  addi $8, $0, 1              ; 163: addi $t0, $0, 1  # wait for delay
[0x00400214]    0x2002000a  addi $2, $0, 10             ; 166: addi $v0, $0, 10       # prepare to exit (system call 10)
[0x00400218]    0x0000000c  syscall                   ; 167: syscall                # exit
```

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

          STACK
[0x7fffee78]                    0x0000000e  0xffffff1
[0x7fffee80]                    0x00000000  0xffffff3  0xffffff8  0x00000010
[0x7fffee90]                    0x0000001e  0xffffffda  0x00000018  0x00000000
[0x7fffeea0]                    0x00000012  0x0000000e  0xffffffe2  0xffffff3
[0x7fffeeb0]                    0xffffffe  0x00000011  0x00000000  0xffffffe5
[0x7fffeec0]                    0x0000000b  0xffffffe2  0x00000019  0x0040010c
[0x7fffeed0]                    0x7fffefca  0x7fffefb3  0x00000000  0x7fffffe1
[0x7fffeee0]                    0x7fffffba  0x7fffff8f  0x7fffff26  0x7ffffecb
[0x7fffeef0]                    0x7ffffe93  0x7ffffe5c  0x7ffffe20  0x7ffffdef
```

8. Conclusion

In this project, we successfully practice the transformation from C language to MIPS assembly language.

Without delay operation, I countered some unexpected errors or bugs during simulation . After adding delay operations for some *jal* commands, such bugs are fixed. I wonder if there is explanation or related knowledge about this phenomenon. Since the part of code for loading 21 variables into $*sp* stack is completed by C++ program, I think there may exist better or senior usage of the assembly language to store an array.

Appendix

Array generation

```python
import random
size = 21
arr = []
for i in range(0, size):
    arr.append(random.randint(-40, 40))
print(arr)

```

project1.s

```
.text
.globl __start
__start:
    addi $s0,$0,21  #int size = 21
    addi $sp,$sp,-84 #stack for 21*4 items

    addu $s1,$0,$0 # int PosCnt = 0
    addu $s2,$0,$0 # int NegCnt = 0
    addu $s3,$0,$0 # int ZeroCnt = 0

    addi $t0, $0, 14 # store 1st temporarily in $t0
    sw $t0, 0($sp) #store in testArray[0]
    addi $t0, $0, -15 # store 2nd temporarily in $t0
    sw $t0, 4($sp) #store in testArray[1]
    addi $t0, $0, 0 # store 3th temporarily in $t0
    sw $t0, 8($sp) #store in testArray[2]
    addi $t0, $0, -13 # store 4th temporarily in $t0
    sw $t0, 12($sp) #store in testArray[3]
    addi $t0, $0, -8 # store 5th temporarily in $t0
    sw $t0, 16($sp) #store in testArray[4]
    addi $t0, $0, 16 # store 6th temporarily in $t0
    sw $t0, 20($sp) #store in testArray[5]
    addi $t0, $0, 30 # store 7th temporarily in $t0
    sw $t0, 24($sp) #store in testArray[6]
    addi $t0, $0, -38 # store 8th temporarily in $t0
    sw $t0, 28($sp) #store in testArray[7]
    addi $t0, $0, 24 # store 9th temporarily in $t0
    sw $t0, 32($sp) #store in testArray[8]
    addi $t0, $0, 0 # store 10th temporarily in $t0
    sw $t0, 36($sp) #store in testArray[9]
    addi $t0, $0, 18 # store 11st temporarily in $t0
    sw $t0, 40($sp) #store in testArray[10]
    addi $t0, $0, 14 # store 12nd temporarily in $t0
    sw $t0, 44($sp) #store in testArray[11]
    addi $t0, $0, -30 # store 13th temporarily in $t0
    sw $t0, 48($sp) #store in testArray[12]
    addi $t0, $0, -13 # store 14th temporarily in $t0
    sw $t0, 52($sp) #store in testArray[13]
    addi $t0, $0, -2 # store 15th temporarily in $t0
    sw $t0, 56($sp) #store in testArray[14]
    addi $t0, $0, 17 # store 16th temporarily in $t0
    sw $t0, 60($sp) #store in testArray[15]
    addi $t0, $0, 0 # store 17th temporarily in $t0
    sw $t0, 64($sp) #store in testArray[16]
    addi $t0, $0, -27 # store 18th temporarily in $t0
    sw $t0, 68($sp) #store in testArray[17]
    addi $t0, $0, 11 # store 19th temporarily in $t0
    sw $t0, 72($sp) #store in testArray[18]
    addi $t0, $0, -30 # store 20th temporarily in $t0
    sw $t0, 76($sp) #store in testArray[19]
    addi $t0, $0, 25 # store 21st temporarily in $t0
    sw $t0, 80($sp) #store in testArray[20]

```

```
54      addu $a0,$0,$sp # a0 = testArray
55      addu $a1,$0,$s0 # a1 = size
56      addi $a2,$0,1 # a2 = 1
57      jal countArray # $v0 = countArray(testArray, size, 1)
58      addi $t0, $0, 1  # wait for delay
59      addu $s1,$v0,$0 # PosCnt = $v0
60
61      addu $a0,$0,$sp # a0 = testArray
62      addu $a1,$0,$s0 # a1 = size
63      addi $a2,$0,-1 # a2 = -1
64      jal countArray # $v0 = countArray(testArray, size, -1)
65      addi $t0, $0, 1  # wait for delay
66      add $s2, $v0, $0 # NegCnt = $v0
67
68      addu $a0,$0,$sp # a0 = testArray
69      addu $a1,$0,$s0 # a1 = size
70      addi $a2,$0,0 # a2 = 0
71      jal countArray # $v0 = countArray(testArray, size, 0)
72      addi $t0, $0, 1  # wait for delay
73      add $s3, $v0, $0 # ZeroCnt = $v0
74      jal exit # exit
75
76  countArray:
77      sw $ra, 84($sp) # save ra into the stack
78      addu $t0,$a1,$0 # save $a1(numElements) into $t0
79      addu $t1,$a2,$0 # save $a2(cntType) into $t1
80      addi $t3,$t0,-1 # t3(i) = numElements - 1
81      add $t9, $0,0 # cnt = 0
82
83  countArrayFor:
84      addi $t0, $0, 1  # wait for delay
85      slt $t4,$t3,$0 # t4 = i < 0
86      bne $t4,$0,countArrayForEnd # if (i< 0 == 1), the loop ends, go to ForEnd
87      sll $t5,$t3,2 # t5 = t3*4
88      add $t5,$t5,$sp # t5 =array + t5
89      lw $a0,0($t5) # a0 = testArray[i]
90      add $t2, $0,1 # t2 = 1
91      addi $t0, $0, 1  # wait for delay
92      beq $t1,$t2,Case1 #if(cntType == 1) go to Case 1
93      addi $t0, $0, 1  # wait for delay
94      add $t2,$0,-1 # t2 = -1
95      beq $t1,$t2,CaseM1 #if(cntType == -1) go to Case M1(-1)
96      addi $t0, $0, 1  # wait for delay
97      beq $t1,$0, Case0 #if(cntType == 0) go to Case 0
98      j countArrayEsac # jump to end of case
99      addi $t0, $0, 1  # wait for delay
100
101 Case1:
102     jal Pos #v0 = Pos(array[i])
103     addi $t0, $0, 1  # wait for delay
104     j countArrayEsac # break; jump to end of case
105     addi $t0, $0, 1  # wait for delay
106
107 CaseM1:
108     jal Neg #v0 = Neg(array[i])
```

```
109     addi $t0, $0, 1  # wait for delay
110     j countArrayEsac # break; jump to end of case
111     addi $t0, $0, 1  # wait for delay
112
113 Case0:
114     jal Zero #v0 = Zero(array[i])
115     addi $t0, $0, 1  # wait for delay
116     j countArrayEsac # break; jump to end of case
117     addi $t0, $0, 1  # wait for delay
118
119 countArrayEsac:
120     addu $t9,$t9,$v0 # cnt+=v0
121     add $v0,$0,$0 # reset v0 to 0
122     addi $t3,$t3,-1 # i--
123     j countArrayFor # next loop
124     addi $t0, $0, 1  # wait for delay
125
126 countArrayForEnd:
127     addi $t0, $0, 1  # wait for delay
128     addu $v0,$0,$t9 # v0 = cnt
129     lw $ra,84($sp) #recover ra from the stack
130     addi $t0, $0, 1  # wait for delay
131     jr $ra # return
132     addi $t0, $0, 1  # wait for delay
133
134 Pos:
135     slt $t5, $0, $a0       # $t5 = 0 < x
136     bne $t5, $0, PosIf     # if ($t5 == 1) go to PosIf
137     addi $v0, $0, 0         # $v0 = 0
138     jr $ra                  # return
139     addi $t0, $0, 1  # wait for delay
140 PosIf:
141     addi $v0, $0, 1         # $v0 = 1
142     jr $ra                  # return
143     addi $t0, $0, 1  # wait for delay
144
145 Neg:
146     slt $t5, $a0, $0       # $t5 = x < 0
147     bne $t5, $0, NegIf     # if ($t5 == 1) go to NegIf
148     addi $v0, $0, 0         # $v0 = 0
149     jr $ra                  # return
150     addi $t0, $0, 1  # wait for delay
151 NegIf:
152     addi $v0, $0, 1         # $v0 = 1
153     jr $ra                  # return
154     addi $t0, $0, 1  # wait for delay
155 Zero:
156     beq $a0, $0, ZeroIf     # if ($a0 == 0) go to ZeroIf
157     addi $v0, $0, 0         # $v0 = 0
158     jr $ra                  # return
159     addi $t0, $0, 1  # wait for delay
160 ZeroIf:
161     addi $v0, $0, 1         # $v0 = 1
162     jr $ra                  # return
163     addi $t0, $0, 1  # wait for delay
```

```
164
165 exit:
166     addi $v0, $0, 10        # prepare to exit (system call 10)
167     syscall                 # exit
```