BBM234: Computer Organization
2019-2020 Spring
MIPS Project Report

İbrahim Burak Tanrıkulu, b21827852

April 15, 2020

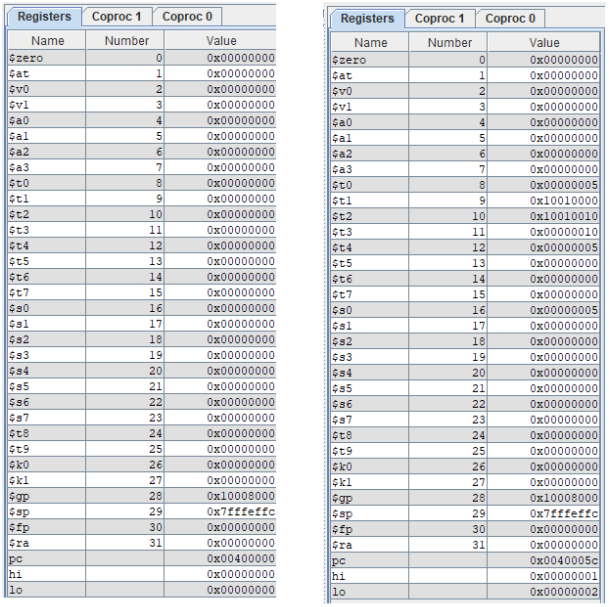
1 Problem: Arrays Using for Loops

MIPS Assembly code:

```
.data
A: .word 1, 2, 3, 4, 5       # test 1
#A: .word 2, 4, 10, 0, 50    # test 2
#A: .word 1, 3, 5, 9, 13     # test 3
.text
main: la $t1, A

addi $s0, $0, 0      # i = 0
addi $t0, $0, 5
# $s0 = i , $t0 = 5 , $t1 = array base address
for:    beq $s0, $t0, done
        sll $t2, $s0, 2      # byte offset
        add $t2, $t2, $t1    # $t2 = address of array[i]
        lw $t3, 0($t2)       # $t3 = value of array[i]
        rem $t4, $t3, 2      # $t4 = array[i] % 2
        if:     bne $t4, 0, else
                sra $t3, $t3, 1
                j endIf
        else:   addi $t4, $t4, -1   # we don't need remainder, so reset it. $t4 = 0
                add $t4, $t4, $t3   # $t4 = array[i]
                sll $t3, $t3, 1     # array[i] * 3 = (array[i] * 2) + array[i]
                add $t3, $t3, $t4   # $t3 = array[i] * 3
                addi $t3, $t3, 1    # $t3 = (array[i]*3)+1
        endIf:
        sw $t3, 0($t2)
        addi $s0, $s0, 1
        j for
done:
```

Firstly, i created two variables for "for loop".
s0 is an integer variable "i" and t0 is max value of "i" (5)
In every loop, i increased "i" and when "i" reaches 5, jumped to out of the for loop.
– if "i" is equal "5", then jump "done" .
In loop; there is if-else statement. I used "bne" instruction for that.
I used "rem" instruction for calculating array[i] mod 2



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000005
$t1	9	0x10010000
$t2	10	0x10010010
$t3	11	0x00000010
$t4	12	0x00000005
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000005
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0040005c
hi		0x00000001
lo		0x00000002

Figure 1: Test 1: Registers before and after

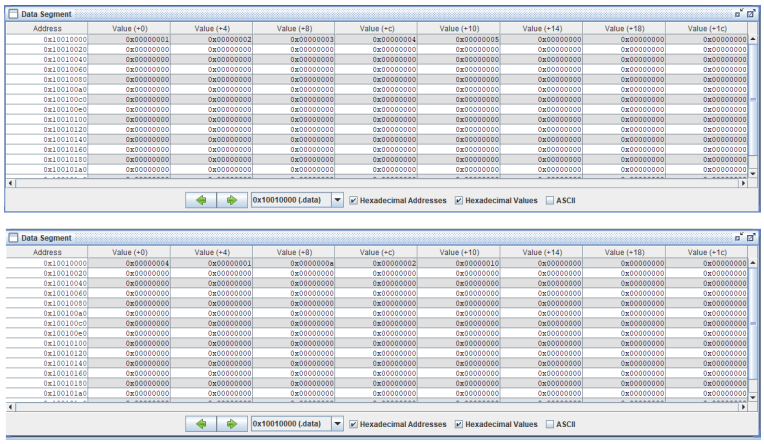


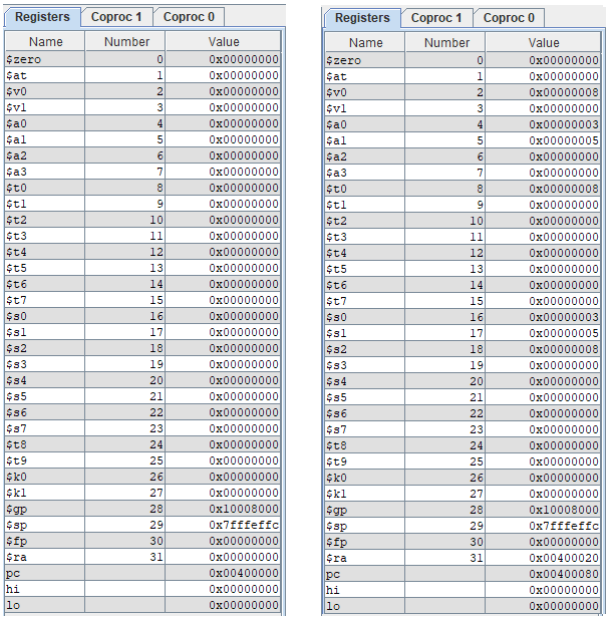
Figure 2: Test 1: Data before and after

2 Problem: Function Calls

MIPS Assembly code:

```
# $s0 = a , $s1 = b , $s2 = result
# test 1:
#addi $s0, $0, 8
#addi $s1, $0, 8
# test 2:
addi $s0, $0, 3
addi $s1, $0, 5
# test 3:
#addi $s0, $0, 5
#addi $s1, $0, 3

if:     bne $s0, $s1, else  # if a = b
        mul $s2, $s0, $s1
        j done
else:   add $a0, $s0, $0    # else return assess
        add $a1, $s1, $0
        jal assess
        add $s2, $v0, $0
        j done
assess:
        assessIf:   ble $a0, $a1, assessElse    # if b < a
                    addi $sp, $sp, -4
                    sw $ra, 0($sp)  # store return address
                    jal upgrade
                    lw $ra, 0($sp)  # load return address
                    addi $sp, $sp, 4
                    jr $ra
        assessElse: addi $sp, $sp, -4           # b >= a
                    sw $ra, 0($sp)  # store return address
                    jal demote
                    lw $ra, 0($sp)  # load return address
                    addi $sp, $sp, 4
                    jr $ra
upgrade:    # upgrade function
        add $t0, $a0, $a1
        sll $t0, $t0, 2
        add $v0, $t0, $0
        jr $ra
demote:     # demote function
        sub $t0, $a1, $a0
        sll $t0, $t0, 2
        add $v0, $t0, $0
        jr $ra
done:
```



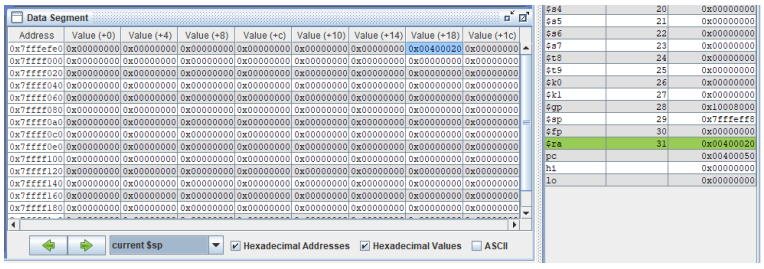
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000008
$v1	3	0x00000000
$a0	4	0x00000003
$a1	5	0x00000005
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000008
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000003
$s1	17	0x00000005
$s2	18	0x00000008
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400020
pc		0x00400080
hi		0x00000000
lo		0x00000000

Figure 3: Test 2: Registers before and after



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00400020	0x00000000
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

current $sp | Hexadecimal Addresses | Hexadecimal Values | ASCII

$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeff8
$fp	30	0x00000000
$ra	31	0x00400020
pc		0x00400050
hi		0x00000000
lo		0x00000000

Figure 4: Test 2: Storing ra to sp