Hacettepe University

Computer Engineering Department

BM234 Computer Organization - 2021 Spring

MIPS Project Report

April 17, 2021

Student name:
İlayda Atmaca

Student Number:
b21827101

1 Problem Definition

In this project we have a task and this task is to help it find cheaters with mips code.While writing the mips code, we have to take into account many functions we have to do and the array that includes the grades of the students given to us. Also we need to consider some limitations in this project.

2 Solution Implementation

```
1
2       .data
3   arr:    .word 0, 1, 5, 400, 112, 17, 7, 0, 560, 13, 0, 11, 3, 5, 0
4       .text
5       .globl main
6
7       .data
8   output: .asciiz "The average similarity score is: "
9       .text
10      .globl main2
11
12  # $s1 = datasize, $s0 = array base address
13  # $s2 = i
14
15  #initialization code
16
17  addi $s2, $0, 0 # i=0
18
19  main:
20      la  $t1, arr  #address of the array
21
22  loop:
23      sub $t0, $s2, $s1 #i-datasize
24      bne $t0, $0, if   #if i - datasize() is 0,outside the loop
25      li $s2, 0
26      li $t0, 0       #clear registers
27      li $t4, 0
28      li $s0, 0
29
30      jal main2   #call main2
31
32  if:
33      sll $t0, $s2, 2
34      addu $t0, $t0, $t1
35      lw $t2, 0($t0) # calc address of data[i]
36
37      rem $t3, $t2, 2    #it is remainder,it gives remain (data[i]%2)
38      bne $t3, $0, else  #if it is not even,go to else block
39      sra $t2, $t2, 3    #data[i]/8
40      j end
41
```

```
42
43 else:
44     sll $t4, $t2, 2  #$t4 = data[i]*4
45     sll $t5, $t2, 0  #$t5 = data[i]*1
46     add $t2, $t4, $t5
47     j end
48
49 end:
50     sw $t2, 0($t0)
51     addi $s2, $s2, 1 #i=i+1
52     j loop           #restart loop
53
54
55 #$s0=sum, $s1=avg
56 #$a1=n, $a2=data
57 main2:
58     la  $a0, arr #address of arr array
59
60     add $a1,$s1,$0    #n=datasize
61     li $s1, 0
62     jal average_recursive #call part 2
63
64     li $v0, 4
65     la $a0, output #print "The average..." string
66     syscall
67
68     add $a0, $v1, $0 # Print average.
69         li $v0, 1
70         syscall
71
72     li $v0, 10
73     syscall # Execute the 'exit' syscall
74
75 average_recursive:
76
77     addi $sp, $sp, -12# make space on stack to store three registers
78     sw $ra, 0($sp)#caller saved
79     sw $s2, 4($sp) #save data[n-1]
80     sw $a1, 8($sp) #save n
81
82     beq $a1, 1, basecase #if n=1,go to the basecase
83
84     sub $t0, $a1, 1 #[size-1]
85     mul  $t0,  $t0, 4
86     addu  $t0,  $t0, $a0
87     lw $s2, 0($t0) #data[n-1]
88
89
90     sub $a1, $a1, 1 #n=n-1
91     jal average_recursive # call again with n-1,data  ,recursive call
92
93     mul $t4, $a1,$v0 #(n-1)*average_recursive
```

```
94      addu $s0, $s2, $t4 #added data[n-1] to $t4
95
96      jal average_recursive_end
97
98  basecase:
99      lw $s0, 0($a0) #sum = data[0]
100     jal average_recursive_end  #call average_recursive_end
101
102 average_recursive_end:
103     lw $a1, 8($sp)  #restore $a1 from stack
104
105     div $s1, $s0, $a1 #avg = sum /n
106
107     addi $v0, $s1, 0  #put the avg value in the $v0
108     addi $v1, $s1, 0
109
110     lw $ra, 0($sp) #restore $ra from stack
111     lw $s2, 4($sp)  #restore $s2 from stack
112     addi $sp, $sp, 12 # deallocate stack space
113
114     jr $ra #return to caller
```

3 How you used function calls (jal instructions)?– How you used stack during function calls and why?

Briefly state the problem that you are trying to solve. Add any background information if necessary. MIPS uses the jump-and-link instruction jal to call functions. In this part, I should implement recursive function.In function average-recursive calls average-recursive.So I used jal instruction for recursive call.The jal instruction continued until n equals 1.

Code shows an improved version that saves and restores ra, a2, and s1. average-recursive makes room for three words on the stack by decrementing the stack pointer sp by 12. It then stores the current values of ra, a2, and s1.in the newly allocated space. It executes the rest of the function, changing the values in these three registers. At the end of the function, function restores the values of ra, a2, and s1 from the stack, deallocates its stack space, and returns. When the function returns, v0 holds the result, but there are no other side effects: ra, a2, and s1 and sp have the same values as they did before the function call .The stack is memory that is used to save local variables within a function. We have overwrites registers so we have to use stack.

4 Results

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	720	480	80	3	1	0	543516756	1919252065
0x10010020	543516513	1768778099	1769103724	1931508084	1701998435	980642080	32	0
0x10010040	0	0	0	0	0	0	0	0

Figure 1: Test 1 Data Segment before

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	90	60	10	15	5	0	543516756	1919252065
0x10010020	543516513	1768778099	1769103724	1931508084	1701998435	980642080	32	0
0x10010040	0	0	0	0	0	0	0	0

Figure 2: Test 1 Data Segment after

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194304
hi		0
lo		0

Figure 3: Test 1 Register before

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	29
$a0	4	29
$a1	5	6
$a2	6	0
$a3	7	0
$t0	8	268500996
$t1	9	268500992
$t2	10	0
$t3	11	0
$t4	12	175
$t5	13	1
$t6	14	0
$t7	15	0
$s0	16	175
$s1	17	29
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	4194428
pc		4194464
hi		1
lo		29

Figure 4: Test 1 Register after

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	1	5	400	112	17	7	0
0x10010020	560	13	0	11	3	5	0	543516756
0x10010040	1919252065	543516513	1768778099	1769103724	1931508084	1701998435	980642080	32
0x10010060	0	0	0	0	0	0	0	0

Figure 5: Test 2 Data Segment before

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	5	25	50	14	85	35	0
0x10010020	70	65	0	55	15	25	0	543516756
0x10010040	1919252065	543516513	1768778099	1769103724	1931508084	1701998435	980642080	32
0x10010060	0	0	0	0	0	0	0	0

Figure 6: Test 2 Data Segment after

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194304
hi		0
lo		0

Figure 7: Test 2 Register before

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	27
$a0	4	27
$a1	5	15
$a2	6	0
$a3	7	0
$t0	8	268500996
$t1	9	268500992
$t2	10	0
$t3	11	0
$t4	12	406
$t5	13	5
$t6	14	0
$t7	15	0
$s0	16	406
$s1	17	27
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	4194428
pc		4194464
hi		1
lo		27

Figure 8: Test 2 Register after

References

- Digital Design and Computer Architecture