VE370 Intro to Computer Organization

Project 1
MIPS Assembly

Written by

Hanyun Zhao 518370910091

October 5, 2020

University of Michigan - Shanghai Jiao Tong University Joint Institute
(UM-SJTU JI)
800 Dongchuan Road, Shanghai, China 200240

1 Objectives

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 as well as procedures for 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 30 elements.

2 Procedure

The whole program in C++ is composed of main function, and several subfunctions. What we need to do is to translate the given C++ code to MIPS assembly code, which practice basic skills like loop, conditional operation, and function calls.

2.1 Main function

In the main function, we define our own array and its size. I use CASIO calculator to generate one, with all elements uniformly distributed from -20 to 55. This makes the number of cold, hot, and comfortable days approximately the same. The size is 36.

```
int main() {
    int size = 36; //determine the size of the array here
    int hotDay, coldDay, comfortDay;
    int tempArray[36] = {23,41,-10,-4,4,16,29,-5,-6,
     -11,38,8,16,42,31,39,39,29,11, -17,19,53,4,21,
     32,42,37,-15, 38,-5,32,-1,41,14,13,18};
     //compose your own array here
    hotDay = countArray (tempArray, size, 1);
    coldDay = countArray (tempArray, size, -1);
    comfortDay = countArray (tempArray, size, 0);
    cout<<hotDay<<" "<<coldDay<<" "<<comfortDay<<endl;
}
```

2.2 countArray

This function is called by the main function three times in order to count the cold days, hot days, and comfortable days respectively. It uses the array's base address, the array size, and count type as arguments.

```
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 30; *
* When cntType = -1, count the elements less than or
* equal to 5; *
* When cntType = 0, count the elements greater than 5
* and less than 30. *
*************************************************************/
int i, cnt = 0;
for(i=numElements-1, i>=0, i--) {
switch (cntType) {
case '1' : cnt += hot(A[i]); break;
case '-1': cnt += cold(A[i]); break;
otherwise: cnt += comfort(A[i]);
}
}
return cnt;
}
```

3 Supporting function

Three supporting function hot(int), cold(int), and comfort(int), used for judging whether the temperature lies in the range of each category.

```
int hot(int x) {
if(x>=30) return 1;
else return 0;
}
int cold(int x) {
if (x<=5) return 1;
else return 0;
}
int comfort(int x) {
if (x>5 && x<30) return 1;
else return 0;
}
```

4 Result

I use 23,41,-10,-4,4,16,29,-5,-6,-11,38,8,16,42,31,39,39,29,11,-17,19,53,4,21,32,42,37,-15,38,-5,32,-1,41,14,13,18 as the sample array. There're totally 13 hot days, 11 cold days, and 12 comfortable days. The output is correct, as Fig 1 shows.

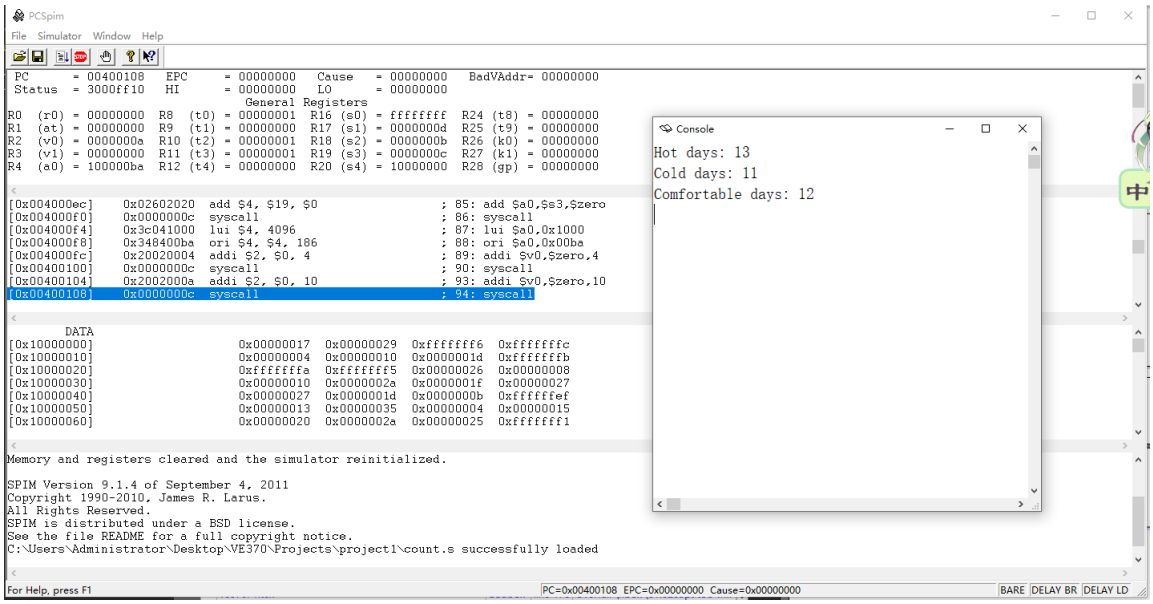

Figure 1: Simulation result

5 Discussion

5.1 Input

In order to avoid pushing the array elements into the stack one after another by hand, like the following

```
1           sw    $a0, 24($sp)
2           addi   $a0, $zero,   7
```

I use .word to save the data in the memory.

```
1 tempArray: .word 23,41,-10,-4,4,16,29,-5,-6,-11,38,8,16,42,31,39,
2 39,29,11,-17,19,53,4,21,32,42,37,-15,38,-5,32,-1,41,14,13,18
```

By checking the Data block, I found the base address is 0x10000000, which is also shown in Fig 1 above.

6 Output

To clearly convey what the three outputs indicate, I add a string before each output number, and separate them into three lines. What I need to display is first a

string, with terminating symbol, then the number, third the breaking line symbol.

To output a string with terminating symbol, I use .asciiz, which means a string ended by an empty character. This could avoid displaying all the characters stored right after.

```
$output_hot: .asciiz "Hot days: " #90-9a
$output_cold: .asciiz "Cold days: " #9b-a6
$output_comfort: .asciiz "Comfortable days: " #a7-ba
```

And by commenting all of them and them uncommented one by one, I found their addresses, which is shown in Fig2, from 0x10000090 to 0x100000b9

```
[0x10000050]                    0x00000013  0x00000035  0x00000004  0x00000015
[0x10000060]                    0x00000020  0x0000002a  0x00000025  0xfffffff1
[0x10000070]                    0x00000026  0xfffffffb  0x00000020  0xffffffff
[0x10000080]                    0x00000029  0x0000000e  0x0000000d  0x00000012
[0x10000090]                    0x20746f48  0x73796164  0x4300203a  0x20646c6f
[0x100000a0]                    0x73796164  0x4300203a  0x6f666d6f  0x62617472
[0x100000b0]                    0x6420656c  0x3a737961  0x000a0020  0x00000000
[0x100000c0]...[0x10040000]     0x00000000
```

Figure 2: string addresses

To output the number, its just simply syscall.

To output the breaking line symbol, I stored "\n" character, and found its address to be 0x100000ba. I add one after each number is displayed.

A sample output procedure code is shown below

```
    lui $a0,0x1000
    ori $a0,0x0090
    addi $v0,$zero,4
    syscall
    addi $v0,$zero,1
    add $a0,$s1,$zero
    syscall
    lui $a0,0x1000
    ori $a0,0x00ba
    addi $v0,$zero,4
    syscall #display "Hot days: xx \n"
```

6.1 Delay

A damn clever guy posted that the operating sequence is not explicit after jal,j and beq instructions. Therefore we had to add delay right after them in case the program execute some instructions in advance undesirably. The most direct way is nop instruction, but its forbidden in this project.

6.2 Others

After finishing the project, I get the answer to the question 'where should we push and destroy the stack'. For me, the most wise way is to finish them in the caller, right before and after calling the subfunctions. The reason for not doing these within the subfunctions are: First, we don't know what we should preserve and restore for the caller when writing subfunctions. Second, if we have many multiple subfuncitons to call, pushing and destroying stack within caller function may reduce redundant stack operations, thus the speed is improved.

In this project, many functions are actually limited. For example, the pesudo instruction are not allowed. This caused me some trouble when dealing with strings and arrays. If there's no limits, I can simply write

```
output_hot: .asciiz "Hot days: "
la $a0, output_hot
addi $v0,$zero,4
syscall
```

instead of looking for the physical address in the PCspim simulator.

Also, I tried using macro, but it seems not available neither.

7 Appendix

```
.data
tempArray: .word 23,41,-10,-4,4,16,29,-5,-6,-11,38,8,16,42,31,39,
39,29,11,-17,19,53,4,21,32,42,37,-15,38,-5,32,-1,41,14,13,18
$output_hot: .asciiz "Hot days: " #90-9a
$output_cold: .asciiz "Cold days: " #9b-a6
$output_comfort: .asciiz "Comfortable days: " #a7-b9
$newline: .asciiz"\n"


.text
.globl __start

#.macro print_int($int)
#addi $v0,$zero,1
#add $a0,$int,$zero
#syscall
#.end_macro

__start:
#size=s0,hotDay=s1,coldDay=s2,comfortDay=s3,tempArray=s4
    lui $s4,0x1000
    ori $s4,0x0000
    addi $s0,$zero,36
```

```
24      addi $sp,$sp,-40
25      sw $s0,36($sp)
26      sw $s1,32($sp)
27      sw $s2,28($sp)
28      sw $s3,24($sp)
29      sw $s4,20($sp)
30      add $a0,$s4,$zero
31      add $a1,$s0,$zero
32      addi $a2,$zero,1
33      jal countArray
34      add $t7,$t7,$zero
35
36      add $s1,$v0,$zero
37      sw $s1,32($sp)
38      lw $a0,20($sp)
39      lw $a1,36($sp)
40      addi $a2,$zero,-1
41      jal countArray
42      add $t7,$t7,$zero
43
44      add $s2,$v0,$zero
45      sw $s2,28($sp)
46      lw $a0,20($sp)
47      lw $a1,36($sp)
48      add $a2,$zero,$zero
49      jal countArray
50      add $t7,$t7,$zero
51
52      add $s3,$v0,$zero
53      lw $s1,32($sp)
54      lw $s2,28($sp)
55      addi $sp,$sp,40
56
57      lui $a0,0x1000
58      ori $a0,0x0090
59      addi $v0,$zero,4
60      syscall
61      addi $v0,$zero,1
62      add $a0,$s1,$zero
63      syscall
64      lui $a0,0x1000
65      ori $a0,0x00ba
66      addi $v0,$zero,4
67      syscall #display "Hot days: xx \n"
68
69      lui $a0,0x1000
70      ori $a0,0x009b
71      addi $v0,$zero,4
72      syscall
73      addi $v0,$zero,1
```

```
74      add $a0,$s2,$zero
75      syscall
76      lui $a0,0x1000
77      ori $a0,0x00ba
78      addi $v0,$zero,4
79      syscall
80
81      lui $a0,0x1000
82      ori $a0,0x00a7
83      addi $v0,$zero,4
84      syscall
85      addi $v0,$zero,1
86      add $a0,$s3,$zero
87      syscall
88      lui $a0,0x1000
89      ori $a0,0x00ba
90      addi $v0,$zero,4
91      syscall
92  #print
93
94      addi $v0,$zero,10
95      syscall
96
97  #countArray**************************************
98  countArray:
99      addi $s0,$a1,-1  # i=s0, i initilaized to be numElements-1
100     add $s1,$zero,$zero  #s1=cnt=0
101     j For
102     add $t7,$t7,$zero
103
104 For:
105     slt $t0,$s0,$zero #s0<0?
106     addi $t2,$zero,1
107     beq $t0,$t2,ExitFor
108     add $t7,$t7,$zero
109
110     #stack
111     addi $sp,$sp,-32
112     sw $a0,28($sp)
113     sw $a1,24($sp)
114     sw $a2,20($sp)
115     sw $s0,16($sp)
116     sw $s1,12($sp)
117     sw $ra,8($sp)
118     add $s2,$a0,$zero  #s2=A[]
119     sll $t0,$s0,2
120     add $s2,$s2,$t0
121
122     addi $t2,$zero,1
123     beq $a2,$t2,Hotplus
```

```
124     addi $t2,$zero,-1
125     beq $a2,$t2,Coldplus
126     j Comfortplus
127     add $t7,$t7,$zero
128
129 Hotplus:
130     lw $a0,0($s2)
131     jal Hot
132     add $t7,$t7,$zero
133     j Fortail
134     add $t7,$t7,$zero
135
136 Coldplus:
137     lw $a0,0($s2)
138     jal Cold
139     add $t7,$t7,$zero
140     j Fortail
141     add $t7,$t7,$zero
142
143 Comfortplus:
144     lw $a0,0($s2)
145     jal Comfort
146     add $t7,$t7,$zero
147     j Fortail
148     add $t7,$t7,$zero
149
150 Fortail:
151     lw $a0,28($sp)
152     lw $a1,24($sp)
153     lw $a2,20($sp)
154     lw $s0,16($sp)
155     lw $s1,12($sp)
156     lw $ra,8($sp)
157     add $sp,$sp,32
158     add$s1,$s1,$v0
159     addi $s0,$s0,-1
160     j For
161     add $t7,$t7,$zero
162
163 ExitFor:
164     add $v0,$s1,$zero
165     jr $ra
166     add $t7,$t7,$zero
167     #countArray over
168
169 #*********************************************
170
171 Hot:
172     addi $t2,$zero,30
173     slt $t0,$t2,$a0
```

```
174     addi $t3,$zero,1
175     beq $t0,$t3,hotreturn1
176     add $v0,$zero,$zero
177     jr $ra
178     add $t7,$t7,$zero
179     hotreturn1: # greater than 30,hot
180         addi $v0,$zero,1
181         jr $ra
182         add $t7,$t7,$zero
183
184 Cold:
185     addi $t2,$zero,5
186     slt $t0,$t2,$a0
187     addi $t3,$zero,1
188     beq $t0,$t3,coldreturn0
189     addi $v0,$zero,1
190     jr $ra
191     add $t7,$t7,$zero
192     coldreturn0: #greater than 5, not cold
193         add $v0,$zero,$zero
194         jr $ra
195         add $t7,$t7,$zero
196
197 Comfort:
198     addi $t2,$zero,30
199     slt $t0,$t2,$a0
200     addi $t3,$zero,1
201     beq $t0,$t3,comfortreturn0
202     addi $t2,$zero,5
203     slt $t0,$a0,$t2
204     beq $t0,$t3,comfortreturn0
205     addi $v0,$zero,1
206     jr $ra
207     add $t7,$t7,$zero
208     comfortreturn0:
209         add $v0,$zero,$zero
210         jr $ra
211         add $t7,$t7,$zero
```