

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.

```
1  main() {
2      int size = ...; //determine the size of the array here
3      int PassCnt, FailCnt;
4      int testArray[size] = { 55, 83,
5          ... //compose your own array here
6          };
7      PassCnt = countArray(testArray, size, 1);
8      FailCnt = countArray(testArray, size, -1);
9  }
10
11 int countArray(int A[], int numElements, int cntType) {
12     /******
13     * Count specific elements in the integer array A[] whose size is      *
14     * numElements and return the following:                                *
15     * When cntType = 1, count the elements greater than or equal to 60;    *
16     * When cntType = -1, count the elements less than 60;                 *
17     *****/
18     int i, cnt = 0;
19     for (i=numElements-1; i>=-1; i--) {
20         switch (cntType) {
21             case '1' : cnt += Pass(A[i]); break;
22             otherwise: cnt += Fail(A[i]);
23         }
24     }
25     return cnt;
26 }
27
28 int Pass(int x) {
29     if(x>=60) return 1;
```

```

30     else return 0;
31 }
32
33 int Fail(int x) {
34     if (x<60) return 1;
35     else return 0;
36 }

```

Procedure

Overview

```

1  # Start
2  #----- Main -----
3
4  # 1. ajust stack (for `int A[]` and `string output`) and
5  #     generate `int numElement`, `int cntType`.
6
7  # 2. copy and past `generated_array.s` below
8
9  # ----- case 1 -----
10 # 3. generate ARGUMENT for `countArray()`
11 #     with `cntType = 1`
12
13 # 4. $v0 = countArray(A[], size, 1)
14 #     count the elements greater than or equal to 60;
15
16 # 5. println(PassNum) like `Pass: 10`
17
18 # ----- case 2 -----
19 # 6. generate ARGUMENT for `countArray()`
20 #     with `cntType = -1`
21
22 # 7. $v0 = countArray(A[], size, -1)
23 #     count the elements less than 60;
24
25 # 8. println(FailNum) like `Fail: 10`
26
27 # ----- Functions (Procedures) -----
28
29 # 1. int countArray(int A[], int numElements, int cntType)
30
31 # 2. int Pass(int x)
32
33 # 3. int Fail(int x)
34
35 # ----- Exit() -----

```

C++ Programe to generate arrays

MIPS_random_generator.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  #include <fstream>
5  using namespace std;
6
7  #define MIN 30
8  #define MAX 90
9  #define SIZE 60
10
11 int main(int argc, char *argv[]) {
12     ofstream oFile;
13     oFile.open("generated_array.s");
14     int* arr = new int[SIZE];
15     srand((unsigned)time(NULL));
16     cout<<"Generate "<< SIZE <<" random numbers from "
17         <<MIN<<" to "<<MAX<<" :\\n[";
18     for(int i=0; i<SIZE; i++)
19     {
20         arr[i] = MIN + rand() % (MAX - MIN - 1);
21         oFile<<"\\taddi $t0, $0, "<<arr[i]<<" \\t# $t0 = "
22             <<arr[i]<<endl;
23         oFile<<"\\tsw $t0, "<< 4*i <<"($s4)"<<" \\t# testArray["
24             <<i<<" ] = $t0"<<endl;
25         cout << arr[i];
26         if(i == SIZE - 1)break;
27         cout << ", ";
28     }
29     cout<<"]"<<endl;
30     delete[] arr;
31     oFile.close();
32     return 0;
33 }
```

Makefile

```
1  all: MIPS_random_generator.cpp
2      g++ -o gen MIPS_random_generator.cpp
3  run: all
4      ./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](#).

```

1 Generate 60 random numbers from 30 to 90 :
2 [61, 51, 35, 67, 74, 66, 45, 87, 55, 53, 39, 43, 65, 71, 79, 80, 81, 81,
3 30, 70, 69, 51, 52, 37, 40, 30, 49, 52, 39, 81, 30, 85, 50, 49, 64, 60,
4 75, 61, 86, 40, 52, 72, 42, 61, 30, 88, 44, 45, 30, 63, 53, 75, 71, 85,
5 78, 37, 36, 57, 64, 61]

```

print string and int

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

```

1      # print("P: ");
2
3      # Init the string "P: \0"
4      addi $t0, $0, 80      # 'P'
5      sb $t0, 0($s3)
6      addi $t0, $0, 58      # ':'
7      sb $t0, 1($s3)
8      addi $t0, $0, 32      # ' '
9      sb $t0, 2($s3)
10     addi $t0, $0, 0        # '\0' the end of a string
11     sb $t0, 3($s3)
12     addiu $a0, $s3, 0      # $a0 = $s3 ("P: \0")
13     addi $v0, $0, 4        # string output (system call 4)
14     syscall                # print("P: ");
15
16     # print(a0);
17     addu $a0, $0, $s5      # $a0 = $s4
18     addi $v0, $0, 1        # int output (system call 1)
19     syscall                # print(a0);

```

generate ARGUMENT for countArray() and call function

generate ARGUMENT

```

1 # 3. generate ARGUMENT for `countArray()`
2 #       with `cntType = 1`
3
4     addu $a0, $0, $s4      # $a0 = A[]
5     addu $a1, $0, $s0      # $a1 = size
6     addi $a2, $0, 1        # $a2 = 1
7                               # if with `cntType = -1`, `addi $a2, $0, -1`

```

call procedure

```

1 # 4. $v0 = countArray(A[], size, 1)
2 #     count the elements greater than or equal to 60;
3     jal countArray      # $v0 = countArray(A, size, 1)
4     addi $t1, $0, 0     # wait for delay
5     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 them to stack before they are used in this procedure.

Moreover, countArray() calls other functions like Pass() and Fail(), so we need also store \$ra into stack.

Pass and Fail

```

1
2 # 2. int Pass(int x)
3 Pass:
4     addi $t0, $0, 60      # $t0 = 60
5     slt $t1, $a0, $t0    # $t1 = x < 60
6     beq $t1, $0, PassCntPP # if ($t1 == 1) goto PassCntPP
7     addi $v0, $0, 0      # $v0 = 0
8     jr $ra               # return
9     addi $t0, $0, 0      # delay to wait for previous progress
10
11 PassCntPP:
12     addi $v0, $0, 1      # $v0 = 1
13     jr $ra               # return
14     addi $t0, $0, 0      # delay to wait for previous progress
15
16 # 3. int Fail(int x)
17 Fail:
18     addi $t0, $0, 60      # $t0 = 60
19     slt $t1, $a0, $t0    # $t1 = x < 60
20     bne $t1, $0, FailCntPP # if ($t1 != 1) goto FailCntPP
21     addi $v0, $0, 0      # $v0 = 0
22     jr $ra               # return
23     addi $t0, $0, 0      # delay to wait for previous progress
24
25 FailCntPP:
26     addi $v0, $0, 1      # $v0 = 1
27     jr $ra               # return
28     addi $t0, $0, 0      # delay to wait for previous progress

```

Adjust stack for used items

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

Destroy stack after use

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

Exit()

```
1  exit:
2      addi $v0, $0, 10
3      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	Temporary register s: \$t0–\$t9
Stack pointer register : \$sp	Argument register s: \$a0–\$a3
Return address register : \$ra	Return value register s: \$v0–\$v1
Stack above 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

```

1      li $v0, 1           # service 1 is print integer
2      add $a0, $t0, $zero
3      # load desired value into argument register $a0, using pseudo-op
4      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

- 1 Generate 60 random numbers from 30 to 90 :
- 2 [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.

The screenshot displays the QtSPIM MIPS simulator interface. The top toolbar includes icons for file operations, execution, and debugging. The main window is divided into several panes:

- FP Regs:** Shows floating-point registers.
- Int Regs [16]:** Shows integer registers. The PC register is highlighted with a value of 4003c4. Other registers like EPC, Cause, BadVAddr, and Status are also visible.
- Data:** A pane for viewing memory data.
- Text:** A pane for viewing assembly code. The code includes instructions like `addi $8, $0, 60`, `slt $9, $4, $8`, `beq $9, $0, 12`, and `syscall`. Comments in the code indicate the purpose of certain instructions, such as "to wait for previous progress" and "to wait for previous delay".

Below the main panes, there is a section for "Memory and registers cleared" and a copyright notice for SPIM Version 9.1.20 of August 29, 2017, by James Larus. The bottom status bar shows "P: 42 F: 18", indicating 42 passes and 18 failures.

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:

```
1 FailCntPP:
2     addi $v0, $0, 1      # $v0 = 1
3     jr $ra              # return
4     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

```
1     # Init the string "F: \0"
2     addi $t0, $0, 70     # 'F'
3     sb $t0, 0($s3)
4     addi $t0, $0, 58     # ':'
5     sb $t0, 1($s3)
6     addi $t0, $0, 32     # ' '
7     sb $t0, 2($s3)
8     addi $t0, $0, 0      # '\0' the end of a string
9     sb $t0, 3($s3)
10    addiu $a0, $s3, 0     # $a0 = $s3 ("F: \0")
11    addi $v0, $0, 4       # string output (system call 4)
12    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

```
1 # p1.s
2 # Wrote by Bingcheng, SJTU, 2018, 10, 07
3     .text
4     .globl __start
5 __start:
6 # Start
7 #----- Main -----
8
9 # 1. ajust stack (for `int A[]` and `string output`) and
10 #     generate `int numElement`, `int cntType`.
11
12     addi $sp, $sp, -204 # adjust stack for 50*4+12 items
13     addi $s0, $0, 60   # int size = 60
14     addu $s1, $0, $0   # int PassCnt = 0
15     addu $s2, $0, $0   # int FailCnt = 0
16     addu $s3, $0, $sp  # String with length 4-1 = 3
17     addiu $s4, $s3, 4  # int A[size]
18
19 # 2. copy and past `generated_array.s` below
20
21     addi $t0, $0, 48   # $t0 = 48
22     sw $t0, 0($s4)     # A[0] = $t0
23     addi $t0, $0, 134  # $t0 = 134
24     sw $t0, 4($s4)     # A[1] = $t0
25     addi $t0, $0, 128  # $t0 = 128
26     sw $t0, 8($s4)     # A[2] = $t0
27     addi $t0, $0, 83   # $t0 = 83
28     sw $t0, 12($s4)    # A[3] = $t0
29     addi $t0, $0, 65   # $t0 = 65
30     sw $t0, 16($s4)    # A[4] = $t0
31     addi $t0, $0, 111  # $t0 = 111
32     sw $t0, 20($s4)    # A[5] = $t0
33     addi $t0, $0, 92   # $t0 = 92
34     sw $t0, 24($s4)    # A[6] = $t0
35     addi $t0, $0, 41   # $t0 = 41
36     sw $t0, 28($s4)    # A[7] = $t0
37     addi $t0, $0, 113  # $t0 = 113
38     sw $t0, 32($s4)    # A[8] = $t0
39     addi $t0, $0, 79   # $t0 = 79
40     sw $t0, 36($s4)    # A[9] = $t0
41     addi $t0, $0, 60   # $t0 = 60
42     sw $t0, 40($s4)    # A[10] = $t0
43     addi $t0, $0, 57   # $t0 = 57
44     sw $t0, 44($s4)    # A[11] = $t0
45     addi $t0, $0, 66   # $t0 = 66
46     sw $t0, 48($s4)    # A[12] = $t0
47     addi $t0, $0, 93   # $t0 = 93
48     sw $t0, 52($s4)    # A[13] = $t0
49     addi $t0, $0, 136  # $t0 = 136
```

50	sw \$t0, 56(\$s4)	# A[14] = \$t0
51	addi \$t0, \$0, 58	# \$t0 = 58
52	sw \$t0, 60(\$s4)	# A[15] = \$t0
53	addi \$t0, \$0, 57	# \$t0 = 57
54	sw \$t0, 64(\$s4)	# A[16] = \$t0
55	addi \$t0, \$0, 132	# \$t0 = 132
56	sw \$t0, 68(\$s4)	# A[17] = \$t0
57	addi \$t0, \$0, 134	# \$t0 = 134
58	sw \$t0, 72(\$s4)	# A[18] = \$t0
59	addi \$t0, \$0, 118	# \$t0 = 118
60	sw \$t0, 76(\$s4)	# A[19] = \$t0
61	addi \$t0, \$0, 129	# \$t0 = 129
62	sw \$t0, 80(\$s4)	# A[20] = \$t0
63	addi \$t0, \$0, 34	# \$t0 = 34
64	sw \$t0, 84(\$s4)	# A[21] = \$t0
65	addi \$t0, \$0, 55	# \$t0 = 55
66	sw \$t0, 88(\$s4)	# A[22] = \$t0
67	addi \$t0, \$0, 120	# \$t0 = 120
68	sw \$t0, 92(\$s4)	# A[23] = \$t0
69	addi \$t0, \$0, 117	# \$t0 = 117
70	sw \$t0, 96(\$s4)	# A[24] = \$t0
71	addi \$t0, \$0, 42	# \$t0 = 42
72	sw \$t0, 100(\$s4)	# A[25] = \$t0
73	addi \$t0, \$0, 110	# \$t0 = 110
74	sw \$t0, 104(\$s4)	# A[26] = \$t0
75	addi \$t0, \$0, 81	# \$t0 = 81
76	sw \$t0, 108(\$s4)	# A[27] = \$t0
77	addi \$t0, \$0, 132	# \$t0 = 132
78	sw \$t0, 112(\$s4)	# A[28] = \$t0
79	addi \$t0, \$0, 46	# \$t0 = 46
80	sw \$t0, 116(\$s4)	# A[29] = \$t0
81	addi \$t0, \$0, 102	# \$t0 = 102
82	sw \$t0, 120(\$s4)	# A[30] = \$t0
83	addi \$t0, \$0, 47	# \$t0 = 47
84	sw \$t0, 124(\$s4)	# A[31] = \$t0
85	addi \$t0, \$0, 117	# \$t0 = 117
86	sw \$t0, 128(\$s4)	# A[32] = \$t0
87	addi \$t0, \$0, 76	# \$t0 = 76
88	sw \$t0, 132(\$s4)	# A[33] = \$t0
89	addi \$t0, \$0, 56	# \$t0 = 56
90	sw \$t0, 136(\$s4)	# A[34] = \$t0
91	addi \$t0, \$0, 60	# \$t0 = 60
92	sw \$t0, 140(\$s4)	# A[35] = \$t0
93	addi \$t0, \$0, 106	# \$t0 = 106
94	sw \$t0, 144(\$s4)	# A[36] = \$t0
95	addi \$t0, \$0, 143	# \$t0 = 143
96	sw \$t0, 148(\$s4)	# A[37] = \$t0
97	addi \$t0, \$0, 50	# \$t0 = 50
98	sw \$t0, 152(\$s4)	# A[38] = \$t0
99	addi \$t0, \$0, 50	# \$t0 = 50
100	sw \$t0, 156(\$s4)	# A[39] = \$t0
101	addi \$t0, \$0, 145	# \$t0 = 145

```

102     sw $t0, 160($s4)    # A[40] = $t0
103     addi $t0, $0, 47    # $t0 = 47
104     sw $t0, 164($s4)    # A[41] = $t0
105     addi $t0, $0, 115   # $t0 = 115
106     sw $t0, 168($s4)    # A[42] = $t0
107     addi $t0, $0, 49    # $t0 = 49
108     sw $t0, 172($s4)    # A[43] = $t0
109     addi $t0, $0, 90    # $t0 = 90
110     sw $t0, 176($s4)    # A[44] = $t0
111     addi $t0, $0, 118   # $t0 = 118
112     sw $t0, 180($s4)    # A[45] = $t0
113     addi $t0, $0, 70    # $t0 = 70
114     sw $t0, 184($s4)    # A[46] = $t0
115     addi $t0, $0, 111   # $t0 = 111
116     sw $t0, 188($s4)    # A[47] = $t0
117     addi $t0, $0, 64    # $t0 = 64
118     sw $t0, 192($s4)    # A[48] = $t0
119     addi $t0, $0, 34    # $t0 = 34
120     sw $t0, 196($s4)    # A[49] = $t0
121     addi $t0, $0, 95    # $t0 = 95
122     sw $t0, 200($s4)    # A[50] = $t0
123     addi $t0, $0, 49    # $t0 = 49
124     sw $t0, 204($s4)    # A[51] = $t0
125     addi $t0, $0, 112   # $t0 = 112
126     sw $t0, 208($s4)    # A[52] = $t0
127     addi $t0, $0, 91    # $t0 = 91
128     sw $t0, 212($s4)    # A[53] = $t0
129     addi $t0, $0, 98    # $t0 = 98
130     sw $t0, 216($s4)    # A[54] = $t0
131     addi $t0, $0, 107   # $t0 = 107
132     sw $t0, 220($s4)    # A[55] = $t0
133     addi $t0, $0, 106   # $t0 = 106
134     sw $t0, 224($s4)    # A[56] = $t0
135     addi $t0, $0, 58    # $t0 = 58
136     sw $t0, 228($s4)    # A[57] = $t0
137     addi $t0, $0, 140   # $t0 = 140
138     sw $t0, 232($s4)    # A[58] = $t0
139     addi $t0, $0, 136   # $t0 = 136
140     sw $t0, 236($s4)    # A[59] = $t0
141
142     # 3. generate ARGUMENT for `countArray()`
143     #         with `cntType = 1`
144
145     addu $a0, $0, $s4    # $a0 = A[]
146     addu $a1, $0, $s0    # $a1 = size
147     addi $a2, $0, 1      # $a2 = 1
148
149     # 4. $v0 = countArray(A[], size, 1)
150     #         count the elements greater than or equal to 60;
151     jal countArray
152     addi $t1, $0, 1
153     add $s1, $0, $v0

```

```

154
155 # 5. generate ARGUMENT 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[i]
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])

```

```

258     addi $t1, $0, 0           # delay to wait for previous progress
259     add $s4, $s4, $v0        # cnt += $v0
260     addi $s3, $s3, -1        # i--
261     j Loop                   # jump to Loop
262     addi $t0, $0, 0           # delay to wait for previous progress
263
264
265
266 breakHere:
267     addi $t0, $0, 0           # delay to wait for previous progress
268     add $v0, $0, $s4
269     lw $s0, 0($sp)
270     lw $s1, 4($sp)
271     lw $s2, 8($sp)
272     lw $s3, 12($sp)
273     lw $s4, 16($sp)
274     lw $ra, 20($sp)
275     addi $sp, $sp, 24
276
277     addi $t0, $0, 0           # delay to wait for previous progress
278     jr $ra                   # return
279     addi $t0, $0, 0           # delay to wait for previous progress
280
281 # 2. int Pass(int x)
282 Pass:
283     addi $t0, $0, 60          # $t0 = 60
284     slt $t1, $a0, $t0         # $t1 = x < 60
285     beq $t1, $0, PassCntPP    # if ($t1 == 1) goto PassCntPP
286     addi $v0, $0, 0          # $v0 = 0
287     jr $ra                   # return
288     addi $t0, $0, 0           # delay to wait for previous progress
289
290 PassCntPP:
291     addi $v0, $0, 1           # $v0 = 1
292     jr $ra                   # return
293     addi $t0, $0, 0           # delay to wait for previous progress
294
295 # 3. int Fail(int x)
296 Fail:
297     addi $t0, $0, 60          # $t0 = 60
298     slt $t1, $a0, $t0         # $t1 = x < 60
299     bne $t1, $0, FailCntPP    # if ($t1 != 1) goto FailCntPP
300     addi $v0, $0, 0          # $v0 = 0
301     jr $ra                   # return
302     addi $t0, $0, 0           # delay to wait for previous progress
303
304 FailCntPP:
305     addi $v0, $0, 1           # $v0 = 1
306     jr $ra                   # return
307     addi $t0, $0, 0           # delay to wait for previous progress
308
309

```

```
310 # ----- Exit() -----  
311 exit:  
312     addi $v0, $0, 10  
313     syscall
```