Ve370 Introduction to Computer Organization
Project 1 Report

Dong Jing   515370910182

OBJECTIVE

Develop a MIPS assembly program that operates on a data segment consisting of an array of 32-bit signed 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 10 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>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;
}
```

PROCEDURE

Data

```
    .data 0x10000000
array:  .word 10,54,62,77,45,61,23,34,86,60,63,59,0,1,2,13,14,15,99,98,7,8,9,80,78
    .data 0x10000100
str1:   .asciiz "The number of numbers greater or equal to 60 is "
    .data 0x10000200
str2:   .asciiz "The number of numbers less than 60 is "
    .data 0x10000300
str3:   .asciiz ".\n"
    .text
```

In this part, we define the values in array. The size of array is 25. Str1, str2 and str3 are used when we use syscall.

Main Function

```
main:
    addi $sp, $sp, -8    # adjust stack for 2 items
    sw $s1, 4($sp)       # save $s1(Fail)
    sw $s0, 0($sp)       # save $s0(Pass)

    lui $a0, 0x1000      # $a0 is the base address of array
    addi $a1, $zero, 25  # $a1 is the size of array

    addi $a2, $zero, 1   # $a2=1 (cntType=1)
    jal CountArray       # $v0=CountArray(Array, size, 1)
    add $t0, $t0, $zero  # no meaning
    add $s0, $v0, $zero  # $s0=$v0($s0=Pass(Array))

    addi $a2, $zero, -1  # $a2=-1 (cntType=-1)
    jal CountArray       # $vo=CountArray(Array, size, -1)
    add $t0, $t0, $zero  # no meaning
    add $s1, $v0, $zero  # $s1=$v0($s1=Fail(Array)

    addi $v0, $zero, 4   # print str1
    lui  $a0, 0x1000
    addi $a0, $a0, 256
    syscall

    addi $v0, $zero, 1   # Load $v0 with syscall 1=print_int
    add $a0, $s0, $zero  # $a0=$s0
    syscall              # print Pass(Array)

    addi $v0, $zero, 4   #print str3
    lui $a0, 0x1000
    addi $a0, $a0, 768
    syscall

    lui $a0, 0x1000      #print str2
    addi $a0, $a0, 512
    syscall

    addi $v0, $zero, 1
    add $a0, $s1, $zero  # $a0=$s1
    syscall              # print Fail(Array)

    lw $s0, 0($sp)       # restore $s0
    lw $s1, 4($sp)       # restore $s1
    addi $sp, $sp, 8     # restore stack pointer

    addi $v0, $zero, 10  # Exit
    syscall              # Exit
```

In main function, we first store things needed in the stack. We will use $s0 to store the value of PassCnt and $s1 to store the value of FailCnt. Then we store the address of array to $a0 and set $a1 to the size of array. 25. $a2 represents the value of cntType. First, we set $a2 equal to 1 and jump to CountArray. Let $s0 equal to $v0 that is the number of the numbers in the array that is greater or equal to 60. Then set $a2 equal to -1 and jump to CountArray again. This time $v0 is the number of the numbers in the array that is less than 60. Store this value in $s1. Then we use syscall to print str1, $s0, str3, str2 and $s1. After this step, give back the stack.

CountArray Function

```
CountArray:
    addi $sp, $sp, -20  # create stack for 5 items
    sw $ra, 16($sp)     # save return address
    sw $a3, 12($sp)     # save $a3
    sw $a0, 8($sp)      # save $a0
    sw $s1, 4($sp)      # save $s1
    sw $s0, 0($sp)      # save $s0

    add $s1, $zero, $zero   # cnt($s1)=0
    add $s0, $zero, $zero   # i($s0)=0
    add $t0, $a1, $zero # $a0=address of Array[size]

Loop:
    slt $t0, $s0, $a1   # if ($s0<$a1) $t0=1 else $t0=0
    beq $t0, $zero, Exit    # if ($t0==0) exit
    add $t0, $t0, $zero # no meaning
    lw $a3, 0($a0)      # $a3=Array[i]
    addi $s0, $s0, 1    # $s0++(i++)
    addi $a0, $a0, 4    # reach the address of next number
    slt $t0, $zero, $a2 # if (0<$a2) $t0=1 else $t0=0
    bne $t0, $zero, Pass    # if ($t0!=0) Pass
    add $t0, $zero, $zero   # no meaning
    slt $t0, $zero, $a2 # if (0<$a2) $t0=1 else $t0=0
    beq $t0, $zero, Fail    # if ($t0==0) Fail
    add $t0, $t0, $zero # no meaning
```

```
Exit:
    add $v0, $s1, $zero # $v0=$s1 (Let CountArray = cnt)
    lw $s0, 0($sp)      # restore $s0
    lw $s1, 4($sp)      # restore $s1
    lw $a0, 8($sp)      # restore $a0
    lw $a3, 12($sp)     # restore $a3
    lw $ra, 16($sp)     # restore $ra
    addi $sp, $sp, 20   # restore stack pointer
    jr $ra          # return
    add $t0, $t0, $zero # no meaning
```

In CountArray function, we first store some things in the stack. $ra is the return address. $a3 is number from the array, which will be posted to the Pass or Fail. $a0 is the address of the array. $s1 represents cnt. $s0 represents i. At the beginning, $s1 and $s0 are 0. $t0 is the size of array.

Then we go to the loop. We will make i++ and judge whether it is less than the size of array. If it is equal or greater than the size of array, it will jump to the Exit part. Otherwise, get the value of Array[i] to $a3. Then i++ and let $a0 be the address of next element. Then we judge the value of $a2. If $a2 is greater than 0, jump to Pass. If $a2 is less than 0, jump to Fail.

In Exit part, we set $v0 equal to the value of $s1 so that the value of cnt will be returned to the main function. Then give back all stacks and jump back to the main function.

Pass & Fail Function

```
Pass:
    jal Passint          # $v0=Passint(Array[i])
    add $t0, $t0, $zero  # no meaning
    add $s1, $s1, $v0    # cnt=cnt+Passint(Array[i])
    j Loop               # jump to Loop
    add $t0, $t0, $zero  # no meaning

Fail:
    jal Failint          # $v0=Failint(Array[i])
    add $t0, $t0, $zero  # no meaning
    add $s1, $s1, $v0    # cnt=cnt+Failint(Array[i])
    j Loop               # jump to Loop
    add $t0, $t0, $zero  # no meaning

Passint:
    addi $t1, $zero, 59
    slt $v0, $t1, $a3    # if (60=<Array[i]) $v0=1 else $v0=0
    jr $ra               # return
    add $t0, $t0, $zero  # no meaning

Failint:
    slti $v0, $a3, 60    # if (Array[i]<60) $v0=1 else $v0=0
    jr $ra               # return
    add $t0, $t0, $zero  # no meaning
```
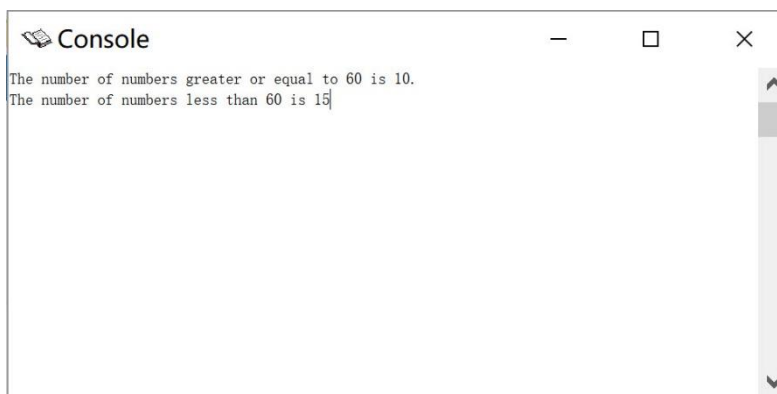
In Pass function, first jump to Passint to see whether the value of Array[i] is greater or equal to 60 or not. If so, $v0 is 1. Else $v0 is 0. Then add $v0 to $s1 and jump back to the loop. Similarly, in Fail function, we first jump to Failint. If the value of Array[i] is less than 60, $v0 will be 1. Otherwise, $v0 is 0. Then add $v0 to $s1. After it, jump back to the loop.

In Passint, first set $t1 as 59. If $t1 is less than the value of Array[i], $v0 will be 1. Otherwise $v0 is 0. Then jump back to Pass. Similar thing happens in Failint. Compare the value of Array[i] with 60. If it is less than 60, $v0 is 1 else 0.
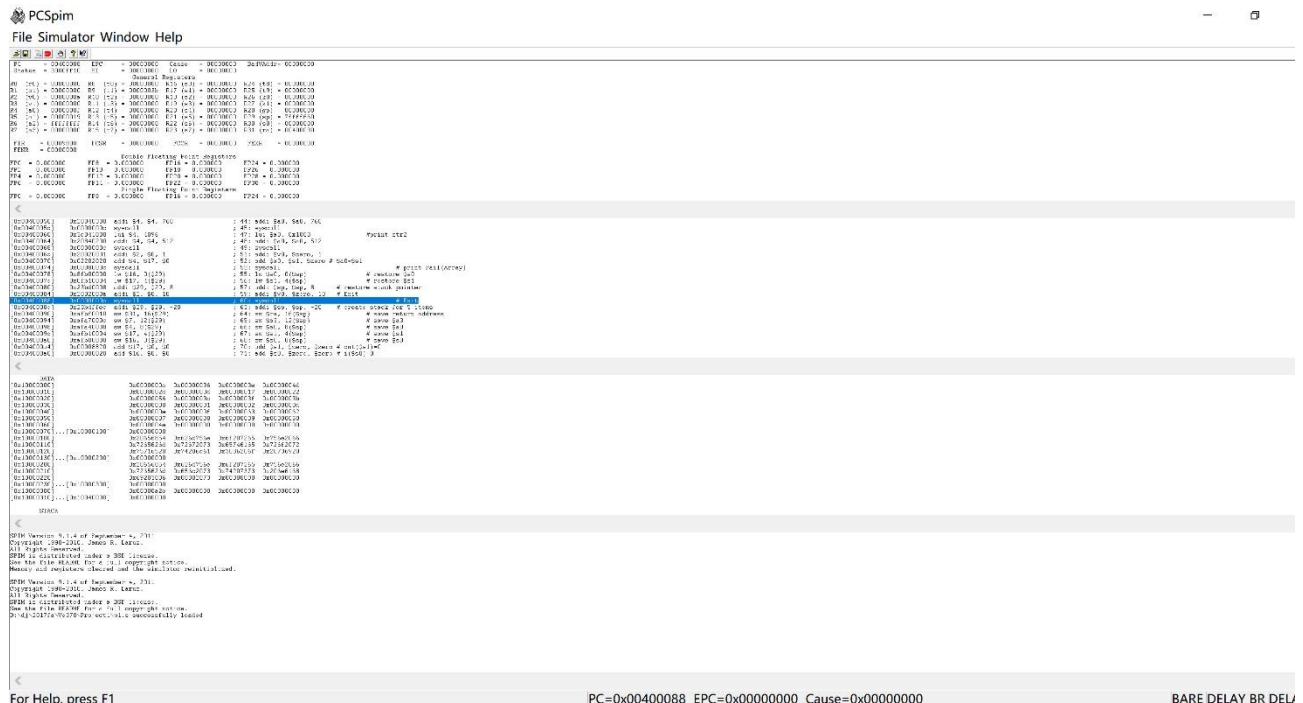
RESULT

Console:

Console

The number of numbers greater or equal to 60 is 10.
The number of numbers less than 60 is 15

Register:

As we know, the array is 10, 54, 62, 77, 45, 61, 23, 34, 86, 60, 63, 59, 0, 1, 2, 13, 14, 15, 99, 98, 7, 8, 9, 80, 78.
62, 77, 61, 86, 60, 63, 99, 98, 80 and 78 are greater or equal to 60 while 10, 54, 45, 23, 34, 59, 0, 1, 2, 13, 14, 15, 7, 8, and 9 are less than 60.

So the PassCnt is 10 and the FailCnt is 15, which is the same as the result show in console.

CONCLUSION:

In this project, we try to use Spim and program with assembly language. This is the first time we write a program in assembly language. Although the program is easy, we still meet some problems and learn a lot.

We should keep in mind that putting things into stacks is important. We are supposed to remember the different use of different registers like $a0 is for transfer parameter of functions, $v0 is for return value of functions. How to use j, jr, jal is also important because we always call the subfunctions and jump back to another function. For syscall, we need to know the different value of $v0 will lead to different result. If $v0 is equal to 1, syscall will print int and if $v0 is 4, syscall will print string. If $v0 is 10, syscall will exit the program. When we use syscall to print the result, it will show on console. Data part can help us to save data more easily. We use it to store array and some string that is used in the print part. When we finish some functions, we need to give back stacks if we use it at the beginning of the functions. Besides, we should know more about the different operands that will help a lot when programming in assembly language.

In the process of programming, we will find assembly language is like some program language. The use of slt,

beq are similar as the use of if in C++. The experience of programming in C++ will help us to learn assembly language faster and more easily. The difference between assembly language and C++ is that assembly language is more complex and more close to computer. It is hard to understand. But learning it can help us know how computer works directly. To learn it well, we need to have a better understanding of operands and develop a better ability to think logically.

NOTICE:

First, correct use of comments will make the assembly language more easily understood by others. It will also help us to debug.

Besides, assembly language is unlike other high-level language. Some small mistakes will cause a lot of troubles. When we use it, we need to be careful, especially for the address and jump part.

APPENDIX:

<u>**Source Code :**</u>

```
# Ve370 Project 1
# Dong Jing 515370910182
# p1.s
        .data 0x10000000
array:  .word 10,54,62,77,45,61,23,34,86,60,63,59,0,1,2,13,14,15,99,98,7,8,9,80,78
        .data 0x10000100
str1:   .asciiz "The number of numbers greater or equal to 60 is "
        .data 0x10000200
str2:   .asciiz "The number of numbers less than 60 is "
        .data 0x10000300
str3:   .asciiz ".\n"
        .text
        .globl __start
__start:
main:
        addi $sp, $sp, -8    # adjust stack for 2 items
        sw $s1, 4($sp)          # save $s1(Fail)
        sw $s0, 0($sp)          # save $s0(Pass)

        lui $a0, 0x1000         # $a0 is the base address of array
        addi $a1, $zero, 25     # $a1 is the size of array

        addi $a2, $zero, 1      # $a2=1 (cntType=1)
        jal CountArray          # $v0=CountArray(Array, size, 1)
        add $t0, $t0, $zero     # no meaning
        add $s0, $v0, $zero     # $s0=$v0($s0=Pass(Array))
```

```
addi $a2, $zero, -1        # $a2=-1 (cntType=-1)
jal CountArray             # $vo=CountArray(Array, size, -1)
add $t0, $t0, $zero        # no meaning
add $s1, $v0, $zero        # $s1=$v0($s1=Fail(Array))

addi $v0, $zero, 4  # print str1
lui  $a0, 0x1000
addi $a0, $a0, 256
syscall

addi $v0, $zero, 1         # Load $v0 with syscall 1=print_int
add $a0, $s0, $zero # $a0=$s0
syscall                             # print Pass(Array)

addi $v0, $zero, 4         #print str3
lui $a0, 0x1000
addi $a0, $a0, 768
syscall

lui $a0, 0x1000            #print str2
addi $a0, $a0, 512
syscall

addi $v0, $zero, 1
add $a0, $s1, $zero # $a0=$s1
syscall                             # print Fail(Array)

lw $s0, 0($sp)             # restore $s0
lw $s1, 4($sp)             # restore $s1
addi $sp, $sp, 8  # restore stack pointer

addi $v0, $zero, 10        # Exit
syscall                             # Exit


CountArray:
addi $sp, $sp, -20         # create stack for 5 items
sw $ra, 16($sp)            # save return address
```

```
        sw $a3, 12($sp)          # save $a3
        sw $a0, 8($sp)           # save $a0
        sw $s1, 4($sp)           # save $s1
        sw $s0, 0($sp)           # save $s0


        add $s1, $zero, $zero     # cnt($s1)=0
        add $s0, $zero, $zero     # i($s0)=0
        add $t0, $a1, $zero       # $a0=address of Array[size]


Loop:
        slt $t0, $s0, $a1   # if ($s0<$a1) $t0=1 else $t0=0
        beq $t0, $zero, Exit      # if ($t0==0) exit
        add $t0, $t0, $zero       # no meaning
        lw $a3, 0($a0)            # $a3=Array[i]
        addi $s0, $s0, 1   # $s0++(i++)
        addi $a0, $a0, 4   # reach the address of next number
        slt $t0, $zero, $a2       # if (0<$a2) $t0=1 else $t0=0
        bne $t0, $zero, Pass      # if ($t0!=0) Pass
        add $t0, $zero, $zero     # no meaning
        slt $t0, $zero, $a2       # if (0<$a2) $t0=1 else $t0=0
        beq $t0, $zero, Fail      # if ($t0==0) Fail
        add $t0, $t0, $zero       # no meaning


Pass:
        jal Passint                       # $v0=Passint(Array[i])
        add $t0, $t0, $zero       # no meaning
        add $s1, $s1, $v0         # cnt=cnt+Passint(Array[i])
        j Loop                            # jump to Loop
        add $t0, $t0, $zero       # no meaning


Fail:
        jal Failint                       # $v0=Failint(Array[i])
        add $t0, $t0, $zero       # no meaning
        add $s1, $s1, $v0         # cnt=cnt+Failint(Array[i])
        j Loop                            # jump to Loop
        add $t0, $t0, $zero       # no meaning


Passint:
```

```
        addi $t1, $zero, 59

        slt $v0, $t1, $a3  # if (60=<Array[i]) $v0=1 else $v0=0

        jr $ra                           # return

        add $t0, $t0, $zero       # no meaning


Failint:

        slti $v0, $a3, 60  # if (Array[i]<60) $v0=1 else $v0=0

        jr $ra                    # return

        add $t0, $t0, $zero       # no meaning


Exit:

        add $v0, $s1, $zero        # $v0=$s1 (Let CountArray = cnt)

        lw $s0, 0($sp)             # restore $s0

        lw $s1, 4($sp)             # restore $s1

        lw $a0, 8($sp)             # restore $a0

        lw $a3, 12($sp)            # restore $a3

        lw $ra, 16($sp)            # restore $ra

        addi $sp, $sp, 20# restore stack pointer

        jr $ra                    # return

        add $t0, $t0, $zero       # no meaning
```