
UM-SJTU JOINT INSTITUTE

Intro to Computer Organization
(VE370)

Project 1

prof. Zheng

XINMIAO YU
518021910792

DATE: 28 Sept. 2020

1 Introduction

In this project, an array of 32-bits signed integer has a predefined **size 32**. These 32 numbers are assigned randomly. The C program is shown below.

```
1  int main() {
2      int size = 32; //determine the size of the array here
3      int hotDay, coldDay, comfortDay;
4      int tempArray[32] = {36, 9, -8, 40, 25, 20, 18, 19, 15, 16, 17, 16, 15,
5          ↪ 14,
6          13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -3, 30, -19, 33};
7      hotDay = countArray (tempArray, size, 1);
8      coldDay = countArray (tempArray, size, -1);
9      comfortDay = countArray (tempArray, size, 0);
10 }
11 int countArray(int A[], int numElements, int cntType) {
12     int i, cnt = 0;
13     for (i = numElements - 1; i >= 0; i--) {
14         switch (cntType) {
15             case 1: cnt += hot(A[i]); break;
16             case -1: cnt += cold(A[i]); break;
17             default: cnt += comfort(A[i]);
18         }
19     }
20     return cnt;
21 }
22 int hot(int x) {
23     if(x>=30) return 1;
24     else return 0;
25 }
26 int cold(int x) {
27     if (x<=5) return 1;
28     else return 0;
29 }
30 int comfort(int x) {
31     if (x>5 && x<30) return 1;
32     else return 0;
33 }
```

A MIPS program with same functions as the C program is developed. In addition, no pseudo-instruction used.

2 Procedures

The program is developed following the logic of C program.

- As use `la` to load in the address of array is not allowed, the address used to store the numbers is predefined, as `0x10001000`. Then `main` function, where `int tempArray[32]=...` is needed, the base address (`0x10001000`) is load in `$a0` through the command `lui` and `ori`.

- In main function, other variables needed is defined as

```

1  $a1 = size           // as numElements = 32
2  $s0 = hotDay         // as 0
3  $s1 = coldDay        // as 0
4  $s2 = comfortDay     // as 0

```

function `countArray` is called three times. Before each call, make `$a2` as 1/-1/0 and the **stack pointer** is adjusted for 6 items. The function arguments (`$a0`, `$a1`), return address (`$ra`) and saved register (`$s0`, `$s1`, `$s2`) are all saved into the stack before call `countArray` and restored after the function call. According to the return value in `$v0`, the number of `hotDay`/`coldDay`/`comfortDay` are all determined.

- In function `countArray`, define

```

1  $s0 = 0              // as numElements - 1 = 31
2  $s1 = cnt            // as 0

```

Same as before, before function call adjust stack pointer for saving `$a0`, `$a1`, `$a2`, `$ra`, `$s0`, `$s1`. Set `$a0 = A[i]`, where the address of `A[i]` is obtained through `$a0 + $s0 * 4`. Restore all the value after each function call and change `$s1` according to the value in `$v0`.

- For leaf function `hot/cold/comfort`, return the right value, stored in `$v0`.

3 Simulation Result

The simulation result is obtained through *QtSpim*. All the register show decimal value for simpler explanation.

1. Before first call `countArray`

* @	Int Regs [10]	* @	Data	Text
PC	= 4194396	[00400028]	34841000	ori \$4, \$4, 4096 ; 40: ori \$a0, \$a0, 0x1000 # BA of
EPC	= 4194392	tempArray in \$a0		
Cause	= 36	[0040002c]	20050020	addi \$5, \$0, 32 ; 41: addi \$a1, \$zero, 32 # numElements
BadVAddr	= 0	in a1		
Status	= 805371664	[00400030]	00008020	add \$16, \$0, \$0 ; 42: add \$s0, \$zero, \$zero # \$s0 =
		hotDay = 4		
HI	= 0	[00400034]	00008820	add \$17, \$0, \$0 ; 43: add \$s1, \$zero, \$zero # \$s1 =
LO	= 0	[00400038]	00009020	add \$18, \$0, \$0 ; 44: add \$s2, \$zero, \$zero # \$s2 =
		coldDay = 9		
		[0040003c]	20060001	addi \$6, \$0, 1 ; 47: addi \$a2, \$zero, 1 # \$a2 = cntType
		comfortDay = 19		
R0 [r0]	= 0	[00400040]	23bdffe8	addi \$29, \$29, -24 ; 48: addi \$sp, \$sp, -24 # adjust stack
R1 [at]	= 0	for 6 items		
R2 [v0]	= 4	[00400044]	afa50014	sw \$5, 20(\$29) ; 49: sw \$a1, 20(\$sp) # save function
R3 [v1]	= 0	arguments		
R4 [a0]	= 268439552	[00400048]	afa40010	sw \$4, 16(\$29) ; 50: sw \$a0, 16(\$sp)
R5 [a1]	= 32	[0040004c]	afb0000c	sw \$31, 12(\$29) ; 51: sw \$ra, 12(\$sp) # save return
R6 [a2]	= 1	address		
R7 [a3]	= 0	[00400050]	afb20008	sw \$18, 8(\$29) ; 52: sw \$s2, 8(\$sp) # save saved
R8 [t0]	= 0	register		
R9 [t1]	= 0	[00400054]	afb10004	sw \$17, 4(\$29) ; 53: sw \$s1, 4(\$sp)
R10 [t2]	= 0	[00400058]	afb00000	sw \$16, 0(\$29) ; 54: sw \$s0, 0(\$sp)
R11 [t3]	= 0	[0040005c]	0c100040	jal 0x00400100 [countArray]; 55: jal countArray
R12 [t4]	= 0	[00400060]	8fb00000	lw \$16, 0(\$29) ; 56: lw \$s0, 0(\$sp) # restore all the
R13 [t5]	= 0	value		
R14 [t6]	= 0	[00400064]	8fb10004	lw \$17, 4(\$29) ; 57: lw \$s1, 4(\$sp)
R15 [t7]	= 0	[00400068]	8fb20008	lw \$18, 8(\$29) ; 58: lw \$s2, 8(\$sp)
R16 [s0]	= 0	[0040006c]	8fbf000c	lw \$31, 12(\$29) ; 59: lw \$ra, 12(\$sp) # \$ra / \$a0 / \$a1
R17 [s1]	= 0	needed when calling		
R18 [s2]	= 0	countArray again		
R19 [s3]	= 0	[00400070]	8fa40010	lw \$4, 16(\$29) ; 60: lw \$a0, 16(\$sp)
R20 [s4]	= 0	[00400074]	8fa50014	lw \$5, 20(\$29) ; 61: lw \$a1, 20(\$sp)
R21 [s5]	= 0	[00400078]	02028020	add \$16, \$16, \$2 ; 62: add \$s0, \$s0, \$v0
R22 [s6]	= 0			
R23 [s7]	= 0			

we could see that `$a0 = (268439552)10 = 0x10001000`, `$a1 = 32`, `$a2 = 1`, as expected.

2. Before first call hot/cold/comfort

FP Regs	Int Regs [10]	Data	Text
PC = 4194612		[00400104] afa40014 sw \$4, 20(\$29)	; 105: sw \$a0, 20(\$sp)
EPC = 4194608		[00400108] afa50010 sw \$5, 16(\$29)	; 106: sw \$a1, 16(\$sp)
Cause = 36		[0040010c] afa7000c sw \$7, 12(\$29)	; 107: sw \$a3, 12(\$sp)
BadVAddr = 0		[00400110] afbf0008 sw \$31, 8(\$29)	; 108: sw \$ra, 8(\$sp)
Status = 805371664		[00400114] 20b0ffff addi \$16, \$5, -1	; 109: addi \$s0, \$a1, -1 # \$s0 for i =
		numElements - 1	
HI = 0		[00400118] 20110000 addi \$17, \$0, 0	; 110: addi \$s1, \$zero, 0 # \$s1 for cnt =
LO = 0		[0040011c] afb00004 sw \$16, 4(\$29)	; 112: sw \$s0, 4(\$sp) # store \$s0 as i,
		not address	
R0 [r0] = 0		[00400120] afb10000 sw \$17, 0(\$29)	; 113: sw \$s1, 0(\$sp)
R1 [at] = 4		[00400124] 00108080 sll \$16, \$16, 2	; 114: sll \$s0, \$s0, 2 # \$s0 = \$s0 * 4
R2 [v0] = 0		[00400128] 00908020 add \$16, \$4, \$16	; 115: add \$s0, \$a0, \$s0 # \$s0 is the
R3 [v1] = 0		address of A[i]	
R4 [a0] = 33		[0040012c] 8e040000 lw \$4, 0(\$16)	; 116: lw \$a0, 0(\$s0) # \$a0 = A[i]
R5 [a1] = 32		[00400130] 20080001 addi \$8, \$0, 1	; 117: addi \$t0, \$zero, 1 # \$t0 = 1
R6 [a2] = 1		[00400134] 14c80003 bne \$6, \$8, 12 [sCase2-0x00400134]	
R7 [a3] = 0		[00400138] 0c100062 jal 0x00400188 [hot]	; 119: jal hot # jump to hot and save
R8 [t0] = 1		position to \$ra	
R9 [t1] = 0		[0040013c] 08100055 j 0x00400154 [switchBreak]; 120: j switchBreak # jump to	
R10 [t2] = 0		switchBreak	
R11 [t3] = 0		[00400140] 2008ffff addi \$8, \$0, -1	; 122: addi \$t0, \$zero, -1 # \$t0 = -1
R12 [t4] = 0		[00400144] 14c80003 bne \$6, \$8, 12 [sDefault-0x00400144]	
R13 [t5] = 0		[00400148] 0c100068 jal 0x004001a0 [cold]	; 124: jal cold # jump to cold and save
R14 [t6] = 0		position to \$ra	
R15 [t7] = 0		[0040014c] 08100055 j 0x00400154 [switchBreak]; 125: j switchBreak	
R16 [s0] = 268439676		[00400150] 0c100070 jal 0x004001c0 [comfort]; 127: jal comfort # jump to comfort and	
R17 [s1] = 0		save position to \$ra	
R18 [s2] = 0		[00400154] 8fb10000 lw \$17, 0(\$29)	; 129: lw \$s1, 0(\$sp)
R19 [s3] = 0		[00400158] 8fb00004 lw \$16, 4(\$29)	; 130: lw \$s0, 4(\$sp)
R20 [s4] = 0		[0040015c] 8fa7000c lw \$7, 12(\$29)	; 131: lw \$a3, 12(\$sp)
R21 [s5] = 0		[00400160] 8fa50010 lw \$5, 16(\$29)	; 132: lw \$a1, 16(\$sp)
R22 [s6] = 0		[00400164] 8fa40014 lw \$4, 20(\$29)	; 133: lw \$a0, 20(\$sp)
R23 [s7] = 0			

the value of $A[i]$, which is $A[31] = 33$, is load in $\$a0$, as function argument to pass to the leaf function. Now $\$s0$ has the address of $A[31] = (268439676)_{10} = 0x1000107c = 0x10001000 + (32 * 4)_{10}$. Then the loop will continue until all the value in the array are passed.

3. After first call countArray

FP Regs	Int Regs [10]	Data	Text
EPC = 4194444		[00400044] afa50014 sw \$5, 20(\$29)	; 49: sw \$a1, 20(\$sp) # save function
Cause = 36		[00400048] afa40010 sw \$4, 16(\$29)	; 50: sw \$a0, 16(\$sp)
BadVAddr = 0		[0040004c] afbf000c sw \$31, 12(\$29)	; 51: sw \$ra, 12(\$sp) # save return
Status = 805371664		address	
HI = 0		[00400050] afb20008 sw \$18, 8(\$29)	; 52: sw \$s2, 8(\$sp) # save saved
LO = 0		register	
		[00400054] afb10004 sw \$17, 4(\$29)	; 53: sw \$s1, 4(\$sp)
R0 [r0] = 0		[00400058] afb00000 sw \$16, 0(\$29)	; 54: sw \$s0, 0(\$sp)
R1 [at] = 0		[0040005c] 0c100040 jal 0x00400100 [countArray]; 55: jal countArray	
R2 [v0] = 4		[00400060] 8fb00000 lw \$16, 0(\$29)	; 56: lw \$s0, 0(\$sp) # restore all the
R3 [v1] = 0		value	
R4 [a0] = 268439552		[00400064] 8fb10004 lw \$17, 4(\$29)	; 57: lw \$s1, 4(\$sp)
R5 [a1] = 32		[00400068] 8fb20008 lw \$18, 8(\$29)	; 58: lw \$s2, 8(\$sp)
R6 [a2] = -1		[0040006c] 8fbf000c lw \$31, 12(\$29)	; 59: lw \$ra, 12(\$sp) # \$ra / \$a0 / \$a1
R7 [a3] = 0		needed when calling countArray again	
R8 [t0] = 1		[00400070] 8fa40010 lw \$4, 16(\$29)	; 60: lw \$a0, 16(\$sp)
R9 [t1] = 0		[00400074] 8fa50014 lw \$5, 20(\$29)	; 61: lw \$a1, 20(\$sp)
R10 [t2] = 0		[00400078] 02028020 add \$16, \$16, \$2	; 62: add \$s0, \$s0, \$v0
R11 [t3] = 0		[0040007c] 2006ffff addi \$6, \$0, -1	; 65: addi \$a2, \$zero, -1 # \$a2 = cntType
R12 [t4] = 0		-1	
R13 [t5] = 0		[00400080] afa50014 sw \$5, 20(\$29)	; 66: sw \$a1, 20(\$sp)
R14 [t6] = 0		[00400084] afa40010 sw \$4, 16(\$29)	; 67: sw \$a0, 16(\$sp)
R15 [t7] = 0		[00400088] afbf000c sw \$31, 12(\$29)	; 68: sw \$ra, 12(\$sp)
R16 [s0] = 4		[0040008c] afb20008 sw \$18, 8(\$29)	; 69: sw \$s2, 8(\$sp)
R17 [s1] = 0		[00400090] afb10004 sw \$17, 4(\$29)	; 70: sw \$s1, 4(\$sp)
R18 [s2] = 0		[00400094] afb00000 sw \$16, 0(\$29)	; 71: sw \$s0, 0(\$sp)
R19 [s3] = 0		[00400098] 0c100040 jal 0x00400100 [countArray]; 72: jal countArray	
R20 [s4] = 0		[0040009c] 8fb00000 lw \$16, 0(\$29)	; 73: lw \$s0, 0(\$sp)
R21 [s5] = 0		[004000a0] 8fb10004 lw \$17, 4(\$29)	; 74: lw \$s1, 4(\$sp)
R22 [s6] = 0		[004000a4] 8fb20008 lw \$18, 8(\$29)	; 75: lw \$s2, 8(\$sp)
R23 [s7] = 0		[004000a8] 8fbf000c lw \$31, 12(\$29)	; 76: lw \$ra, 12(\$sp)
R24 [t8] = 0		[004000ac] 8fa40010 lw \$4, 16(\$29)	; 77: lw \$a0, 16(\$sp)
R25 [t9] = 0		[004000b0] 8fa50014 lw \$5, 20(\$29)	; 78: lw \$a1, 20(\$sp)

The value of hotDay is stored in \$s0, which is 4. \$a2 is set as -1 for count coldDay. The value of \$a0 and \$a1 should not change as restored from stack. Then the procedure is similar, only differ in the leaf function called.

4. Finish

The screenshot shows a MIPS debugger interface with two main panes. The left pane, titled 'Int Regs [10]', displays the current state of the registers:

- PC = 4194556
- EPC = 4194436
- Cause = 36
- BadVAddr = 0
- Status = 805371664
- HI = 0
- LO = 0
- R0 [r0] = 0
- R1 [at] = 0
- R2 [v0] = 10
- R3 [v1] = 0
- R4 [a0] = 268439552
- R5 [a1] = 32
- R6 [a2] = 0
- R7 [a3] = 0
- R8 [t0] = 1
- R9 [t1] = 1
- R10 [t2] = 5
- R11 [t3] = 0
- R12 [t4] = 0
- R13 [t5] = 0
- R14 [t6] = 0
- R15 [t7] = 0
- R16 [s0] = 4
- R17 [s1] = 9
- R18 [s2] = 19
- R19 [s3] = 0
- R20 [s4] = 0
- R21 [s5] = 0
- R22 [s6] = 0
- R23 [s7] = 0

The right pane, titled 'Text', displays the assembly code being executed:

```

[00400088] afbf000c sw $31, 12($29) ; 68: sw $ra, 12($sp)
[0040008c] afb20008 sw $18, 8($29) ; 69: sw $s2, 8($sp)
[00400090] afb10004 sw $17, 4($29) ; 70: sw $s1, 4($sp)
[00400094] afb00000 sw $16, 0($29) ; 71: sw $s0, 0($sp)
[00400098] 0c100040 jal 0x00400100 [countArray]; 72: jal countArray
[0040009c] 8fb00000 lw $16, 0($29) ; 73: lw $s0, 0($sp)
[004000a0] 8fb10004 lw $17, 4($29) ; 74: lw $s1, 4($sp)
[004000a4] 8fb20008 lw $18, 8($29) ; 75: lw $s2, 8($sp)
[004000a8] 8fbf000c lw $31, 12($29) ; 76: lw $ra, 12($sp)
[004000ac] 8fa40010 lw $4, 16($29) ; 77: lw $a0, 16($sp)
[004000b0] 8fa50014 lw $5, 20($29) ; 78: lw $a1, 20($sp)
[004000b4] 02228820 add $17, $17, $2 ; 79: add $s1, $s1, $v0
[004000b8] 00003020 add $6, $0, $0 ; 82: add $a2, $zero, $zero # $a2 =
cntType = 0
[004000bc] afa50014 sw $5, 20($29) ; 83: sw $a1, 20($sp)
[004000c0] afa40010 sw $4, 16($29) ; 84: sw $a0, 16($sp)
[004000c4] afbf000c sw $31, 12($29) ; 85: sw $ra, 12($sp)
[004000c8] afb20008 sw $18, 8($29) ; 86: sw $s2, 8($sp)
[004000cc] afb10004 sw $17, 4($29) ; 87: sw $s1, 4($sp)
[004000d0] afb00000 sw $16, 0($29) ; 88: sw $s0, 0($sp)
[004000d4] 0c100040 jal 0x00400100 [countArray]; 90: jal countArray
[004000d8] 8fb00000 lw $16, 0($29) ; 91: lw $s0, 0($sp) # no function call
afterwards, restore all
[004000dc] 8fb10004 lw $17, 4($29) ; 92: lw $s1, 4($sp)
[004000e0] 8fb20008 lw $18, 8($29) ; 93: lw $s2, 8($sp)
[004000e4] 8fbf000c lw $31, 12($29) ; 94: lw $ra, 12($sp)
[004000e8] 8fa40010 lw $4, 16($29) ; 95: lw $a0, 16($sp)
[004000ec] 8fa50014 lw $5, 20($29) ; 96: lw $a1, 20($sp)
[004000f0] 02429020 add $18, $18, $2 ; 97: add $s2, $s2, $v0
[004000f4] 23bd0018 addi $29, $29, 24 ; 98: addi $sp, $sp, 24
[004000f8] 2002000a addi $2, $0, 10 ; 99: addi $v0, $zero, 10 # prepare for
exit
[004000fc] 0000000c syscall ; 100: syscall

```

When the program finishes, \$s0 = 4 (the number of hotDays), \$s1 = 9 (the number of coldDays), \$s2 = 19 (the number of comfortDays), as expected.

4 Conclusion

In conclusion, the program written through MIPS successfully finished the expected tasks and output the correct answer. Potential error might caused by

- use pseudo-instruction, caused syntax error.
- forget to adjust the stack pointer to save/restore values before/after each function call.
- the existed delay. Add some meaningless command will help.

A Program

```
1      .data 0x10001000
2  tempArray:
3      .word 36
4      .word 9
5      .word -8
6      .word 40
7      .word 25
8      .word 20
9      .word 18
10     .word 19
11     .word 15
12     .word 16
13     .word 17
14     .word 16
15     .word 15
16     .word 14
17     .word 13
18     .word 12
19     .word 11
20     .word 10
21     .word 9
22     .word 8
23     .word 7
24     .word 6
25     .word 5
26     .word 4
27     .word 3
28     .word 2
29     .word 1
30     .word 0
31     .word -3
32     .word 30
33     .word -19
34     .word 33
35     .text
36     .align 2
37     .globl main
38 main:
39     lui    $a0, 0x1000
40     ori    $a0, $a0, 0x1000    # BA of tempArray in $a0
41     addi   $a1, $zero, 32      # numElements in a1
42     add    $s0, $zero, $zero   # $s0 = hotDay = 4
43     add    $s1, $zero, $zero   # $s1 = coldDay = 9
44     add    $s2, $zero, $zero   # $s2 = comfortDay = 19
45
46     ## First call: hotDay = countArray (tempArray, size, 1);
47     addi   $a2, $zero, 1       # $a2 = cntType = 1
```

```

48     addi    $sp, $sp, -24        # adjust stack for 6 items
49     sw      $a1, 20($sp)        # save function arguments
50     sw      $a0, 16($sp)
51     sw      $ra, 12($sp)        # save return address
52     sw      $s2, 8($sp)         # save saved register
53     sw      $s1, 4($sp)
54     sw      $s0, 0($sp)
55     add     $t0, $t0, $0        # meaningless
56     jal     countArray
57     lw      $s0, 0($sp)         # restore all the value
58     lw      $s1, 4($sp)
59     lw      $s2, 8($sp)
60     lw      $ra, 12($sp)        # $ra / $a0 / $a1 needed when calling
    ↪ countArray again
61     lw      $a0, 16($sp)
62     lw      $a1, 20($sp)
63     add     $s0, $s0, $v0
64
65     ## Second call: coldDay = countArray (tempArray, size, -1);
66     addi    $a2, $zero, -1      # $a2 = cntType = -1
67     sw      $a1, 20($sp)
68     sw      $a0, 16($sp)
69     sw      $ra, 12($sp)
70     sw      $s2, 8($sp)
71     sw      $s1, 4($sp)
72     sw      $s0, 0($sp)
73     add     $t0, $t0, $0        # meaningless
74     jal     countArray
75     lw      $s0, 0($sp)
76     lw      $s1, 4($sp)
77     lw      $s2, 8($sp)
78     lw      $ra, 12($sp)
79     lw      $a0, 16($sp)
80     lw      $a1, 20($sp)
81     add     $s1, $s1, $v0
82
83
84     ## comfortDay = countArray (tempArray, size, 0);
85     add     $a2, $zero, $zero   # $a2 = cntType = 0
86     sw      $a1, 20($sp)
87     sw      $a0, 16($sp)
88     sw      $ra, 12($sp)
89     sw      $s2, 8($sp)
90     sw      $s1, 4($sp)
91     sw      $s0, 0($sp)
92     ## only $s1 has changed, store this only
93     add     $t0, $t0, $0        # meaningless
94     add     $t0, $t0, $0        # meaningless
95     jal     countArray

```

```

96     lw      $s0, 0($sp)           # no function call afterwards, restore all
97     lw      $s1, 4($sp)
98     lw      $s2, 8($sp)
99     lw      $ra, 12($sp)
100    lw      $a0, 16($sp)
101    lw      $a1, 20($sp)
102    add      $s2, $s2, $v0
103    addi     $sp, $sp, 24
104    addi     $v0, $zero, 10        # for exit
105    syscall
106
107    ### Function countArray ###
108    countArray:
109        addi     $sp, $sp, -24      # adjust the stack for 6 items
110        sw      $a0, 20($sp)       # save function arguments
111        sw      $a1, 16($sp)
112        sw      $a3, 12($sp)
113        sw      $ra, 8($sp)        # save return address
114        addi     $s0, $a1, -1       # $s0 for i = numElements - 1
115        addi     $s1, $zero, 0     # $s1 for cnt = 0
116    cntLoop:
117        sw      $s0, 4($sp)        # store $s0 as i, not address
118        sw      $s1, 0($sp)
119        sll      $s0, $s0, 2       # $s0 = $s0 * 4
120        add      $s0, $a0, $s0     # $s0 is the address of A[i]
121        lw      $a0, 0($s0)       # $a0 = A[i]
122        addi     $t0, $zero, 1     # $t0 = 1
123        bne      $a2, $t0, sCase2  # if $a2 != 1, then not hot
124        jal      hot               # jump to hot and save position to $ra
125        j        switchBreak      # jump to switchBreak
126    sCase2:
127        addi     $t0, $zero, -1     # $t0 = -1
128        bne      $a2, $t0, sDefault # if $a2 != -1, then not cold
129        jal      cold              # jump to cold and save position to $ra
130        j        switchBreak
131    sDefault:
132        add      $t0, $t0, $0       # meaningless
133        jal      comfort           # jump to comfort and save position to
        ↪ $ra
134    switchBreak:
135        lw      $s1, 0($sp)
136        lw      $s0, 4($sp)
137        lw      $a3, 12($sp)
138        lw      $a1, 16($sp)
139        lw      $a0, 20($sp)
140        add      $s1, $s1, $v0     # cnt += $v0
141        addi     $s0, $s0, -1      # i--
142        slti     $t0, $s0, 0       # if $s0 < 0, $t0 = 0
143        beq      $t0, $zero, cntLoop # if $t0 != 0 then continue the loop

```



```

144     lw      $ra, 8($sp)           # else, exit the loop, restore $ra
145     add     $v0, $s1, $zero       # $v0 = cnt
146     addi    $sp, $sp, 24          # destroy spaces on stack
147     jr      $ra
148
149
150     ### Function hot ###
151 hot:
152     slti    $v0, $a0, 30
153     beq     $v0, $zero, hotTrue
154     add     $v0, $zero, $zero     # A[i] < 30, $v0 = 0
155     jr      $ra
156 hotTrue:
157     addi    $v0, $zero, 1         # A[i] >= 30, $v0 = 1
158     jr      $ra
159
160     ### Function cold ###
161 cold:
162     slti    $t0, $a0, 5           # A[i] < 5, $t0 = 1
163     beq     $t0, $zero, coldFalse
164 coldTrue:
165     addi    $v0, $zero, 1         # $v0 = 1
166     jr      $ra
167 coldFalse:
168     addi    $t1, $zero, 5
169     beq     $a0, $t1, coldTrue    # if $a0 == $t1 == 5 then coldTrue
170     add     $v0, $zero, $zero     # $v0 = $zero + $zero = 0
171     jr      $ra
172
173     ### Function comfort ###
174 comfort:
175     slti    $t0, $a0, 30
176     addi    $t1, $zero, 1
177     beq     $t0, $t1, comfortTrue # if $t0 == 1, $a0 < 30, then
    ↪ comfortTrue
178 comfortFalse:
179     add     $v0, $zero, $zero
180     jr      $ra
181 comfortTrue:
182     slti    $t0, $a0, 5
183     beq     $t0, $t1, comfortFalse # if $t0 == 1, $a0 < 5 then
    ↪ comfortFalse
184     addi    $t2, $zero, 5
185     beq     $a0, $t2, comfortFalse # if $a0 == 5 comfortFalse
186     addi    $v0, $zero, 1
187     jr      $ra
188

```