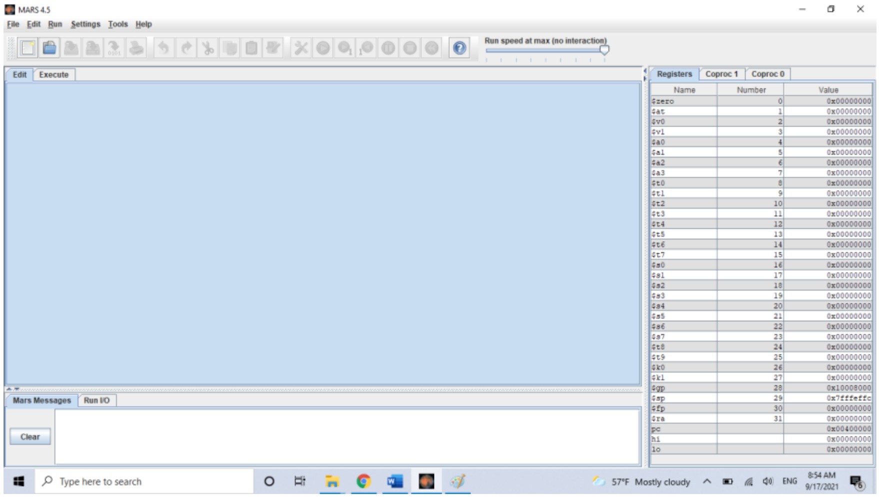
MIPS Assembly Language Programming

Engineering School

San Francisco Bay University

Fremont, California, 2023

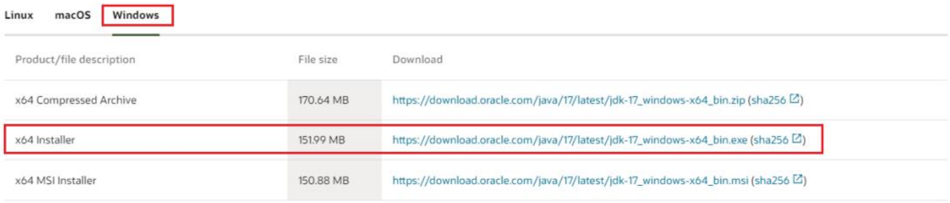
1. Download Mars at http://courses.missouristate.edu/kenvollmar/mars/download.htm
2. Click Mars4_5.jar to run. You will get it as follows



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

If can't, install java kit at

https://www.oracle.com/java/technologies/downloads/#jdk17-windows



Product/file description	File size	Download
x64 Compressed Archive	170.64 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256)
x64 Installer	151.99 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256)
x64 MSI Installer	150.88 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256)

3. Examples of MIPS Assemble programs

```
#-------------------------------------- Ex-1 ------------------------------------------------------------------------------------------

# Program File: Hello.asm

# Purpose: First program, Hello World

.data # Define the program data.

        greeting: .asciiz "Hello World" #The string to print.

.text # Define the program instructions.

        main: # Label to define the main program.

                li $v0,4                # Load 4 into $v0 to indicate a print string.

                la $a0, greeting # Load the address of the greeting into $a0.

                syscall                 # Print greeting. The print is indicated by

                                                # $v0 having a value of 4, and the string to

                                                # print is stored at the address in $a0.

                li $v0, 10                      # Load a 10 (halt) into $v0.

                syscall                 # The program ends; like return 0 in C language
```

**Note: SYSCALL system services*

https://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

```
#--------------------------------------  Ex-2  -----------------------------------------------------------------------------------------

# Program File: Read_String.asm

# Program to read a string from a user, and

# print that string back to the console.

.data

        input: .space 81

        inputSize: .word 80

        prompt: .asciiz "Please enter an string: "

        output: .asciiz "\nYou typed the string: "
```

```
.text
        main:
                # Prompt for the string to enter
                li $v0, 4
                la $a0, prompt
                syscall

                # Read the string.
                li $v0, 8
                la $a0, input
                lw $a1, inputSize
                syscall

                # Output the text
                li $v0, 4
                la $a0, output
                syscall

                # Output the number
                li $v0, 4
                la $a0, input
                syscall

                # Exit the program
                li $v0, 10
                syscall
#---------------------------------------------- Ex-3  ------------------------------------------------------------------------------------
# File: Addition.asm
# Purpose: To illustrate some addition operators
```

```
.data

.text
        # illustrate R format add operator
        li $t1, 100
        li $t2, 50
        add $t0, $t1, $t2

        # illustrate add with an immediate. Note that
        # an add with a pseudo instruction translated
        # into an addi instruction
        addi $t0, $t0, 50
        add $t0, $t0, 50

        # using an unsign number. Note that the
        # result is not what is expected
        # for negative numbers.
        addiu $t0, $t2, -100

        # addition using a 32 immediate. Note that 5647123
        # base 10 is 0x562b13
        addi $t1, $t2, 5647123

#---------------------------------------------- Ex-4  ----------------------------------------------------------------------------------
# File name: Psudo_Code.asm
# Purpose: To illustrate how to translate a pseudo code
#          program into assembly
#
# Pseudo Code
# main{
        # register int i = input("Please enter the first value to add: ");
        # register int j = input("Please enter the second value to add: ");
        # register int k = i + j;
        # print("The result is " + k);
# }
.data
        prompt1: .asciiz "Please enter the first value to add: "
        prompt2: .asciiz "Please enter the second value to add: "
        result: .asciiz "The result is "

.text
.globl main
# global main basically means that the symbol should be visible to the linker
# because other object files will use it.
# Without it, the symbol main is considered local to the object file it's assembled to,
# and will not appear after the assembly file is assembled.

main:
        # Register conventions
        # i is $s0
        # j is $s1
        # k is $s2
        # register int i =
        # input("Please enter the first value to add: ");
        addi $v0, $zero, 4
        la $a0, prompt1
        syscall
        addi $v0, $zero, 5
```

```
	syscall
	move $s0, $v0

	# register int j =
	# input("Please enter the second value to add: ");
	addi $v0, $zero, 4
	la $a0, prompt2
	syscall
	addi $v0, $zero, 5
	syscall
	move $s1, $v0

	# register int k = i + j;
	add $s2, $s1, $s0

	# print("The result is " + k);
	addi $v0, $zero, 4
	la $a0, result
	syscall
	addi $v0, $zero, 1
	move $a0, $s2
	syscall

	#End the program
	addi $v0, $zero, 10
	syscall
#---------------------------------------------- Ex-5  ------------------------------------------------------------------------------------
# File: Even_Odd.asm
# Purpose: To have a user enter a number, and print 0 if
#               the number is even, 1 if the number is odd
.data
        prompt: .asciiz "Enter your number: "
        result: .asciiz "A result of 0 is even, 1 is odd: result = "
.text
.globl main
main:
        # Get input value
        addi $v0, $zero, 4 # Write Prompt
        la $a0, prompt
        syscall
        addi $v0, $zero, 5 # Retrieve input
        syscall
        move $s0, $v0

        # Check if odd or even
        addi $t0, $zero, 2 # Store 2 in $t0
        div $t0, $s0, $t0 # Divide input by 2
        mfhi $s1 # Save remainder in $s1

        # Print output
        addi $v0, $zero, 4 # Print result string
        la $a0, result
        syscall
        addi $v0, $zero, 1 # Print result
        move $a0, $s1
        syscall

        #Exit program
```

```
            addi $v0, $zero, 10
            syscall
#---------------------------------------------------- Ex-6  ----------------------------------------------------------------------------
# File: Arith_Expr.asm

# Purpose: To calculate the result of 5*x^ + 2*x + 3
.data
            prompt: .asciiz "Enter a value for x: "
            result: .asciiz "The result is: "


.text
.globl main
main:
            # Get input value, x
            addi $v0, $zero, 4
            la $a0, prompt
            syscall
            addi $v0, $zero, 5
            syscall
            move $s0, $v0

            # Calculate the result of 5*x*x + 2* x + 3 and store it in $s1.
            mul $t0, $s0, $s0
            mul $t0, $t0, 5
            mul $t1, $s0, 2
            add $t0, $t0, $t1
            addi $s1, $t0, 3

            # Print output
```

```
        addi $v0, $zero, 4 # Print result string
        la $a0, result
        syscall
        addi $v0, $zero, 1 # Print result
        move $a0, $s1
        syscall

        #Exit program
        addi $v0, $zero, 10
        syscall

#-----------------------------------------------  Ex-7  -------------------------------------------------------------------------------
# File: Division.asm
# Purpose: To show the difference in result if
#            ordering of multiplication and division
#            are reversed.
.data
        result1: .asciiz "\n(10/3)*3 = "
        result2: .asciiz "\n(10*3)/3 = "

.text
.globl main
main:
        addi $s0, $zero, 10 # Store 10 and 3 in registers $s0 and $s1
        addi $s1, $zero, 3

        div $s2, $s0, $s1 # Write out (10/3) * 3
        mul $s2, $s2, $s1
        addi $v0, $zero, 4
        la $a0, result1
        syscall
        addi $v0, $zero, 1
        move $a0, $s2
        syscall

        mul $s2, $s0, $s1 # Write out (10*3)/3
        div $s2, $s2, $s1
        addi $v0, $zero, 4
        la $a0, result2
        syscall
        addi $v0, $zero, 1
        move $a0, $s2
        syscall

        addi $v0, $zero, 10 #Exit program
        syscall
#--------------------------------------------- Ex-8 -------------------------------------------------------------------------------
# File: XOR.asm
#Purpose: To show the XOR operation is reversible

.data
        output1: .asciiz "\nAfter first xor: "
        output2: .asciiz "\nAfter second xor: "
.text
.globl main
main:
        ori $s0, $zero, 0x01234567 # the hex numbers
```

```
    # Write out the XOR'ed value
    la $a0, output1
    li $v0, 4
    syscall
    xori $s0, $s0, 0xffffffff # the results in $t1 will be fedcba98
    move $a0, $s0
    li $v0, 34
    syscall

    # Show the original value has been restored.
    la $a0, output2
    li $v0, 4
    syscall
    xori $s0, $s0, 0xffffffff # the results in $t1 will be fedcba98
    move $a0, $s0
    li $v0, 34
    syscall

    ori $v0, $zero, 10 # Exit program
    syscall

#---------------------------------------------  Ex-9  -------------------------------------------------------------------------------
# File: Shift.asm
# Purpose: To illustrate various shift operations.

.data
    result1: .asciiz "\nshift left logical 4 by 2 bits is "
    result2: .asciiz "\nshift right logical 16 by 2 bits is "
    result3: .asciiz "\nshift right arithmetic 34 by 2 bits is "
    result4: .asciiz "\nshift right arithmetic -34 by 2 bits is "
    result5: .asciiz "\nrotate right 0xffffffe1 by 2 bits is "
    result6: .asciiz "\nrotate left 0xffffffe1 by 2 bits is "

.text
.globl main
main:
    #SLL example
    addi $t0, $zero, 4
    sll $s0, $t0, 2
    addi $v0, $zero, 4
    la $a0, result1
    syscall
    addi $v0, $zero, 1
    move $a0, $s0
    syscall

    #SRL example
    addi $t0, $zero, 16
    srl $s0, $t0, 2
    addi $v0, $zero, 4
    la $a0, result2
    syscall
    addi $v0, $zero, 1
    move $a0, $s0
    syscall

    #SRA example
    addi $t0, $zero, 34
```

```
sra $s0, $t0, 2
addi $v0, $zero, 4
la $a0, result3
syscall
addi $v0, $zero, 1
move $a0, $s0
syscall

#SRA example
addi $t0, $zero, -34
sra $s0, $t0, 2 # sra 2 bits, which is division by 4
addi $v0, $zero, 4 # Output the result
la $a0, result4
syscall
addi $v0, $zero, 1
move $a0, $s0
syscall

#rol example
ori $t0, $zero, 0xffffffe1
ror $s0, $t0, 2
li $v0, 4
la $a0, result6
syscall
li $v0, 34
move $a0, $s0
syscall

#rol example
```

```
        ori $t0, $zero, 0xffffffe1
        rol $s0, $t0, 2
        li $v0, 4
        la $a0, result6
        syscall
        li $v0, 34
        move $a0, $s0
        syscall

        addi $v0, $zero, 10 # Exit program
        syscall

#---------------------------------------------  Ex-10  ----------------------------------------------------------------------------
# File: Function.asm
# Purpose: To illustrate implementing and calling a
#         subprogram named Exit.
.data
        prompt: .asciiz "Please enter an integer: "
        result: .asciiz "\nYou entered: "

.text
main:
        # read an input value from the user
        li $v0, 4
        la $a0, prompt
        syscall
        li $v0, 5
        syscall
        move $s0, $v0
```

```
	# print the value back to the user
	li $v0, 4
	la $a0, result
	syscall
	li $v0, 1
	move $a0, $s0
	syscall

	# call the Exit subprogram to exit
	jal Exit

# subprogram: Exit
# purpose: to use syscall service 10 to exit a program
# input: None
# output: None
# side effects: The program is exited
Exit:
	li $v0, 10
	syscall

#---------------------------------------------  Ex-11  ------------------------------------------------------------------------------
# File: PrintNewLine_Func.asm
# Purpose: To illustrate implementing and calling a
#			subprogram named PrintNewLine.
.data
	prompt: .asciiz "Please enter an integer: "
	result: .asciiz "You entered: "
	__PNL_newline: .asciiz "\n"
```

```
.text
main:
        # read an input value from the user
        li $v0, 4
        la $a0, prompt
        syscall
        li $v0, 5
        syscall
        move $s0, $v0

        # print the value back to the user
        jal PrintNewLine
        li $v0, 4
        la $a0, result
        syscall
        li $v0, 1
        move $a0, $s0
        syscall

        # call the Exit subprogram to exit
        jal Exit

        # subprogram: PrintNewLine
        # purpose: to output a new line to the user console
        # input: None
        # output: None
        # side effects: A new line character is printed to the
        # user's console
```

```
PrintNewLine:

        li $v0, 4

        la $a0, __PNL_newline

        syscall

        jr $ra



# subprogram: Exit

# purpose: to use syscall service 10 to exit a program

# input: None

# output: None

# side effects: The program is exited


Exit:

        li $v0, 10

        syscall


#----------------------------------------------  Ex-12  ------------------------------------------------------------------------------

# File: PrintInt.asm

# Purpose: To illustrate implementing and calling a

#       subprogram named PrintNewLine.

.data

        prompt: .asciiz "Please enter an integer: "

        result: .asciiz "You entered: "


        __PNL_newline: .asciiz "\n"


.text
```

```
main:
        # read an input value from the user
        la $a0, prompt
        jal PrintString
        li $v0, 5
        syscall
        move $s0, $v0

        # print the value back to the user
        jal PrintNewLine
        la $a0, result
        move $a1, $s0
        jal PrintInt

        # call the Exit subprogram to exit
        jal Exit

# subprogram: PrintNewLine
# purpose: to output a new line to the user console
# input: None
# output: Nones
# side effects: A new line character is printed to the
# user's console

PrintNewLine:
        li $v0, 4
        la $a0, __PNL_newline
        syscall
        jr $ra
```

```
# subprogram: PrintInt
# purpose: To print a string to the console
# input:   $a0 - The address of the string to print.
#                    $a1 - The value of the int to print
# returns: None
# side effects: The String is printed followed by the integer value.

PrintInt:
        # Print string. The string address is already in $a0
        li $v0, 4
        syscall

        # Print integer. The integer value is in $a1, and must
        # be first moved to $a0.
        move $a0, $a1
        li $v0, 1
        syscall

        #return
        jr $ra

# subprogram: PrintString
# purpose: To print a string to the console
# input:   $a0 - The address of the string to print.
# returns: None
# side effects: The String is printed to the console.

PrintString:
        addi $v0, $zero, 4
        syscall
        jr $ra
# subprogram: Exit
# purpose: to use syscall service 10 to exit a program
# input: None
# output: None
# side effects: The program is exited

Exit:
        li $v0, 10
        syscall

#---------------------------------------------- Ex-13 ---------------------------------------------------------------------------
# File: PromptInt.asm
# Purpose: To illustrate implementing and calling a
#        subprogram named PrintNewLine.

.data
        prompt: .asciiz "Please enter an integer: "
        result: .asciiz "You entered: "

        __PNL_newline: .asciiz "\n"



.text
main:
        # read an input value from the user
        la $a0, prompt
        jal PromptInt
        move $s0, $v0

        # print the value back to the user
        jal PrintNewLine
        la $a0, result
        move $a1, $s0
        jal PrintInt

        # call the Exit subprogram to exit
        jal Exit

# subprogram: PrintNewLine
# purpose: to output a new line to the user console
# input: None
# output: None
# side effects: A new line character is printed to the
# user's console

PrintNewLine:
        li $v0, 4
        la $a0, __PNL_newline
        syscall
        jr $ra

# subprogram: PrintInt
# purpose: To print a string to the console
# input:   $a0 - The address of the string to print.
#         $a1 - The value of the int to print
# returns: None
# side effects: The String is printed followed by the integer value.

PrintInt:
        # Print string. The string address is already in $a0
        li $v0, 4
        syscall

        # Print integer. The integer value is in $a1, and must
        # be first moved to $a0.
        move $a0, $a1
        li $v0, 1
        syscall

        #return
        jr $ra

# subprogram: PromptInt
# purpose: To print the user for an integer input, and
#       to return that input value to the caller.
# input: $a0 - The address of the string to print.
# returns: $v0 - The value the user entered
# side effects: The String is printed followed by the integer value.

PromptInt:
        # Print the prompt, which is already in $a0
        li $v0, 4
        syscall
        # Read the integer value. Note that at the end of the
        # syscall the value is already in $v0, so there is no
        # need to move it anywhere.
        move $a0, $a1
        li $v0, 5
        syscall

        #return
        jr $ra
# subprogram: PrintString
# purpose: To print a string to the console
# input: $a0 - The address of the string to print.
# returns: None
# side effects: The String is printed to the console.

PrintString:
        addi $v0, $zero, 4
        syscall
        jr $ra

# subprogram: Exit
# purpose: to use syscall service 10 to exit a program
# input: None
# output: None
# side effects: The program is exited

Exit:
        li $v0, 10
        syscall
```

```
#---------------------------------------------- Ex-14 --------------------------------------------------------------------------

# File: utils.asm

# Purpose: To define utilities which will be used in MIPS programs.

#

# Subprograms Index:

# Exit                          - Call syscall with a server 10 to exit the program

# NewLine                       - Print a new line character (\n) to the console

# PrintInt                      - Print a string with an integer to the console

# PrintString       - Print a string to the console

# PromptInt         - Prompt the user to enter an integer, and return

#                                                   it to the calling program.

#

# subprogram: PrintNewLine

# purpose: to output a new line to the user console

# input: None

# output: None

# side effects: A new line character is printed to the

#                                       user's console


.data

        __PNL_newline: .asciiz "\n"


.text

PrintNewLine:

        li $v0, 4

        la $a0, __PNL_newline

        syscall

        jr $ra
```

```
# subprogram: PrintInt
# purpose: To print a string to the console
# input: $a0 - The address of the string to print.
# $a1 - The value of the int to print
# returns: None
# side effects: The String is printed followed by the integer value.


PrintInt:
        # Print string. The string address is already in $a0
        li $v0, 4
        syscall

        # Print integer. The integer value is in $a1, and must
        # be first moved to $a0.
        move $a0, $a1
        li $v0, 1
        syscall

        #return
        jr $ra

# subprogram: PromptInt
# purpose: To print the user for an integer input, and
#                     to return that input value to the caller.
# input: $a0 - The address of the string to print.
# returns: $v0 - The value the user entered
# side effects: The String is printed followed by the integer value.
```

```
PromptInt:
        # Print the prompt, which is already in $a0
        li $v0, 4
        syscall

        # Read the integer value. Note that at the end of the
        # syscall the value is already in $v0, so there is no
        # need to move it anywhere.
        move $a0, $a1
        li $v0, 5
        syscall

        #return
        jr $ra
# subprogram: PrintString
# purpose: To print a string to the console
# input: $a0 - The address of the string to print.
# returns: None
# side effects: The String is printed to the console.

PrintString:
        addi $v0, $zero, 4
        syscall
        jr $ra
# subprogram: Exit
# purpose: to use syscall service 10 to exit a program
# input: None
# output: None
# side effects: The program is exited

Exit:
        li $v0, 10
        syscall



# File: Include.asm
# Purpose: To illustrate implementing and calling a
#           subprogram named PrintNewLine.

.data
        prompt: .asciiz "Please enter an integer: "
        result: .asciiz "You entered: "

.text
main:
        # read an input value from the user
        la $a0, prompt
        jal PromptInt
        move $s0, $v0

        # print the value back to the user
        jal PrintNewLine
        la $a0, result
        move $a1, $s0
        jal PrintInt

        # call the Exit subprogram to exit
        jal Exit

.include "utils.asm"



#---------------------------------------------  Ex-15  ----------------------------------------------------------------------------
//Pseudo code
main{
        static volatile int a = 5;
        static volatile int b = 2;
        static volatile int c = 3;
        int x = prompt("Enter a value for x: ");
        int y = a * x * x + b * x + c;
        print("The result is: " + y);
}

# File: Mem_Var.asm
.data
        a: .word 5
        b: .word 2
        c: .word 3
        y: .word 0
        prompt: .asciiz "Enter a value for x: "
        result: .asciiz "The result is: "

.text
.globl main

main:
```

```
# Get input value and store it in $s0
la $a0, prompt
jal PromptInt
move $s0, $v0

# Load constants a, b, and c into registers
lw $t5, a
lw $t6, b
lw $t7, c

# Calculate the result of y=a*x*x + b * x + c and store it.
mul $t0, $s0, $s0
mul $t0, $t0, $t5
mul $t1, $s0, $t6
add $t0, $t0, $t1
add $s1, $t0, $t7

# Store the result from $s1 to y.
sw $s1, y

# Print output from memory y
la $a0, result
lw $a1, y
jal PrintInt
jal PrintNewLine

#Exit program
jal Exit
```

```
.include "utils.asm"

#-----------------------------------------------  Ex-16  ------------------------------------------------------------------------------
#File: Reg_Dir_Access.asm

.data
	y: .word 0
	prompt: .asciiz "Enter a value for x: "
	result: .asciiz "The result is: "

.text
.globl main

main:
	# Get input value and store it in $s0
	la $a0, prompt
	jal PromptInt
	move $s0, $v0

	# Load constants a, b, and c into registers
	li $t5, 5
	li $t6, 2
	li $t7, 3

	# Calculate the result of y=a*x*x + b * x + c and store it.
	mul $t0, $s0, $s0
	mul $t0, $t0, $t5
	mul $t1, $s0, $t6
	add $t0, $t0, $t1
```

```
        add $s1, $t0, $t7

        # Print output from memory y
        la $a0, result
        move $a1, $s1
        jal PrintInt
        jal PrintNewLine

        #Exit program
        jal Exit

.include "utils.asm"

#---------------------------------------------  Ex-17  -------------------------------------------------------------------------
#File: Reg_Indir_Access.asm
.data
        .word 5
        .word 2
        .word 3
        y: .word 0
        prompt: .asciiz "Enter a value for x: "
        result: .asciiz "The result is: "

.text
.globl main
main:
        # Get input value and store it in $s0
        la $a0, prompt
        jal PromptInt
```

```
	move $s0, $v0

	# Load constants a, b, and c into registers
	lui $t0, 0x1001
	lw $t5, 0($t0)
	addi $t0, $t0, 4
	lw $t6, 0($t0)
	addi $t0, $t0, 4
	lw $t7, 0($t0)

	# Calculate the result of y=a*x*x + b * x + c and store it.
	mul $t0, $s0, $s0
	mul $t0, $t0, $t5
	mul $t1, $s0, $t6
	add $t0, $t0, $t1
	add $s1, $t0, $t7

	# Print output from memory y
	la $a0, result
	move $a1, $s1
	jal PrintInt
	jal PrintNewLine

	#Exit program
	jal Exit

.include "utils.asm"

#---------------------------------------------  Ex-18  ---------------------------------------------------------------------------
```

```
#File: Reg_offset_Access_0.asm

.data
.word 5
.word 2
.word 3
y: .word 0
prompt: .asciiz "Enter a value for x: "
result: .asciiz "The result is: "

.text
.globl main
main:
        # Get input value and store it in $s0
        la $a0, prompt
        jal PromptInt
        move $s0, $v0

        # Load constants a, b, and c into registers
        lui $t0, 0x1001
        lw $t5, 0($t0)
        lw $t6, 4($t0)
        lw $t7, 8($t0)

        # Calculate the result of y=a*x*x + b * x + c and store it.
        mul $t0, $s0, $s0
        mul $t0, $t0, $t5
        mul $t1, $s0, $t6
        add $t0, $t0, $t1
        add $s1, $t0, $t7

        # Print output from memory y
        la $a0, result
        move $a1, $s1
        jal PrintInt
        jal PrintNewLine

        #Exit program
        jal Exit

.include "utils.asm"

#---------------------------------------------------
#File: #File: Reg_offset_Access_1.asm

.data
        .word constants
        y: .word 0
        prompt: .asciiz "Enter a value for x: "
        result: .asciiz "The result is: "
        constants:
        .word 5
        .word 2
        .word 3

.text
.globl main
main:
        # Get input value and store it in $s0
        la $a0, prompt
        jal PromptInt
        move $s0, $v0

        # Load constants a, b, and c into registers
        lui $t0, 0x1001
        lw $t0, 0($t0)
        lw $t5, 0($t0)
        lw $t6, 4($t0)
        lw $t7, 8($t0)

        # Calculate the result of y=a*x*x + b * x + c and store it.
        mul $t0, $s0, $s0
        mul $t0, $t0, $t5
        mul $t1, $s0, $t6
        add $t0, $t0, $t1
        add $s1, $t0, $t7

        # Print output from memory y
        la $a0, result
        move $a1, $s1
        jal PrintInt
        jal PrintNewLine

        #Exit program
        jal Exit

.include "utils.asm"
```

```
#----------------------------------------------  Ex-19  ----------------------------------------------------------------------------
#File: Simple_if.asm

.data
	num: .word 5
	PositiveNumber: .asciiz "Number is positive"

.text
	# if (num > 0 )
	lw $t0, num
	sgt $t1, $t0, $zero # $t1 is the boolean (num > 0)
	beqz $t1, end_if		# note: the code block is entered if

	# if logical is true, skipped if false.
	# {
	#	print ("Number is positive")
	la $a0, PositiveNumber
	jal PrintString
	# }

end_if:
	jal Exit

.include "utils.asm"

#----------------------------------------------  Ex-20  ----------------------------------------------------------------------------
#File: if_else.asm
```

```
.data
        num: .word -5
        PositiveNumber: .asciiz "Number is positive"
        NegativeNumber: .asciiz "Number is negative"

.text
        lw $t0, num
        sgt $t1, $t0, $zero
        beqz $t1, else

        #if block
        la $a0, PositiveNumber
        jal PrintString
        b end_if

        #else block
else:
        la $a0, NegativeNumber
        jal PrintString

end_if:
        jal Exit

.include "utils.asm"

#---------------------------------------------  Ex-21  -------------------------------------------------------------------------
#File: if_elseif_else.asm

.data
```

```
        num: .word 70
        InvalidInput: .asciiz "Number must be > 0 and < 100"
        OutputA: .asciiz "Grade is A"
        OutputB: .asciiz "Grade is B"
        OutputC: .asciiz "Grade is C"
        OutputD: .asciiz "Grade is D"
        OutputF: .asciiz "Grade is F"

.text
        #if block
        lw $s0, num
        slti $t1, $s0, 0
        sgt $t2, $s0, 100
        or $t1, $t1, $t2
        beqz $t1, grade_A

        #invalid input block
        la $a0, InvalidInput
        jal PrintString
        b end_if

grade_A:
        sge $t1, $s0, 90
        beqz $t1, grade_B
        la $a0, OutputA
        jal PrintString
        b end_if

grade_B:
        sge $t1, $s0, 80
        beqz $t1, grade_C
        la $a0, OutputB
        jal PrintString
        b end_if

grade_C:
        sge $t1, $s0, 70
        beqz $t1, grade_D
        la $a0, OutputC
        jal PrintString
        b end_if

grade_D:
        sge $t1, $s0, 60
        beqz $t1, else
        la $a0, OutputD
        jal PrintString
        b end_if

else:
        la $a0, OutputF
        jal PrintString
        b end_if

end_if:
        jal Exit

.include "utils.asm"
```

```
#---------------------------------------------- Ex-22 ----------------------------------------------------------------------------
#File: for_loop.asm


#n = prompt("enter the value to calculate the sum up to: ")
#total = 0; # Initial the total variable for sum
#for (i = 0; i < n; i++){
#           total = total + i
#}
#print("Total = " + total);


.data
            prompt: .asciiz "enter the value to calculate the sum up to: "
            output: .asciiz "The final result is: "


.text
            la $a0, prompt
            jal PromptInt
            move $s1, $v0
            li $s0, 0
            li $s2, 0 # Initialize the total


start_loop:
            sle $t1, $s0, $s1
            beqz $t1, end_loop


            # code block
            add $s2, $s2, $s0


            addi $s0, $s0, 1
```

```
        b start_loop

end_loop:
        la $a0, output
        move $a1, $s2
        jal PrintInt

        jal Exit

.include "utils.asm"

#---------------------------------------------  Ex-23  ---------------------------------------------------------------------------
#File: Nested_Blk.asm

#int n = prompt("Enter a value for the summation n, -1 to stop");
#while (n != -1){
#       if (n < -1){
#               print("Negative input is invalid");
#       }
#       else{
#               int total = 0
#               for (int i = 0; i < n; i++){
#                       total = total + i;
#               }
#               print("The summation is " + total);
#       }
#}

.data
```

```
    prompt: .asciiz "\nEnter an integer, -1 to stop: "
    error: .asciiz "\nValues for n must be > 0"
    output: .asciiz "\nThe total is: "


.text
    # Sentinel Control Loop
    la $a0, prompt
    jal PromptInt
    move $s0, $v0

    start_outer_loop:
        sne $t1, $s0, -1
        beqz $t1, end_outer_loop

        # If test for valid input
        slti $t1, $s0, -1
        beqz $t1, else
        la $a0, error
        jal PrintString
            b end_if

        else:
            # summation loop
            li $s1, 0
            li $s2, 0 # initialize total

            start_inner_loop:
                sle $t1, $s1, $s0
                beqz $t1, end_inner_loop
                add $s2, $s2, $s1
                addi $s1, $s1, 1
                b start_inner_loop

            end_inner_loop:
                la $a0, output
                move $a1, $s2
                jal PrintInt
        end_if:
        la $a0, prompt
        jal PromptInt
        move $s0, $v0
        b start_outer_loop

    end_outer_loop:
        jal Exit

.include "utils.asm"

#---------------------------------------------  Ex-24  ---------------------------------------------------------------------------
# Filename: AverageGrade.asm
# Purpose: Illustration of program to calculate a student grade
# Pseudo Code
#global main()
#{
# // The following variables are to be stored in data segment, and
# // not simply used from a register. They must be read each time
# // they are used, and saved when they are changed.
# static volatile int numberOfEntries = 0
# static volatile int total = 0
#
# // The following variable can be kept in a save register.
# register int inputGrade # input grade from the user
# register int average
#
# // Sentinel loop to get grades, calculate total.
# inputGrade = prompt("Enter grade, or -1 when done")
# while (inputGrade != -1)
# {
# numberOfEntries = numberOfEntries + 1
# total = total + inputGrade
# inputGrade = prompt("Enter grade, or -1 when done")
# }
#
# # Calculate average
# average = total / numberOfEntries
#
# // Print average
# print("Average = " + average)
#
# //Print grade if average is between 0 and 100, otherwise an error
# if ((grade >= 0) & (grade <= 100))
# {
# if (grade >= 90)
# {
# print("Grade is A")
# }
# if (grade >= 80)
# {
# print("Grade is B")
# }
# if (grade >= 70)
# {
# print("Grade is C")
# }
# else
# {
# print("Grade is F")
# }
# }
# else
# {
# print("The average is invalid")
# }
#}
.data
        numberOfEntries: .word 0
        total: .word 0
        average: .word
        prompt: .asciiz "Enter grade, or -1 when done: "
        avgOutput: .asciiz "The average is "
        gradeA: .asciiz "The grades is an A"
        gradeB: .asciiz "The grade is a B"
        gradeC: .asciiz "The grade is a C"
        gradeF: .asciiz "The grade is a F"
        invalidAvg: .asciiz "The average is invalid"
```

```
.text
.globl main
main:
        # Register Conventions:
        # $s0 - current inputGrade
        # $s1 - average
        la $a0, prompt
        jal PromptInt
        move $s0, $v0

        BeginInputLoop:
                addi $t0, $zero, -1 # set condition $s0 != -1
                seq $t0, $t0, $s0
                xor $t0, $t0, 0x00000001
                beqz $t0, EndInputLoop # check condition to end loop

                la $t0, numberOfEntries # increment # of entries
                lw $t1, 0($t0)
                addi $t1, $t1, 1
                sw $t1, 0($t0)

                la $t0, total # accumulate total
                lw $t1, 0($t0)
                add $t1, $t1, $s0
                sw $t1, 0($t0)

                la $a0, prompt # prompt for next input
                jal PromptInt
                move $s0, $v0
                b BeginInputLoop
        EndInputLoop:
        la $t0, numberOfEntries #Calculate Average
        lw $t1, 0($t0)
        la $t0, total
        lw $t2, 0($t0)
        div $s1, $t2, $t1

        la $a0, avgOutput # Print the average
        move $a1, $s1
        jal PrintInt
        jal PrintNewLine

        sge $t0, $s1, 0   # Set the condition
                                               #(average > 0) & (average < 100)
        addi $t1, $zero, 100
        sle $t1, $s1, $t1
        and $t0, $t0, $t1
        beqz $t0, AverageError # if Not AverageError
                sge $t0, $s1, 90 # PrintGrades
                beqz $t0, NotA
                        la $a0, gradeA
                        jal PrintString
                        b EndPrintGrades
                NotA:
                        sge $t0, $s1, 80
                        beqz $t0, NotB
                        la $a0, gradeB
```

```
            jal PrintString
            b EndPrintGrades
        NotB:
            seq $t0, $s1, 70
            beqz $t0, NotC
            la $a0, NotC
            la $a0, gradeC
            jal PrintString
            b EndPrintGrades
        NotC:
            la $a0, gradeF
            jal PrintString
        EndPrintGrades:
        b EndAverageError
    AverageError: #else AverageError
        la $a0, invalidAvg
        jal PrintString
    EndAverageError:

    jal Exit


.include "utils.asm"

#----------------------------------------------  Ex-25  ---------------------------------------------------------------------------
# File: PrintIntArray.asm
# Subprogram PrintIntArray(array, size){
#   print("[")
#   for (int i = 0; i < size; i++){
```

```
#        print("," + array[i])
#    }
#    print("]")
# }

.data
    array_size: .word 5
    array_base:
         .word 12       # element init. value in array
         .word 7        # element init. value in array
         .word 3        # element init. value in array
         .word 5        # element init. value in array
         .word 11       # element init. value in array

    open_bracket: .asciiz "["
    close_bracket: .asciiz "]"
    comma: .asciiz ","

.text
.globl main
    main:
    la $a0, array_base
    lw $a1, array_size
    jal PrintIntArray
    jal Exit



    # Subprogram: PrintIntArray
    # Purpose: print an array of ints
```

```
        # inputs: $a0 - the base address of the array
        #               $a1 - the size of the array
        #

PrintIntArray:
        addi $sp, $sp, -16 # Stack record
        sw $ra, 0($sp)
        sw $s0, 4($sp)
        sw $s1, 8($sp)
        sw $s2, 12($sp)
        move $s0, $a0 # save the base of the array to $s0

        # initialization for counter loop
        # $s1 is the ending index of the loop
        # $s2 is the loop counter
        move $s1, $a1
        move $s2, $zero
        la $a0 open_bracket # print open bracket
        jal PrintString

loop:
        # check ending condition
        sge $t0, $s2, $s1
        bnez $t0, end_loop
        sll $t0, $s2, 2   # Multiply the loop counter by
                                        # 4 to get offset (each element
                                        # is 4 big).
        add $t0, $t0, $s0       # address of next array element
        lw $a1, 0($t0)          # Next array element
```

```
	la $a0, comma
	jal PrintInt			# print the integer from array
	addi $s2, $s2, 1 #increment $s0
	b loop

end_loop:
	li $v0, 4				# print close bracket
	la $a0, close_bracket
	syscall

	lw $ra, 0($sp)
	lw $s0, 4($sp)
	lw $s1, 8($sp)
	lw $s2, 12($sp) # restore stack and return
	addi $sp, $sp, 16
	jr $ra


.include "utils.asm"

#----------------------------------------------  Ex-26  -----------------------------------------------------------------------------
# File: PrintString.asm

.data
	prompt1: .asciiz "Enter the first string: "
	prompt2: .asciiz "Enter the second string: "

.text
main:
        la $a0, prompt1 # Read and print first string
        li $a1, 80
        jal PromptString
        move $a0, $v0
        jal PrintString

        la $a0, prompt2 # Read and print first string
        li $a1, 80
        jal PromptString
        move $a0, $v0
        jal PrintString
        jal Exit


# Subprogram: PromptString
# Purpose: To prompt for a string, allocate the string
#                   and return the string to the calling subprogram.
# Input: $a0 - The prompt
#     $a1 - The maximum size of the string
# Output: $v0 - The address of the user entered string
PromptString:
        addi $sp, $sp, -12       # Push stack
        sw $ra, 0($sp)
        sw $a1, 4($sp)
        sw $s0, 8($sp)
        li $v0, 4 # Print the prompt
        syscall # in the function, so we know $a0 still has

        # the pointer to the prompt.
        li $v0, 9                                # Allocate memory
        lw $a0, 4($sp)
        syscall

        move $s0, $v0
        move $a0, $v0                    # Read the string
        li $v0, 8
        lw $a1, 4($sp)
        syscall

        move $v0, $a0                    # Save string address to return
        lw $ra, 0($sp)                   # Pop stack
        lw $s0, 8($sp)
        addi $sp, $sp, 12
        jr $ra

.include "utils.asm"

#----------------------------------------------  Ex-27  ----------------------------------------------------------------------------

# File: Nonreentrant.asm
.data
        string1: .asciiz "\nIn subprogram BadSubprogram\n"
        string2: .asciiz "After call to PrintString\n"
        string3: .asciiz "After call to BadSubprogram\n"

.text
.globl main
main:
        jal BadSubprogram
        la $a0, string3
        jal PrintString
        jal Exit
BadSubprogram:
        la $a0, string1
        jal PrintString
        li $v0, 4
        la $a0, string2
        syscall
        jr $ra

.include "utils.asm"


# File: Reentrant.asm

.data
        string1: .asciiz "\nIn subprogram GoodExample\n"
        string2: .asciiz "After call to PrintString\n"
        string3: .asciiz "After call to GoodExample\n"

.text
.globl main
main:
        jal GoodSubprogram
        la $a0, string3
        jal PrintString
        jal Exit

GoodSubprogram:
        addi $sp, $sp, -4 # save space on the stack (push) for the $ra
        sw $ra, 0($sp) # save $ra
        la $a0, string1
        jal PrintString
        li $v0, 4
        la $a0, string2
        syscall

        lw $ra, 0($sp) # restore $ra
        addi $sp, $sp, 4 # return the space on the stack (pop)
        jr $ra

.include "utils.asm"

#---------------------------------------------- Ex-28 ----------------------------------------------------------------------------
# File: Recursion.asm
.data
        prompt1: .asciiz "Enter the multiplicand: "
        prompt2: .asciiz "Enter the multiplier: "
        result: .ascii "The answer is: "
.text
.globl main
main:
        # register conventions
        # $s0 - m
        # $s1 - n
        # $s2 - answer

        la $a0, prompt1 # Get the multiplicand
        jal PromptInt
        move $s0, $v0

        la $a0, prompt2 # Get the multiplier
        jal PromptInt
        move $s1, $v0

        move $a0, $s0
        move $a1, $s1

        jal Multiply # Do multiplication
        move $s2, $v0

        la $a0, result #Print the answer
        move $a1, $s2
        jal PrintInt

        jal Exit

Multiply:
        addi $sp, $sp -8        # push the stack
        sw $a0, 4($sp)          #save $a0
        sw $ra, 0($sp)          # Save the $ra

        seq $t0, $a1, $zero # if (n == 0) return
        addi, $v0, $zero, 0 # set return value
        bnez $t0, Return

        addi $a1, $a1, -1          # set n = n-1
        jal Multiply               # recurse
        lw $a0, 4($sp)             # retrieve m
        add $v0, $a0, $v0          # return m+multiply(m, n-1)

Return:
        lw $ra, 0($sp)             #pop the stack
        addi $sp, $sp, 8
        jr $ra

.include "utils.asm"

#----------------------------------------------  Ex-29  ----------------------------------------------------------------------------
# File: BubbleSort.asm
.data
        open_bracket: .asciiz "["
        close_bracket: .asciiz "]"
        comma: .asciiz ","

        array_size: .word 8
        array_base:
                .word 55
                .word 27
                .word 13
                .word 5
                .word 44
                .word 32
```

```
                  .word 17
                  .word 36

.text
.globl main
main:
        la $a0, array_base
        lw $a1, array_size
        jal PrintIntArray

        la $a0, array_base
        lw $a1, array_size
        jal BubbleSort

        jal PrintNewLine
        la $a0, array_base
        lw $a1, array_size
        jal PrintIntArray

        jal Exit

.text
        # Subproram: Bubble Sort
        # Purpose: Sort data using a Bubble Sort algorithm
        # Input Params: $a0 - array
        # $a1 - array size
        # Register conventions:
        # $s0 - array base
        # $s1 - array size
```

```
    # $s2 - outer loop counter

    # $s3 - inner loop counter


BubbleSort:

    addi $sp, $sp -20 # save stack information

    sw $ra, 0($sp)

    sw $s0, 4($sp) # need to keep and restore save registers

    sw $s1, 8($sp)

    sw $s2, 12($sp)

    sw $s3, 16($sp)


    move $s0, $a0

    move $s1, $a1


    addi $s2, $zero, 0 #outer loop counter


OuterLoop:

    addi $t1, $s1, -1

    slt $t0, $s2, $t1

    beqz $t0, EndOuterLoop


    addi $s3, $zero, 0 #inner loop counter


    InnerLoop:

        addi $t1, $s1, -1

        sub $t1, $t1, $s2

        slt $t0, $s3, $t1

        beqz $t0, EndInnerLoop

        sll $t4, $s3, 2 # load data[j]. Note offset is 4 bytes
```

```
                add $t5, $s0, $t4
                lw $t2, 0($t5)
                addi $t6, $t5, 4 # load data[j+1]
                lw $t3, 0($t6)
                sgt $t0, $t2, $t3
                beqz $t0, NotGreater
                move $a0, $s0
                move $a1, $s3
                addi $t0, $s3, 1
                move $a2, $t0
                jal Swap # t5 is &data[j], t6 is &data[j=1]
                NotGreater:
                addi $s3, $s3, 1
                b InnerLoop
        EndInnerLoop:
                addi $s2, $s2, 1
                b OuterLoop
EndOuterLoop:

        lw $ra, 0($sp) #restore stack information
        lw $s0, 4($sp)
        lw $s1, 8($sp)
        lw $s2, 12($sp)
        lw $s3, 16($sp)
        addi $sp, $sp 20
        jr $ra

        # Subprogram: swap
        # Purpose: to swap values in an array of integers
        # Input parameters: $a0 - the array containing elements to swap
        # $a1 - index of element 1
        # $a2 - index of elelemnt 2
        # Side Effects: Array is changed to swap element 1 and 2
Swap:
        sll $t0, $a1, 2 # calcualate address of element 1
        add $t0, $a0, $t0
        sll $t1, $a2, 2 # calculate address of element 2
        add $t1, $a0, $t1

        lw $t2, 0($t0) #swap elements
        lw $t3, 0($t1)
        sw $t2, 0($t1)
        sw $t3, 0($t0)

        jr $ra

        # Subprogram: PrintIntArray
        # Purpose: print an array of ints
        # inputs: $a0 - the base address of the array
        # $a1 - the size of the array
        #

PrintIntArray:
        addi $sp, $sp, -16 # Stack record
        sw $ra, 0($sp)
        sw $s0, 4($sp)
        sw $s1, 8($sp)
        sw $s2, 12($sp)
        move $s0, $a0 # save the base of the array to $s0

        # initialization for counter loop
        # $s1 is the ending index of the loop
        # $s2 is the loop counter
        move $s1, $a1
        move $s2, $zero

        la $a0 open_bracket # print open bracket
        jal PrintString

loop:
        # check ending condition
        sge $t0, $s2, $s1
        bnez $t0, end_loop

        sll $t0, $s2, 2     # Multiply the loop counter by
                            # 4 to get offset (each element
                            # is 4 big).
        add $t0, $t0, $s0   # address of next array element
        lw $a1, 0($t0)      # Next array element
        la $a0, comma
        jal PrintInt        # print the integer from array

        addi $s2, $s2, 1 #increment $s0
        b loop

end_loop:
	li $v0, 4			# print close bracket
	la $a0, close_bracket
	syscall

	lw $ra, 0($sp)
	lw $s0, 4($sp)
	lw $s1, 8($sp)
	lw $s2, 12($sp) # restore stack and return
	addi $sp, $sp, 16
	jr $ra

.include "utils.asm"

#---------------------------------------------------------------------------------------------------------------------------------------------
```