VE370 project 1

Bingcheng HU

516021910219

PROJECT DESCRIPTION

Develop a MIPS assembly program that operates on a data segment consisting of an array of 32-bit unsigned 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 20 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>-1; 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

Overview

```
# Start
#----------------- Main -----------------

# 1. ajust stack (for `int A[]` and `string output`) and
#       generate `int numElement`, `int cntType`.

# 2. copy and past `generated_array.s` below

# ---------------- case 1 ----------------
# 3. generate ARGUIMENT for `countArray()`
#       with `cntType = 1`

# 4. $v0 = countArray(A[], size, 1)
#       count the elements greater than or equal to 60;

# 5. println(PassNum) like `Pass: 10`

# ---------------- case 2 ----------------
# 6. generate ARGUIMENT for `countArray()`
#       with `cntType = -1`

# 7. $v0 = countArray(A[], size, -1)
#       count the elements less than 60;

# 8. println(FailNum) like `Fail: 10`

# ----------- Functions (Procedures) ------------

# 1. int countArray(int A[], int numElements, int cntType)

# 2. int Pass(int x)

# 3. int Fail(int x)

# ----------------- Exit() --------------------
```

C++ Programe to generate arrays

MIPS_random_generator.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <fstream>
using namespace std;

#define MIN 30
#define MAX 90
#define SIZE 60

int main(int argc, char *argv[]) {
    ofstream oFile;
    oFile.open("generated_array.s");
    int* arr = new int[SIZE];
    srand((unsigned)time(NULL));
    cout<<"Generate "<< SIZE <<" random numbers from "
        <<MIN<<" to "<<MAX<<" :\n[";
    for(int i=0; i<SIZE; i++)
    {
        arr[i] = MIN + rand() % (MAX - MIN - 1);
        oFile<<"\taddi $t0, $0, "<<arr[i]<<"  \t# $t0 = "
            <<arr[i]<<endl;
        oFile<<"\tsw $t0, "<< 4*i <<"($s4)"<<"  \t# testArray["
            <<i<<"] = $t0"<<endl;
        cout << arr[i];
        if(i == SIZE - 1)break;
        cout << ", ";
    }
    cout<<"]"<<endl;
    delete[] arr;
    oFile.close();
    return 0;
}
```

Makefile

```
all: MIPS_random_generator.cpp
    g++ -o gen MIPS_random_generator.cpp
run: all
    ./gen
```

Output

First `cd path/to/the/file` to get into folder, then input `make run` to the terminal, and you can get the feedback below with a file named generated_array.s.

```
Generate 60 random numbers from 30 to 90 :
[61, 51, 35, 67, 74, 66, 45, 87, 55, 53, 39, 43, 65, 71, 79, 80, 81, 81,
30, 70, 69, 51, 52, 37, 40, 30, 49, 52, 39, 81, 30, 85, 50, 49, 64, 60,
75, 61, 86, 40, 52, 72, 42, 61, 30, 88, 44, 45, 30, 63, 53, 75, 71, 85,
78, 37, 36, 57, 64, 61]
```

print string and int

You can check **Table of Common Services** for more information.

```
    # print("P: ");

    # Init the string "P: \0"
    addi $t0, $0, 80    # 'P'
    sb $t0, 0($s3)
    addi $t0, $0, 58    # ':'
    sb $t0, 1($s3)
    addi $t0, $0, 32    # ' '
    sb $t0, 2($s3)
    addi $t0, $0, 0     # '\0' the end of a string
    sb $t0, 3($s3)
    addiu $a0, $s3, 0   # $a0 = $s3 ("P: \0")
    addi $v0, $0, 4     # string output (system call 4)
    syscall             # print("P: ");

    # print(a0);
    addu $a0, $0, $s5   # $a0 = $s4
    addi $v0, $0, 1     # int output (system call 1)
    syscall             # print(a0);
```

generate ARGUIMENT for `countArray()` and call function

generate ARGUIMENT

```
# 3. generate ARGUIMENT for `countArray()`
#       with `cntType = 1`

    addu $a0, $0, $s4   # $a0 = A[]
    addu $a1, $0, $s0   # $a1 = size
    addi $a2, $0, 1     # $a2 = 1
                # if with `cntType = -1`, `addi $a2, $0, -1`
```

call procedure

```
# 4. $v0 = countArray(A[], size, 1)
#       count the elements greater than or equal to 60;
    jal countArray      # $v0 = countArray(A, size, 1)
    addi $t1, $0, 0     # wait for delay
    addu $s5, $0, $v0   # save the result into $s5
```

In procedure

`$s0, $s1, $s2, $s3, $s4`are used in my program, so I need to store hem to stack before they are used in this procedure.

Moreover, `countArry()` calls other functions like `Pass()` and `Fail()`, so we need alse store `$ra` into stack.

Pass and Fail

```

# 2. int Pass(int x)
Pass:
    addi $t0, $0, 60        # $t0 = 60
    slt $t1, $a0, $t0       # $t1 = x < 60
    beq $t1, $0, PassCntPP     # if ($t1 == 1) goto PassCntPP
    addi $v0, $0, 0         # $v0 = 0
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

PassCntPP:
    addi $v0, $0, 1         # $v0 = 1
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

# 3. int Fail(int x)
Fail:
    addi $t0, $0, 60        # $t0 = 60
    slt $t1, $a0, $t0       # $t1 = x < 60
    bne $t1, $0, FailCntPP    # if ($t1 != 1) goto FailCntPP
    addi $v0, $0, 0         # $v0 = 0
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

FailCntPP:
    addi $v0, $0, 1         # $v0 = 1
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress
```

Adjust stack for used items

```
    addi $sp, $sp, -24  # adjust stack for 6 items
    sw $s0, 0($sp)      # save $s0 on stack
    sw $s1, 4($sp)      # save $s1 on stack
    sw $s2, 8($sp)      # save $s2 on stack
    sw $s3, 12($sp)     # save $s3 on stack
    sw $s4, 16($sp)     # save $s4 on stack
    sw $ra, 20($sp)     # save $ra on stack
```

Destroy stack after use

```
    lw $s0, 0($sp)          # restore $s0 from stack
    lw $s1, 4($sp)          # restore $s1 from stack
    lw $s2, 8($sp)          # restore $s2 from stack
    lw $s3, 12($sp)         # restore $s3 from stack
    lw $s4, 16($sp)         # restore $s4 from stack
    lw $ra, 20($sp)         # restore $ra from stack
    addi $sp, $sp, 24       # recover the stack
```

Exit()

```
exit:
    addi $v0, $0, 10
    syscall
```

`countArray` function

Because I used `jal countArray`, `$ra` is automatic saved. Procedure call operations. Jump and link `jal` ProcedureLabel (J-type) makes `$ra` = PC+4; Address of following instruction put in `$ra`.

Stack usage

Table below summarizes what is preserved across a procedure call.

Preserved	Not preserved
Saved register s: `$s0–$s7`	Temporar y register s: `$t0–$t9`
Stack pointer register : `$sp`	Argument register s: `$a0–$a3`
Return address register : `$ra`	Return value register s: `$v0–$v1`
Stack abo ve the stack pointer	Stack below the stack pointer

How to use SYSCALL system services

Step 1. Load the service number in register $v0.

Step 2. Load argument values, if any, in `$a0, $a1, $a2,` or `$f12` as specified.

Step 3. Issue the SYSCALL instruction.

Step 4. Retrieve return values, if any, from result registers as specified.

Example: display the value stored in `$t0` on the console

```
    li $v0, 1           # service 1 is print integer
    add $a0, $t0, $zero
    # load desired value into argument register $a0, using pseudo-op
    syscall
```

Table of Common Services

You can check full Table HERE.

Service	Code in $v0	Arguments
print integer	1	$a0 = integer to print
print float	2	$f12 = float to print
print double	3	$f12 = double to print
print string	4	$a0 = address of null-terminated string to print
exit (terminate execution)	10	

For print string, you can check `ascii` Table HERE.

Result

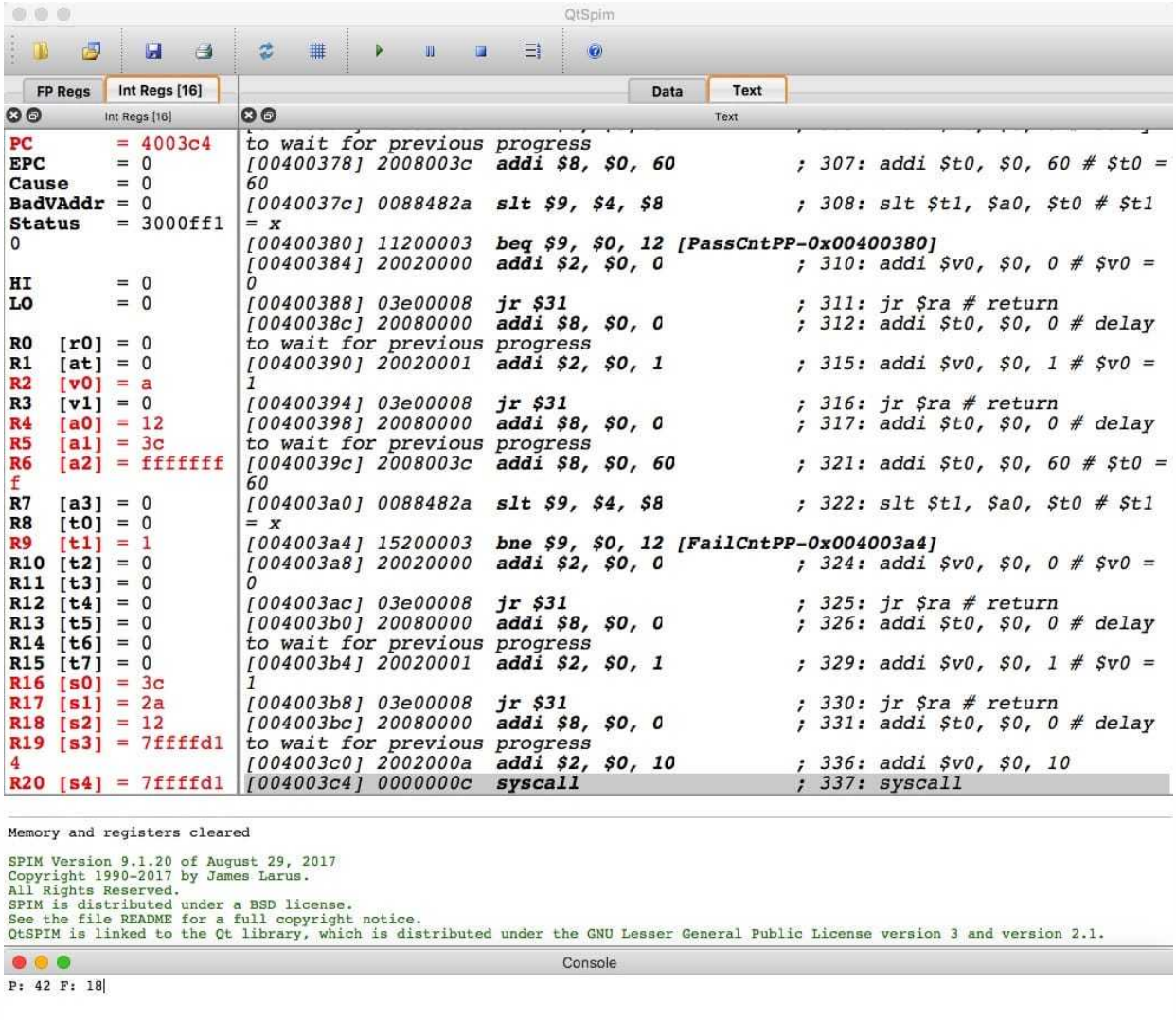
```
Generate 60 random numbers from 30 to 90 :
[61, 51, 35, 67, 74, 66, 45, 87, 55, 53, 39, 43, 65, 71, 79, 80, 81, 81,
30, 70, 69, 51, 52, 37, 40, 30, 49, 52, 39, 81, 30, 85, 50, 49, 64, 60,
75, 61, 86, 40, 52, 72, 42, 61, 30, 88, 44, 45, 30, 63, 53, 75, 71, 85,
78, 37, 36, 57, 64, 61]
```

In this array, there are 42 pass and 18 fail.

$42 + 18 = 60$

which is equal to the number of numbers, and it is the right answer.

The simulation shortcut is shown in Figure 1.



Conclution

Some problems

Delay to wait for previous progress

The delay in different procedures are very confusing. It's so hard to debug with MIPS language because sometimes only a small delay could cause big problem!

For example:

```
FailCntPP:
    addi $v0, $0, 1         # $v0 = 1
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress
```

Without `addi $t0, $0, 0` here, the answer will be `P: 42 F: 180`, which is wrong because Fail should be `18` instead of `180`.

Print strings

```
    # Init the string "F: \0"
    addi $t0, $0, 70    # 'F'
    sb $t0, 0($s3)
    addi $t0, $0, 58    # ':'
    sb $t0, 1($s3)
    addi $t0, $0, 32    # ' '
    sb $t0, 2($s3)
    addi $t0, $0, 0     # '\0' the end of a string
    sb $t0, 3($s3)
    addiu $a0, $s3, 0   # $a0 = $s3 ("F: \0")
    addi $v0, $0, 4     # string output (system call 4)
    syscall             # print("F: ");
```

You must add `\0` at the end of the string, without `addi $t0, $0, 0` and `sb $t0, 3($s3)`, the outprint string will be strange (Some times it will be normal).

Suggestions

It will be better for you to debug with adding `delay part` after every `j` and `jal`.

You can print whatever you want to debug or just watch the `Int Regs [16]` with QtSpim step by step.

Manually stepping through a program can be tedious for long-running programs. To make things easier, you can specify a stopping point called a "breakpoint" in your code.

Full Program `p1.s`

```
# p1.s
# Wrote by Bingcheng, SJTU, 2018, 10, 07
    .text
    .globl __start
__start:
# Start
#----------------- Main -----------------

# 1. ajust stack (for `int A[]` and `string output`) and
#       generate `int numElement`, `int cntType`.

    addi $sp, $sp, -204 # adjust stack for 50*4+12 items
    addi $s0, $0, 60    # int size = 60
    addu $s1, $0, $0    # int PassCnt = 0
    addu $s2, $0, $0    # int FailCnt = 0
    addu $s3, $0, $sp   # String with length 4-1 = 3
    addiu $s4, $s3, 4   # int A[size]

# 2. copy and past `generated_array.s` below

    addi $t0, $0, 48    # $t0 = 48
    sw $t0, 0($s4)      # A[0] = $t0
    addi $t0, $0, 134   # $t0 = 134
    sw $t0, 4($s4)      # A[1] = $t0
    addi $t0, $0, 128   # $t0 = 128
    sw $t0, 8($s4)      # A[2] = $t0
    addi $t0, $0, 83    # $t0 = 83
    sw $t0, 12($s4)     # A[3] = $t0
    addi $t0, $0, 65    # $t0 = 65
    sw $t0, 16($s4)     # A[4] = $t0
    addi $t0, $0, 111   # $t0 = 111
    sw $t0, 20($s4)     # A[5] = $t0
    addi $t0, $0, 92    # $t0 = 92
    sw $t0, 24($s4)     # A[6] = $t0
    addi $t0, $0, 41    # $t0 = 41
    sw $t0, 28($s4)     # A[7] = $t0
    addi $t0, $0, 113   # $t0 = 113
    sw $t0, 32($s4)     # A[8] = $t0
    addi $t0, $0, 79    # $t0 = 79
    sw $t0, 36($s4)     # A[9] = $t0
    addi $t0, $0, 60    # $t0 = 60
    sw $t0, 40($s4)     # A[10] = $t0
    addi $t0, $0, 57    # $t0 = 57
    sw $t0, 44($s4)     # A[11] = $t0
    addi $t0, $0, 66    # $t0 = 66
    sw $t0, 48($s4)     # A[12] = $t0
    addi $t0, $0, 93    # $t0 = 93
    sw $t0, 52($s4)     # A[13] = $t0
    addi $t0, $0, 136   # $t0 = 136
```

```
    sw $t0, 56($s4)     # A[14] = $t0
    addi $t0, $0, 58    # $t0 = 58
    sw $t0, 60($s4)     # A[15] = $t0
    addi $t0, $0, 57    # $t0 = 57
    sw $t0, 64($s4)     # A[16] = $t0
    addi $t0, $0, 132   # $t0 = 132
    sw $t0, 68($s4)     # A[17] = $t0
    addi $t0, $0, 134   # $t0 = 134
    sw $t0, 72($s4)     # A[18] = $t0
    addi $t0, $0, 118   # $t0 = 118
    sw $t0, 76($s4)     # A[19] = $t0
    addi $t0, $0, 129   # $t0 = 129
    sw $t0, 80($s4)     # A[20] = $t0
    addi $t0, $0, 34    # $t0 = 34
    sw $t0, 84($s4)     # A[21] = $t0
    addi $t0, $0, 55    # $t0 = 55
    sw $t0, 88($s4)     # A[22] = $t0
    addi $t0, $0, 120   # $t0 = 120
    sw $t0, 92($s4)     # A[23] = $t0
    addi $t0, $0, 117   # $t0 = 117
    sw $t0, 96($s4)     # A[24] = $t0
    addi $t0, $0, 42    # $t0 = 42
    sw $t0, 100($s4)    # A[25] = $t0
    addi $t0, $0, 110   # $t0 = 110
    sw $t0, 104($s4)    # A[26] = $t0
    addi $t0, $0, 81    # $t0 = 81
    sw $t0, 108($s4)    # A[27] = $t0
    addi $t0, $0, 132   # $t0 = 132
    sw $t0, 112($s4)    # A[28] = $t0
    addi $t0, $0, 46    # $t0 = 46
    sw $t0, 116($s4)    # A[29] = $t0
    addi $t0, $0, 102   # $t0 = 102
    sw $t0, 120($s4)    # A[30] = $t0
    addi $t0, $0, 47    # $t0 = 47
    sw $t0, 124($s4)    # A[31] = $t0
    addi $t0, $0, 117   # $t0 = 117
    sw $t0, 128($s4)    # A[32] = $t0
    addi $t0, $0, 76    # $t0 = 76
    sw $t0, 132($s4)    # A[33] = $t0
    addi $t0, $0, 56    # $t0 = 56
    sw $t0, 136($s4)    # A[34] = $t0
    addi $t0, $0, 60    # $t0 = 60
    sw $t0, 140($s4)    # A[35] = $t0
    addi $t0, $0, 106   # $t0 = 106
    sw $t0, 144($s4)    # A[36] = $t0
    addi $t0, $0, 143   # $t0 = 143
    sw $t0, 148($s4)    # A[37] = $t0
    addi $t0, $0, 50    # $t0 = 50
    sw $t0, 152($s4)    # A[38] = $t0
    addi $t0, $0, 50    # $t0 = 50
    sw $t0, 156($s4)    # A[39] = $t0
    addi $t0, $0, 145   # $t0 = 145
```

```
    sw $t0, 160($s4)    # A[40] = $t0
    addi $t0, $0, 47    # $t0 = 47
    sw $t0, 164($s4)    # A[41] = $t0
    addi $t0, $0, 115   # $t0 = 115
    sw $t0, 168($s4)    # A[42] = $t0
    addi $t0, $0, 49    # $t0 = 49
    sw $t0, 172($s4)    # A[43] = $t0
    addi $t0, $0, 90    # $t0 = 90
    sw $t0, 176($s4)    # A[44] = $t0
    addi $t0, $0, 118   # $t0 = 118
    sw $t0, 180($s4)    # A[45] = $t0
    addi $t0, $0, 70    # $t0 = 70
    sw $t0, 184($s4)    # A[46] = $t0
    addi $t0, $0, 111   # $t0 = 111
    sw $t0, 188($s4)    # A[47] = $t0
    addi $t0, $0, 64    # $t0 = 64
    sw $t0, 192($s4)    # A[48] = $t0
    addi $t0, $0, 34    # $t0 = 34
    sw $t0, 196($s4)    # A[49] = $t0
    addi $t0, $0, 95    # $t0 = 95
    sw $t0, 200($s4)    # A[50] = $t0
    addi $t0, $0, 49    # $t0 = 49
    sw $t0, 204($s4)    # A[51] = $t0
    addi $t0, $0, 112   # $t0 = 112
    sw $t0, 208($s4)    # A[52] = $t0
    addi $t0, $0, 91    # $t0 = 91
    sw $t0, 212($s4)    # A[53] = $t0
    addi $t0, $0, 98    # $t0 = 98
    sw $t0, 216($s4)    # A[54] = $t0
    addi $t0, $0, 107   # $t0 = 107
    sw $t0, 220($s4)    # A[55] = $t0
    addi $t0, $0, 106   # $t0 = 106
    sw $t0, 224($s4)    # A[56] = $t0
    addi $t0, $0, 58    # $t0 = 58
    sw $t0, 228($s4)    # A[57] = $t0
    addi $t0, $0, 140   # $t0 = 140
    sw $t0, 232($s4)    # A[58] = $t0
    addi $t0, $0, 136   # $t0 = 136
    sw $t0, 236($s4)    # A[59] = $t0

# 3. generate ARGUIMENT for `countArray()`
#       with `cntType = 1`

    addu $a0, $0, $s4   # $a0 = A[]
    addu $a1, $0, $s0   # $a1 = size
    addi $a2, $0, 1     # $a2 = 1

# 4. $v0 = countArray(A[], size, 1)
#       count the elements greater than or equal to 60;
    jal countArray
    addi $t1, $0, 1
    add $s1, $0, $v0
```

```
154
155 # 5. generate ARGUIMENT for `countArray()`
156 #       with `cntType = -1`
157     addu $a0, $0, $s4   # $a0 = A[]
158     addu $a1, $0, $s0   # $a1 = size
159     addi $a2, $0, -1    # $a2 = -1
160
161 # 6. $v0 = countArray(A[], size, -1)
162 #       count the elements less than 60;
163     jal countArray
164     addi $t1, $0, 1
165     add $s2, $0, $v0
166
167 # 7. println(PassNum) like `P: 10`
168     # Init the string "P: \0"
169     addi $t0, $0, 80    # 'P'
170     sb $t0, 0($s3)
171     addi $t0, $0, 58    # ':'
172     sb $t0, 1($s3)
173     addi $t0, $0, 32    # ' '
174     sb $t0, 2($s3)
175     addi $t0, $0, 0     # '\0' the end of a string
176     sb $t0, 3($s3)
177     addiu $a0, $s3, 0   # $a0 = $s3 ("P: \0")
178     addi $v0, $0, 4     # string output (system call 4)
179     syscall             # print("P: ");
180
181     add $a0, $0, $s1    # $a0 = $s4
182     addi $v0, $0, 1     # int output (system call 1)
183     syscall             # print(a0);
184 # print a blank
185     addi $t0, $0, 32    # ' '
186     sb $t0, 0($s3)
187     addi $t0, $0, 0     # '\0' the end of a string
188     sb $t0, 1($s3)
189     addiu $a0, $s3, 0   # $a0 = $s3 (" ")
190     addi $v0, $0, 4     # string output (system call 4)
191     syscall             # print(" ");
192
193 # 8. println(FailNum) like `F: 10`
194     # Init the string "F: \0"
195     addi $t0, $0, 70    # 'F'
196     sb $t0, 0($s3)
197     addi $t0, $0, 58    # ':'
198     sb $t0, 1($s3)
199     addi $t0, $0, 32    # ' '
200     sb $t0, 2($s3)
201     addi $t0, $0, 0     # '\0' the end of a string
202     sb $t0, 3($s3)
203     addiu $a0, $s3, 0   # $a0 = $s3 ("F: \0")
204     addi $v0, $0, 4     # string output (system call 4)
205     syscall             # print("F: ");
```

```
206
207     add $a0, $0, $s2    # $a0 = $s5
208     addi $v0, $0, 1     # int output (system call 1)
209     syscall             # print(a0);
210
211 # 9. exit
212     jal exit
213     addi $t0, $0, 0
214
215 # ----------- Functions (Procedures) ------------
216
217 # 1. int countArray(int A[], int numElements, int cntType)
218 countArray:
219     addi $sp, $sp, -24  # adjust stack for 6 items
220     sw $s0, 0($sp)
221     sw $s1, 4($sp)
222     sw $s2, 8($sp)
223     sw $s3, 12($sp)
224     sw $s4, 16($sp)
225     sw $ra, 20($sp)
226
227     addu $s0, $0, $a0
228     addu $s1, $0, $a1
229     add $s2, $0, $a2
230
231     addi $s3, $s1, -1
232     addi $s4, $0, 0
233     addi $v0, $0, 0
234
235 Loop:
236     addi $t0, $0, 0         # delay to wait for previous progress
237     slt $t0, $s3, $0        # $t0 = i < 0
238     bne $t0, $0, breakHere
239
240     sll $t0, $s3 ,2         # $t0 = i * 4
241     add $t0, $s0, $t0       # $t0 = A + $t0
242     lw $a0, 0($t0)          # $a0 = $t0 = A[]
243     addi $t1, $0, 1         # $t1 = 1
244     addi $t1, $0, 1         # delay to wait for previous progress
245     beq $s2, $t1, JalToPass # if (cntType == 1) JalToPass
246     addi $t0, $0, 0         # delay to wait for previous progress
247
248 JalToFail:
249     jal Fail                # $v0 = Fail(A[i])
250     addi $t0, $0, 0         # delay to wait for previous progress
251     add $s4, $s4, $v0       # cnt += $v0
252     addi $s3, $s3, -1       # i--
253     j Loop                  # jump to Loop
254     addi $t0, $0, 0         # delay to wait for previous progress
255
256 JalToPass:
257     jal Pass                # $v0 = Pass(A[i])
```

```
    addi $t1, $0, 0         # delay to wait for previous progress
    add $s4, $s4, $v0       # cnt += $v0
    addi $s3, $s3, -1       # i--
    j Loop                  # jump to Loop
    addi $t0, $0, 0         # delay to wait for previous progress



breakHere:
    addi $t0, $0, 0         # delay to wait for previous progress
    add $v0, $0, $s4
    lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $s2, 8($sp)
    lw $s3, 12($sp)
    lw $s4, 16($sp)
    lw $ra, 20($sp)
    addi $sp, $sp, 24

    addi $t0, $0, 0         # delay to wait for previous progress
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

# 2. int Pass(int x)
Pass:
    addi $t0, $0, 60        # $t0 = 60
    slt $t1, $a0, $t0       # $t1 = x < 60
    beq $t1, $0, PassCntPP  # if ($t1 == 1) goto PassCntPP
    addi $v0, $0, 0         # $v0 = 0
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

PassCntPP:
    addi $v0, $0, 1         # $v0 = 1
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

# 3. int Fail(int x)
Fail:
    addi $t0, $0, 60        # $t0 = 60
    slt $t1, $a0, $t0       # $t1 = x < 60
    bne $t1, $0, FailCntPP  # if ($t1 != 1) goto FailCntPP
    addi $v0, $0, 0         # $v0 = 0
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress

FailCntPP:
    addi $v0, $0, 1         # $v0 = 1
    jr $ra                  # return
    addi $t0, $0, 0         # delay to wait for previous progress



```

```
# ----------------- Exit() --------------------
exit:
    addi $v0, $0, 10
    syscall
```