



San Francisco Bay University

EE488 - Computer Architecture Homework Assignment #3

Due day:
3/20/2025

Instruction:

1. Implement a program (MIPS Assembly) which multiplies user input by 10 using only bit shift operations and addition. Check to see if your program is correct by using the *mult* and *mflo* operators. Your program should include a proper and useful prompt for input, and print the results meaningfully.

⇒ Answer:

⇒ The code:

```
.data
prompt:      .asciiz "Enter an integer
to multiply by 10: "
result1:     .asciiz "Result using shift
and add: "
result2:     .asciiz "\nResult using
mult and mflo: "
newline:     .asciiz "\n"

.text
.globl main

main:
# Prompt user for input
li $v0, 4
la $a0, prompt
```

```
syscall

    # Read integer
    li $v0, 5
    syscall
    move $t0, $v0          # Store input
in $t0

        ##### Method 1: Multiply using
shift and add #####
    sll $t1, $t0, 3        # $t1 = input
* 8
    sll $t2, $t0, 1        # $t2 = input
* 2
    add $t3, $t1, $t2      # $t3 = input
* 10

    # Print result1
    li $v0, 4
    la $a0, result1
    syscall

    li $v0, 1
    move $a0, $t3
    syscall

        ##### Method 2: Multiply using
mult and mflo #####
    li $t5, 10              # Load 10 into
register
    mult $t0, $t5           # Multiply
input * 10
    mflo $t4                # Store result
in $t4

    # Print result2
    li $v0, 4
    la $a0, result2
    syscall
```

```

    li $v0, 1
    move $a0, $t4
    syscall

    # Print newline
    li $v0, 4
    la $a0, newline
    syscall

    # Exit
    li $v0, 10
    syscall

```

The screenshot shows the MARS 4.5 assembly debugger interface. The assembly code in the editor window is:

```

1 .data
2 prompt: .asciiz "Enter an integer to multiply by 10: "
3 result1: .asciiz "Result using shift and add: "
4 result2: .asciiz "\nResult using mult and mflo: "
5 newline: .asciiz "\n"
6
7 .text
8 .globl main
9
10 main:
11     # Prompt user for input
12     li $v0, 5
13     la $a0, prompt
14     syscall
15
16     # Read integer
17     li $v0, 5
18     syscall
19     move $t0, $v0      # Store input in $t0
20
21 ##### Method 1: Multiply using shift and add #####
22 sll $t1, $t0, 3      # $t1 = input * 8
23 sll $t2, $t0, 1      # $t2 = input * 2
24 add $t3, $t1, $t2    # $t3 = input * 10
25

```

The Registers window shows the following state:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000003
\$v1	3	0x00000000
\$a0	4	0x10010003
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000002
\$t1	9	0x00000010
\$t2	10	0x00000004
\$t3	11	0x00000014
\$t4	12	0x00000014
\$t5	13	0x00000003
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t8	16	0x00000000
\$t9	17	0x00000000
\$k0	18	0x00000000
\$k1	19	0x00000000
\$gp	20	0x00000000
\$sp	21	0x00000000
\$fp	22	0x00000000
\$ra	23	0x00000000
pc	24	0x00000000
hi	25	0x00000000
lo	26	0x7fffffc
	27	0x10000000
	28	0x10000000
	29	0x00000000
	30	0x00000000
	31	0x00000000

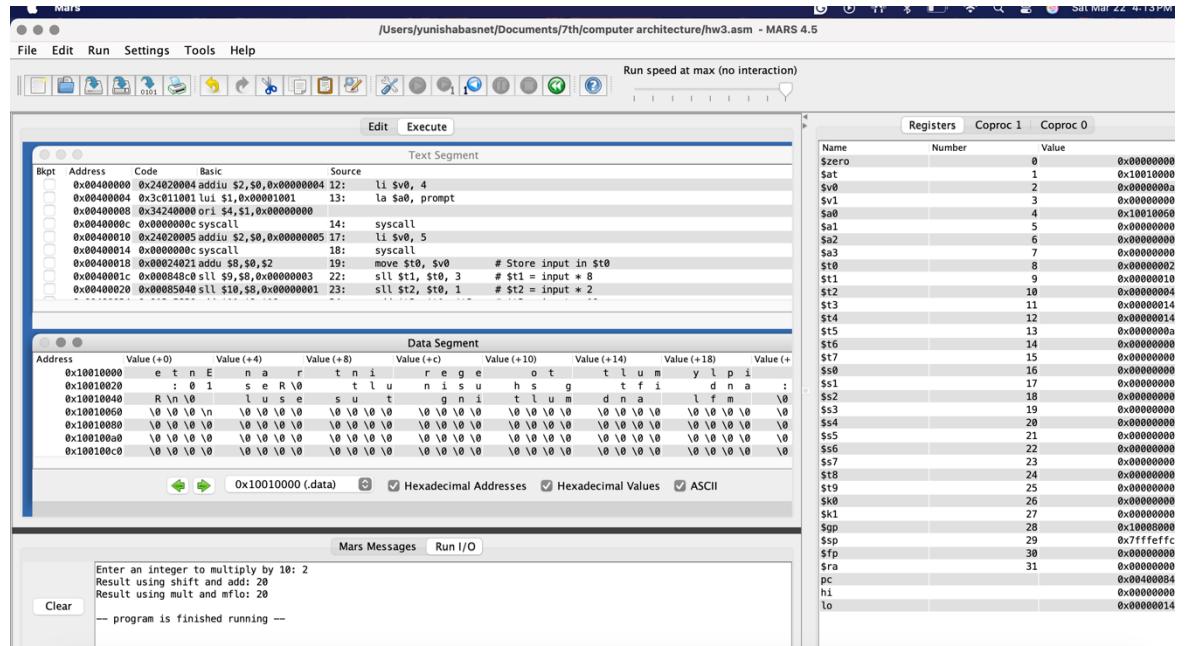
The Mars Messages window shows the output:

```

Enter an integer to multiply by 10: 2
Result using shift and add: 20
Result using mult and mflo: 20
-- program is finished running --

```

⇒ The output:



2. Write programs (MIPS Assembly) to evaluate the following expressions. The user should enter the variables, and the program should print back an answer. Prompt the user for all variables in the expression, and print the results in a meaningful manner. The results should be as accurate as possible.

$$a. \quad 5x + 3y + z$$

⇒ The answer:

⇒ The code:

```
.data
prompt_x:    .asciiz "Enter value for x:
"
```

```
prompt_y:    .asciiz "Enter value for y:
"
```

```
prompt_z:    .asciiz "Enter value for z:
"
```

```
result:      .asciiz "Result of 5x + 3y
+ z = "
```

```
newline:     .asciiz "\n"
```

```
.text
```

```
.globl main
main:
    # Prompt x
    li $v0, 4
    la $a0, prompt_x
    syscall
    li $v0, 5
    syscall
    move $t0, $v0      # x in $t0

    # Prompt y
    li $v0, 4
    la $a0, prompt_y
    syscall
    li $v0, 5
    syscall
    move $t1, $v0      # y in $t1

    # Prompt z
    li $v0, 4
    la $a0, prompt_z
    syscall
    li $v0, 5
    syscall
    move $t2, $v0      # z in $t2

    # Compute 5x → 5 * $t0
    li $t3, 5
    mul $t4, $t3, $t0  # $t4 = 5x

    # Compute 3y → 3 * $t1
    li $t5, 3
    mul $t6, $t5, $t1  # $t6 = 3y

    # Add: 5x + 3y
    add $t7, $t4, $t6

    # Add z: 5x + 3y + z
```

```
add $t8, $t7, $t2
```

```
# Print result
li $v0, 4
la $a0, result
syscall
```

```
li $v0, 1
move $a0, $t8
syscall
```

```
# Print newline
li $v0, 4
la $a0, newline
syscall
```

```
# Exit
li $v0, 10
syscall
```

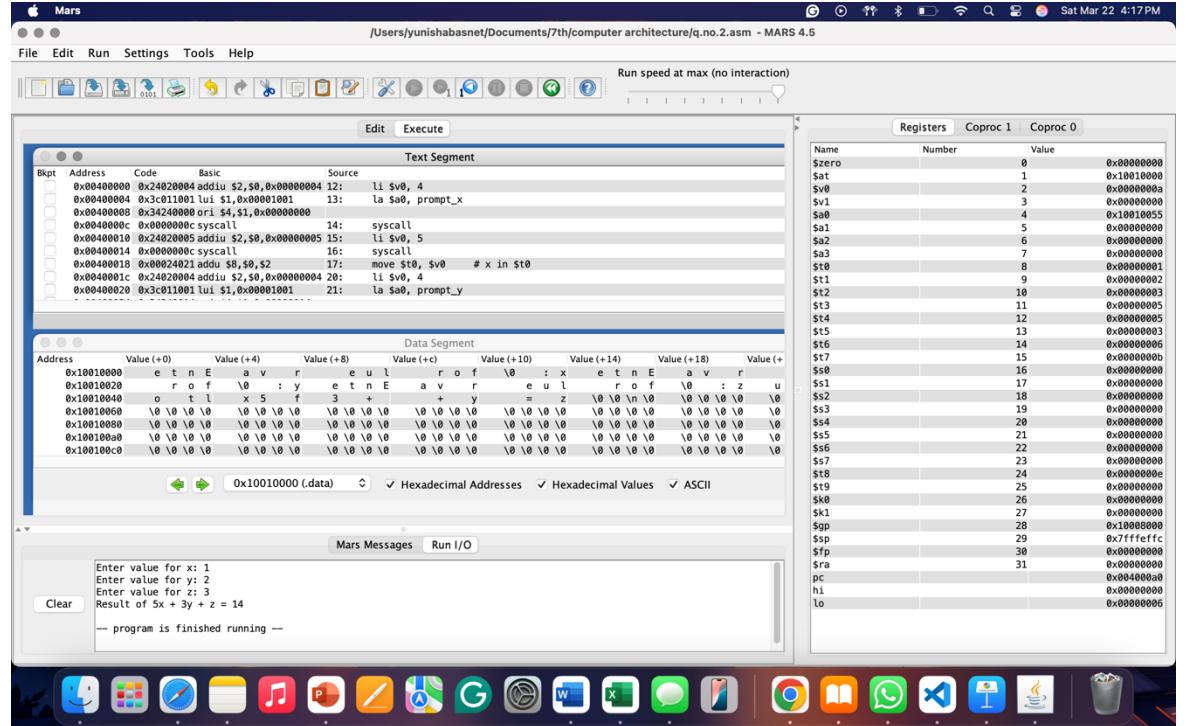
The screenshot shows the MARS 4.5 assembly debugger interface. The assembly code in the editor window is:

```
1 .data
2 prompt_x: .ascii "Enter value for x: "
3 prompt_y: .ascii "Enter value for y: "
4 prompt_z: .ascii "Enter value for z: "
5 result: .ascii "Result of 5x + 3y + z = "
6 newline: .ascii "\n"
7
8 .text
9 .globl main
10
11 main:
12     # Prompt x
13     li $v0, 4
14     la $a0, prompt_x
15     syscall
16     li $v0, 5
17     syscall
18     move $t0, $v0      # x in $t0
19
20     # Prompt y
21     li $v0, 4
22     la $a0, prompt_y
23     syscall
24     li $v0, 5
25     syscall
```

The registers window on the right shows the following values:

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	0x10000000

⇒ The output:



$$b. \quad \left(\frac{5x+3y+z}{2} \right) * 3$$

⇒ The answer:

⇒ The code:

```

.data
prompt_x:    .asciiz "Enter value for x:
"
prompt_y:    .asciiz "Enter value for y:
"
prompt_z:    .asciiz "Enter value for z:
"
result:      .asciiz "Result of ((5x +
3y + z)/2) * 3 = "
newline:     .asciiz "\n"

.text
.globl main

```

main:

```

# Prompt x
li $v0, 4
la $a0, prompt_x
syscall

```

```

    li $v0, 5
    syscall
    move $t0, $v0      # x

    # Prompt y
    li $v0, 4
    la $a0, prompt_y
    syscall
    li $v0, 5
    syscall
    move $t1, $v0      # y

    # Prompt z
    li $v0, 4
    la $a0, prompt_z
    syscall
    li $v0, 5
    syscall
    move $t2, $v0      # z

    # 5x
    li $t3, 5
    mul $t4, $t3, $t0    # $t4 = 5x

    # 3y
    li $t5, 3
    mul $t6, $t5, $t1    # $t6 = 3y

    # sum = 5x + 3y
    add $t7, $t4, $t6

    # sum = sum + z
    add $t7, $t7, $t2    # $t7 = 5x +
    3y + z

    # Divide by 2
    li $t8, 2
    div $t7, $t8

```

```
        mflo $t9          # $t9 = (5x +
3y + z) / 2

        # Multiply by 3
        li $t3, 3
        mul $s0, $t9, $t3    # $s0 = final
result

        # Print result
        li $v0, 4
        la $a0, result
        syscall

        li $v0, 1
        move $a0, $s0
        syscall

        # Newline
        li $v0, 4
        la $a0, newline
        syscall

        # Exit
        li $v0, 10
        syscall
```

The screenshot shows the Mars 4.5 assembly debugger interface. The assembly code in the editor window is:

```

52:    div $t7, $t8
53:    mflo $t9          # $t9 = (5x + 3y + z) / 2
54:
55:    # Multiply by 3
56:    li $t3, 3
57:    mul $s0, $t9, $t3  # $s0 = final result
58:
59:    # Print result
60:    li $v0, 4
61:    la $a0, result
62:    syscall
63:
64:    li $v0, 1
65:    move $a0, $s0
66:    syscall
67:
68:    # Newline
69:    li $v0, 4
70:    la $a0, newline
71:    syscall
72:
73:    # Exit
74:    li $v0, 10
75:    syscall
76

```

The Registers window shows the following initial values:

Name	Number	Value
\$zero	0	0x000000
\$at	1	0x000000
\$v0	2	0x000000
\$v1	3	0x000000
\$a0	4	0x000000
\$a1	5	0x000000
\$a2	6	0x000000
\$a3	7	0x000000
\$t0	8	0x000000
\$t1	9	0x000000
\$t2	10	0x000000
\$t3	11	0x000000
\$t4	12	0x000000
\$t5	13	0x000000
\$t6	14	0x000000
\$t7	15	0x000000
\$s0	16	0x000000
\$s1	17	0x000000
\$s2	18	0x000000
\$s3	19	0x000000
\$s4	20	0x000000
\$s5	21	0x000000
\$s6	22	0x000000
\$t7	23	0x000000
\$t8	24	0x000000
\$t9	25	0x000000
\$k0	26	0x000000
\$k1	27	0x000000
\$gp	28	0x100000
\$sp	29	0x7ffffe
\$fp	30	0x000000
\$ra	31	0x000000
pc		0x000000
hi		0x000000
lo		0x000000

The Mars Messages window shows:

- Go: execution completed successfully.
- Assemble: assembling /Users/yunishabasnet/Documents/7th/computer architecture/2(B).asm
- Assemble: operation completed successfully.

⇒ The output:

The screenshot shows the Mars 4.5 assembly debugger interface. The assembly code in the editor window is identical to the previous one:

```

52:    div $t7, $t8
53:    mflo $t9          # $t9 = (5x + 3y + z) / 2
54:
55:    # Multiply by 3
56:    li $t3, 3
57:    mul $s0, $t9, $t3  # $s0 = final result
58:
59:    # Print result
60:    li $v0, 4
61:    la $a0, result
62:    syscall
63:
64:    li $v0, 1
65:    move $a0, $s0
66:    syscall
67:
68:    # Newline
69:    li $v0, 4
70:    la $a0, newline
71:    syscall
72:
73:    # Exit
74:    li $v0, 10
75:    syscall
76

```

The Registers window shows the same initial values as before.

The Text Segment window shows the assembly code with line numbers and source file information.

The Data Segment window shows the memory dump at address 0x10010000:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+22)
0x10010000	e	t	n	E	a	v	r	
0x10010020	r	o	f	\0	:	y	e	u
0x10010040	o	l	()	+	x	5	y
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0

The Mars Messages window shows:

- Enter value for x: 1
- Enter value for y: 2
- Enter value for z: 3
- Result of ((5x + 3y + z)/2) * 3 = 21
- program is finished running —

$$c. \quad x^3 + 2x^2 + 3x + 4$$

⇒ The answer:

⇒ The code:

.data

```

prompt_x:    .asciiiz "Enter value for x:
"
result:      .asciiiz "Result of x^3 +
2x^2 + 3x + 4 = "
newline:     .asciiiz "\n"

.text
.globl main

main:
    # Prompt x
    li $v0, 4
    la $a0, prompt_x
    syscall
    li $v0, 5
    syscall
    move $t0, $v0      # x in $t0

    # x^2 = x * x
    mul $t1, $t0, $t0      # $t1 = x^2

    # x^3 = x^2 * x
    mul $t2, $t1, $t0      # $t2 = x^3

    # 2x^2 = 2 * x^2
    li $t3, 2
    mul $t4, $t3, $t1      # $t4 = 2x^2

    # 3x = 3 * x
    li $t5, 3
    mul $t6, $t5, $t0      # $t6 = 3x

    # Add all: x^3 + 2x^2 + 3x + 4
    add $t7, $t2, $t4      # x^3 + 2x^2
    add $t7, $t7, $t6      # + 3x
    li $t8, 4
    add $t9, $t7, $t8      # + 4 → final
result

```

```

# Print result
li $v0, 4
la $a0, result
syscall

li $v0, 1
move $a0, $t9
syscall

# Newline
li $v0, 4
la $a0, newline
syscall

# Exit
li $v0, 10
syscall

```

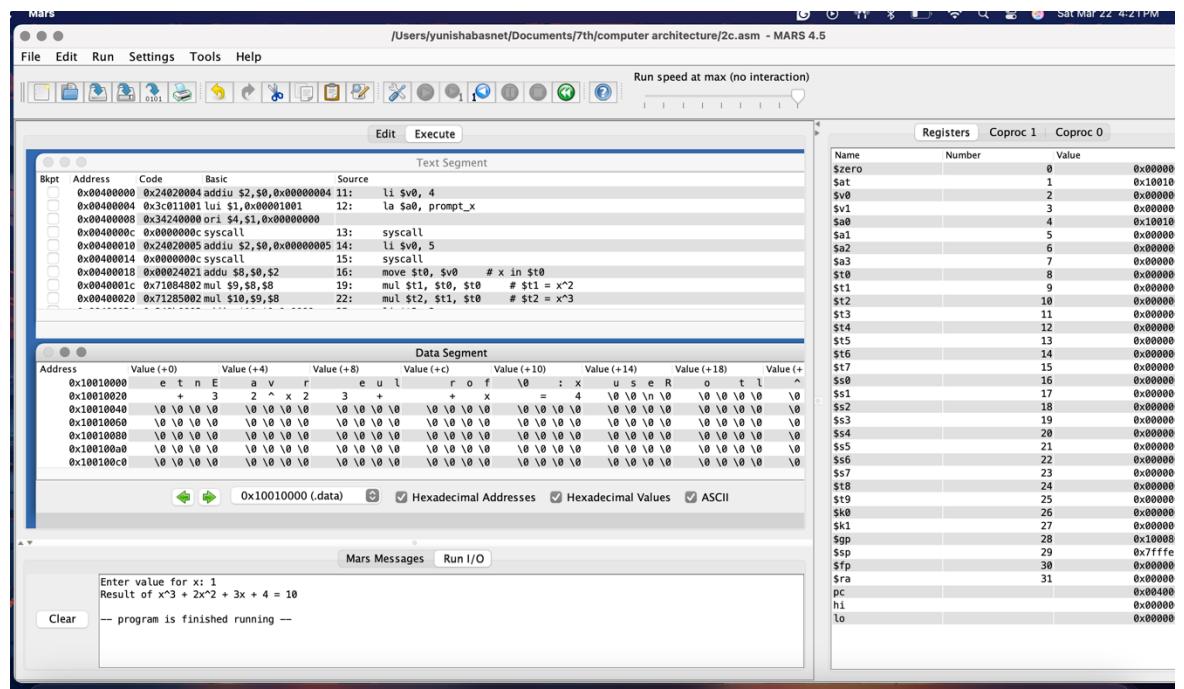
The screenshot shows the MARS 4.5 assembly debugger interface. The assembly code window displays the provided assembly code. The registers window shows the following register values:

Name	Number	Value
\$zero	0	0x000000
\$at	1	0x000000
\$v0	2	0x000000
\$v1	3	0x000000
\$a0	4	0x000000
\$a1	5	0x000000
\$a2	6	0x000000
\$a3	7	0x000000
\$t0	8	0x000000
\$t1	9	0x000000
\$t2	10	0x000000
\$t3	11	0x000000
\$t4	12	0x000000
\$t5	13	0x000000
\$t6	14	0x000000
\$t7	15	0x000000
\$s0	16	0x000000
\$s1	17	0x000000
\$s2	18	0x000000
\$s3	19	0x000000
\$s4	20	0x000000
\$s5	21	0x000000
\$s6	22	0x000000
\$s7	23	0x000000
\$t8	24	0x000000
\$t9	25	0x000000
\$k0	26	0x000000
\$k1	27	0x000000
\$gp	28	0x100008
\$sp	29	0xfffffe
\$fp	30	0x000000
\$ra	31	0x000000
pc		0x000000
hi		0x000000
lo		0x000000

The messages window shows:

- Go: execution completed successfully.
- Assemble: assembling /Users/yunishabasnet/Documents/7th/computer architecture/2c.asm
- Assemble: operation completed successfully.

⇒ The output:



$$d. \quad \left(\frac{4x}{3}\right) * y$$

⇒ The answer:

⇒ The code:

```

.data
prompt_x:    .asciiz "Enter value for x:
"
prompt_y:    .asciiz "Enter value for y:
"
result:      .asciiz "Result of ((4x) /
3) * y = "
newline:     .asciiz "\n"

.text
.globl main

main:
# Prompt x
li $v0, 4
la $a0, prompt_x
syscall
li $v0, 5
syscall
move $t0, $v0      # x in $t0

```

```

# Prompt y
li $v0, 4
la $a0, prompt_y
syscall
li $v0, 5
syscall
move $t1, $v0      # y in $t1

# 4x
li $t2, 4
mul $t3, $t2, $t0      # $t3 = 4x

# (4x) / 3
li $t4, 3
div $t3, $t4
mflo $t5                # $t5 = (4x)
/ 3

# * y
mul $t6, $t5, $t1      # $t6 = ((4x))
/ 3) * y

# Print result
li $v0, 4
la $a0, result
syscall

li $v0, 1
move $a0, $t6
syscall

# Newline
li $v0, 4
la $a0, newline
syscall

# Exit
li $v0, 10

```

syscall

```

1 .data
2 prompt_x: .ascii "Enter value for x: "
3 prompt_y: .ascii "Enter value for y: "
4 result: .ascii "Result of ((4x) / 3) * y = "
5 newline: .ascii "\n"
6
7 .text
8 .globl main
9
10 main:
11     # Prompt x
12     li $v0, 4
13     la $a0, prompt_x
14     syscall
15     li $v0, 5
16     syscall
17     move $t0, $v0      # x in $t0
18
19     # Prompt y
20     li $v0, 4
21     la $a0, prompt_y
22     syscall
23     li $v0, 5
24     syscall
25     move $t1, $v0      # y in $t1

```

Line: 56 Column: 1 Show Line Numbers

Mars Messages Run I/O

Go: execution completed successfully.
Assemble: assembling /Users/yunishabasnet/Documents/7th/computer architecture/2d.asm
Assemble: operation completed successfully.

⇒ The output:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu	\$2,\$0,0x00000004
	0x00400004	0x3c011001	lui	\$1,0x00001001
	0x00400008	0x34240000	ori	\$4,\$1,0x00000000
	0x0040000c	0x000000c	syscall	
	0x00400010	0x24020005	addiu	\$2,\$0,0x00000005
	0x00400014	0x000000c	syscall	
	0x00400018	0x00024021	addu	\$8,\$0,\$2
	0x00400022	0x24020004	addiu	\$2,\$0,0x00000004
	0x00400020	0x3c011001	lui	\$1,0x00001001

Label	Address	Labels
(global)	0x00400000	main
2d.asm	0x10010000	
prompt_x	0x10010014	
prompt_y	0x10010018	
result	0x10010028	
newline	0x10010044	

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	e	t	n	E	a	v	r	e
0x10010004	f	o	n	f	u	e	l	u
0x10010008	\	\	\	\	\	\	\	\
0x1001000c	=	\	\	\	\	\	\	\
0x10010010	\	\	\n	\	\	\	\	\
0x10010014	\	\	\	\	\	\	\	\
0x10010018	\	\	\	\	\	\	\	\
0x10010022	\	\	\	\	\	\	\	\
0x10010026	\	\	\	\	\	\	\	\
0x10010030	\	\	\	\	\	\	\	\
0x10010034	\	\	\	\	\	\	\	\
0x10010038	\	\	\	\	\	\	\	\
0x10010042	\	\	\	\	\	\	\	\
0x10010046	\	\	\	\	\	\	\	\
0x10010050	\	\	\	\	\	\	\	\
0x10010054	\	\	\	\	\	\	\	\
0x10010058	\	\	\	\	\	\	\	\
0x10010062	\	\	\	\	\	\	\	\
0x10010066	\	\	\	\	\	\	\	\
0x10010070	\	\	\	\	\	\	\	\
0x10010074	\	\	\	\	\	\	\	\
0x10010078	\	\	\	\	\	\	\	\
0x10010082	\	\	\	\	\	\	\	\
0x10010086	\	\	\	\	\	\	\	\
0x10010090	\	\	\	\	\	\	\	\
0x10010094	\	\	\	\	\	\	\	\
0x10010098	\	\	\	\	\	\	\	\
0x100100a2	\	\	\	\	\	\	\	\
0x100100a6	\	\	\	\	\	\	\	\
0x100100b0	\	\	\	\	\	\	\	\
0x100100b4	\	\	\	\	\	\	\	\
0x100100b8	\	\	\	\	\	\	\	\
0x100100bc	\	\	\	\	\	\	\	\
0x100100cc	\	\	\	\	\	\	\	\
0x100100dc	\	\	\	\	\	\	\	\
0x100100ec	\	\	\	\	\	\	\	\
0x100100fc	\	\	\	\	\	\	\	\
0x10010100	\	\	\	\	\	\	\	\

Mars Messages Run I/O

Enter value for x: 2
Enter value for y: 3
Result of ((4x) / 3) * y = 6
— program is finished running —

- Write a program (MIPS Assembly) to retrieve two numbers from a user and swap those numbers using only the *XOR* operation. You should not use a temporary

variable to store the numbers while swapping them. Your program should include a proper and useful prompt for input, and print the results meaningfully.

⇒ The answer:

⇒ The code:

```
.data
prompt_a:    .asciiz "Enter first number
(a): "
prompt_b:    .asciiz "Enter second
number (b): "
before:      .asciiz "\nBefore swapping:
a = "
and_b:       .asciiz ", b = "
after:       .asciiz "\nAfter swapping:
a = "
newline:     .asciiz "\n"

.text
.globl main

main:
    # Prompt for a
    li $v0, 4
    la $a0, prompt_a
    syscall

    li $v0, 5
    syscall
    move $t0, $v0        # a in $t0

    # Prompt for b
    li $v0, 4
    la $a0, prompt_b
    syscall

    li $v0, 5
    syscall
    move $t1, $v0        # b in $t1
```

```
##### Print BEFORE swap #####
li $v0, 4
la $a0, before
syscall

li $v0, 1
move $a0, $t0
syscall

li $v0, 4
la $a0, and_b
syscall

li $v0, 1
move $a0, $t1
syscall

##### XOR Swap #####
xor $t0, $t0, $t1    # a = a ^ b
xor $t1, $t0, $t1    # b = a ^ b →
original a
xor $t0, $t0, $t1    # a = a ^ b →
original b

##### Print AFTER swap #####
li $v0, 4
la $a0, after
syscall

li $v0, 1
move $a0, $t0
syscall

li $v0, 4
la $a0, and_b
syscall
```

```

    li $v0, 1
    move $a0, $t1
    syscall

    # Newline
    li $v0, 4
    la $a0, newline
    syscall

    # Exit
    li $v0, 10
    syscall

```

The screenshot shows the MARS 4.5 assembly debugger interface. The assembly code in the editor window is:

```

1 .data
2 prompt_a: .ascii "Enter first number (a): "
3 prompt_b: .ascii "Enter second number (b): "
4 after: .ascii "Before swapping: a = "
5 and_b: .ascii ", b = "
6 after: .ascii "\nAfter swapping: a = "
7 newline: .ascii "\n"
8
9 .text
10 .globl main
11 main:
12     # Prompt for a
13     li $v0, 4
14     la $a0, prompt_a
15     syscall
16
17     li $v0, 5
18     syscall
19     move $t0, $v0      # a in $t0
20
21     # Prompt for b
22     li $v0, 4
23     la $a0, prompt_b
24     syscall
25

```

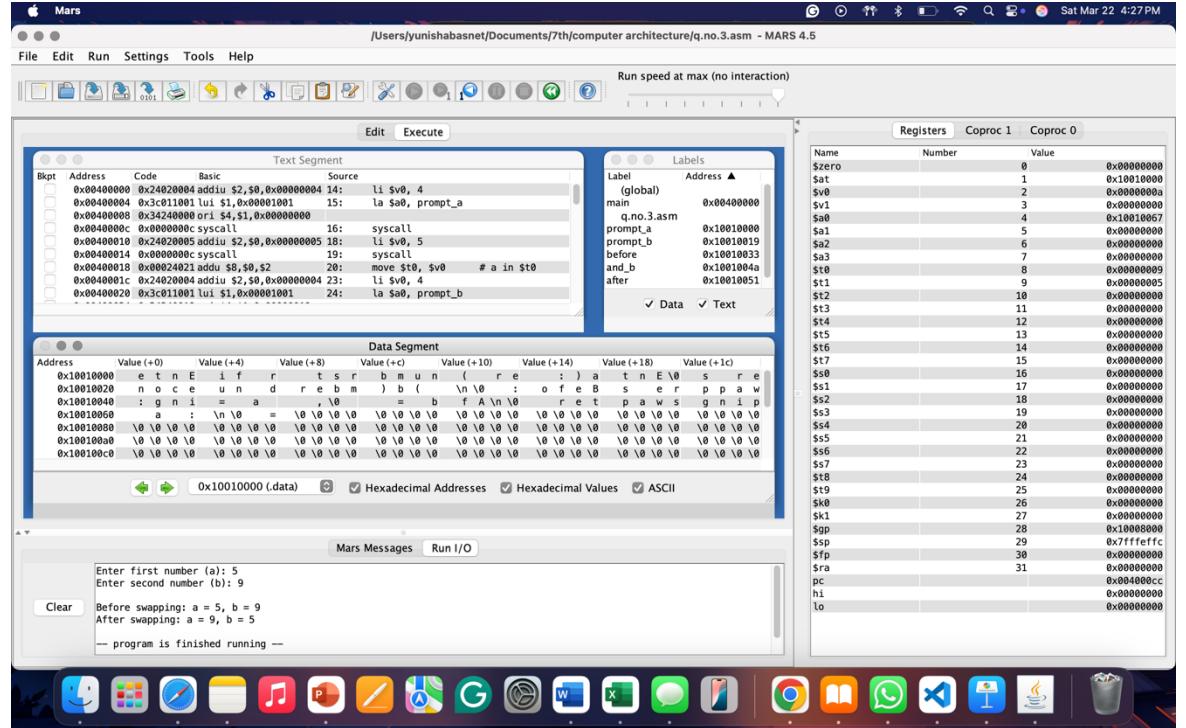
The Mars Messages window shows the interaction with the user:

- Enter first number (a): 5
- Enter second number (b): 9
- Before swapping: a = 5, b = 9
- After swapping: a = 9, b = 5
- program is finished running --

The Registers window displays the following register values:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000003
\$v1	3	0x00000000
\$a0	4	0x10000007
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000005
\$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	0x7fffffcfcf
\$sp	29	0x00000000
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x004008cc
hi		0x00000000
lo		0x00000000

⇒ The output:



4. Using only *sll* and *srl*, implement a program to check if a user input value is even or odd. The result should print out *0* if the number is even or *1* if the number is odd. Your program (MIPS Assembly) should include a proper and useful prompt for input, and print the results in a meaningful manner.

⇒ The answer:

⇒ The code:

```
.data
prompt:    .asciiz "Enter a number to
check even (0) or odd (1): "
result_msg: .asciiz "Result: "
newline:   .asciiz "\n"
```

```
.text
.globl main
```

```
main:
# Prompt user
```

```
    li $v0, 4
    la $a0, prompt
    syscall

    # Read integer input
    li $v0, 5
    syscall
    move $t0, $v0      # store number in
$t0

        # Isolate least significant bit
using only shifts
        sll $t1, $t0, 31      # move LSB to
MSB
        srl $t2, $t1, 31      # move it back
→ now $t2 = 0 or 1

        # Print result message
        li $v0, 4
        la $a0, result_msg
        syscall

        # Print 0 (even) or 1 (odd)
        li $v0, 1
        move $a0, $t2
        syscall

        # Newline
        li $v0, 4
        la $a0, newline
        syscall

        # Exit
        li $v0, 10
        syscall
```

The screenshot shows the Mars 4.5 assembly editor interface. The top menu bar includes File, Edit, Run, Settings, Tools, and Help. The title bar indicates the file path /Users/yunishabasnet/Documents/7th/computer architecture/4.asm - MARS 4.5. The main window has tabs for hw3.asm, q.no.2.asm, 2(B).asm, 2.casm, 2d.asm, q.no.3.asm, and 4.asm, with 4.asm selected. The code area contains assembly instructions for checking if a number is even or odd. The right panel displays the Registers window with columns for Name, Number, and Value, listing various寄存器 (Registers) from \$zero to \$lo with their corresponding values. The bottom left shows Mars Messages and Run I/O windows with assembly completion logs. The bottom dock contains various Mac OS X application icons.

```
1 .data
2 prompt: .ascii "Enter a number to check even (0) or odd (1): "
3 result_msg: .ascii "Result: "
4 newline: .ascii "\n"
5
6 .text
7 .globl main
8
9 main:
10    # Prompt user
11    li $v0, 4
12    la $a0, prompt
13    syscall
14
15    # Read integer input
16    li $v0, 5
17    syscall
18    move $t0, $v0    # store number in $t0
19
20    # Isolate least significant bit using only shifts
21    sll $t1, $t0, 31  # move LSB to MSB
22    srl $t2, $t1, 31  # move it back - now $t2 = 0 or 1
23
24    # Print result message
25    li $v0, 4
```

Line: 21 Column: 10 Show Line Numbers

Mars Messages

Go: execution completed successfully.

Assemble: assembling /Users/yunishabasnet/Documents/7th/computer architecture/4.asm

Assemble: operation completed successfully.

Run I/O

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	0x10000000
\$sp	29	0x7ffffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00000000
hi		0x00000000
lo		0x00000000

⇒ The output:

5. Implement a program (MIPS Assembly) to prompt the user for two numbers, the first being any number and the second a prime number. Return to the user a 0 if the second number is a prime factor for

the first one, otherwise any number if it is not. For example, if the user enters 60 and 5, the program returns 0. If the user enters 62 and 5, the program returns 2.

⇒ The answer:

⇒ The code:

```
.data
prompt_a:    .asciiz "Enter the main
number: "
prompt_b:    .asciiz "Enter a prime
number: "
result:      .asciiz "Result: "
newline:     .asciiz "\n"

.text
.globl main

main:
# Prompt for first number
li $v0, 4
la $a0, prompt_a
syscall

li $v0, 5
syscall
move $t0, $v0      # $t0 = number

# Prompt for prime number
li $v0, 4
la $a0, prompt_b
syscall

li $v0, 5
syscall
move $t1, $v0      # $t1 = supposed
prime

# Divide number by prime
```

```

    div $t0, $t1
    mfhi $t2          # remainder in
$ t2

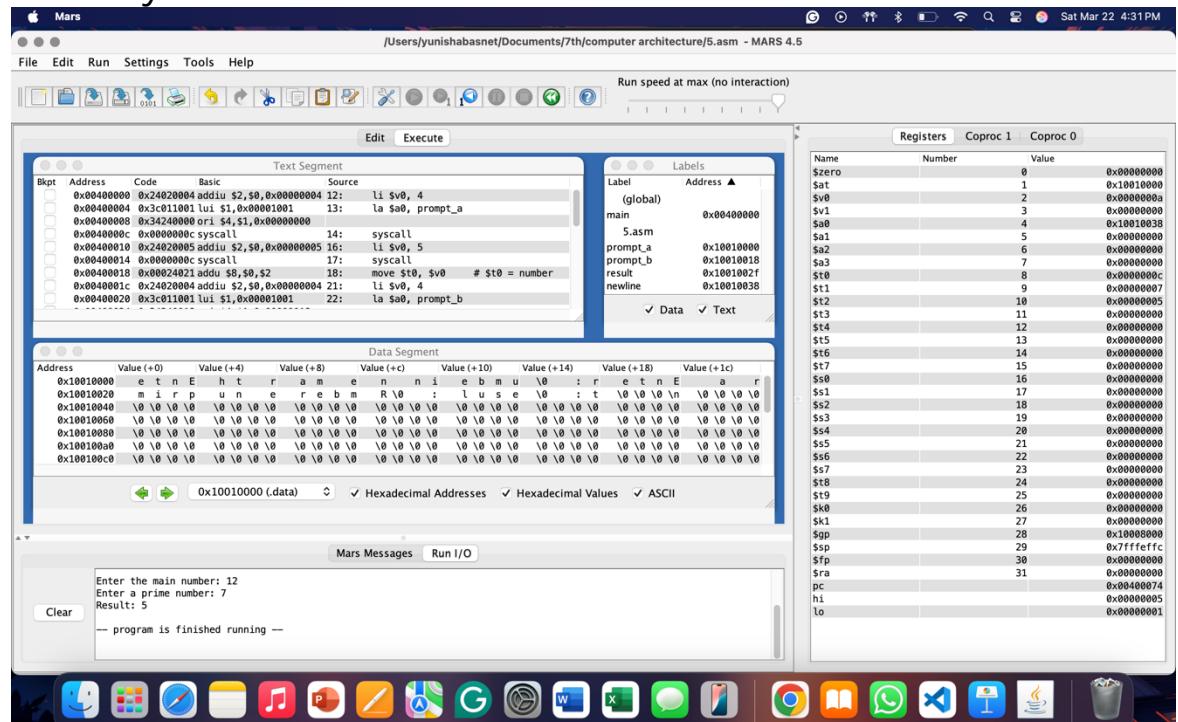
# Print result
li $v0, 4
la $a0, result
syscall

li $v0, 1
move $a0, $t2      # if remainder is
0 → it's a factor
syscall

# Newline
li $v0, 4
la $a0, newline
syscall

# Exit
li $v0, 10
syscall

```



⇒ The output:

Mars

/Users/yunishabasnet/Documents/7th/computer architecture/5.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

hw3.asm q.no.2.asm 2(B).asm 2c.asm 2d.asm q.no.3.asm 4.asm 5.asm

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$t1	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
\$t8	16	0x00000000
\$t9	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	0x10000000
\$sp	29	0x7fffffefff
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Line: 50 Column: 1 Show Line Numbers

Mars Messages Run I/O

Go: execution completed successfully.

Assemble: assembling /Users/yunishabasnet/Documents/7th/computer architecture/5.asm

Assemble: operation completed successfully.

Clear