Republic of the Philippines
CAMARINES SUR POLYTECHNIC COLLEGES
Nabua, Camarines Sur

COLLEGE *of* COMPUTER STUDIES

CS 318 – Architecture and Organization

LEARNING TASK (STRUCTURED ASSEMBLY LANGUAGE P1)

NAME: JOSHUA JERICHO D. BARJA SECTION: BSCS-3A

SAMPLE RUN

Step-by-step sample run of your assembly program with explanation.

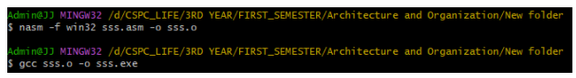
<table>
<tr>
<td>

Step 1

1. *Create file that has an .asm extension (e.g, sample.asm)*
2. *In your dedicated terminal, run the command <nasm -f win32 sample.asm -o sample.o> then hit enter.*
3. *After the first bash/command, type in another bash/command <gcc sample.o -o sample.exe> to perform linking then press enter to execute.*

</td>
<td>

Image 1

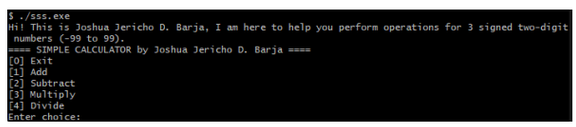


</td>
</tr>
<tr>
<td>

Step 2

1. *Run the executable file using the command <./yourExeFile.exe> in your dedicated terminal and hit enter to execute.*

</td>
<td>



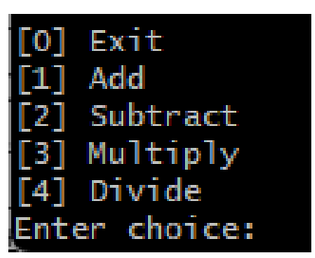
</td>
</tr>
<tr>
<td>

Step 3

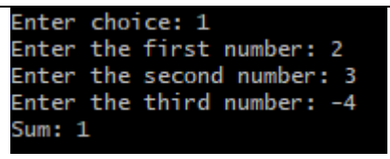
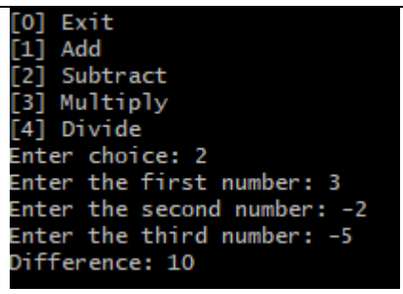
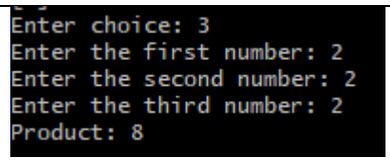
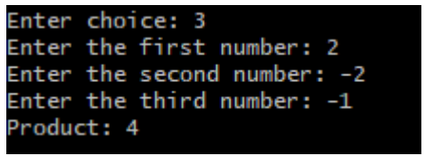
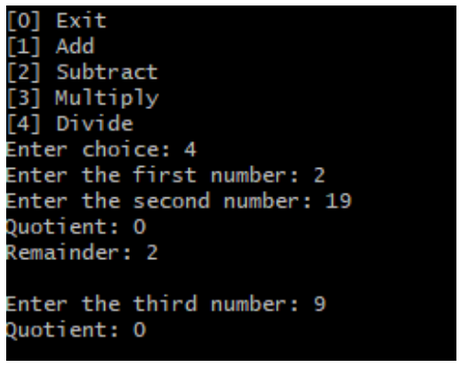
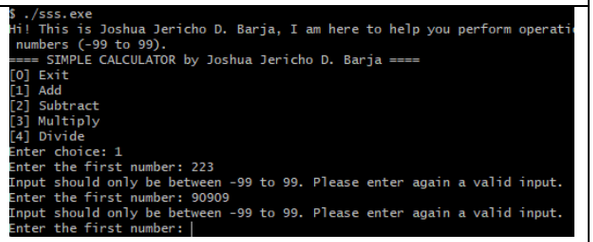
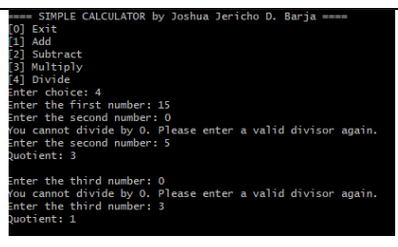
1. *Enter a number ranging from 1-4. Choose what operation you want to try:*

±	*1*	*2*	*3*	*4*
add	☒			
sub		☒		
mul			☒	
div				☒

</td>
<td>



</td>
</tr>
</table>

TEST CASES	*RESULTS*
Addition inputs (2, 3, -4) *Expected result: 1*	
Subtraction inputs (3, -2, -5) *Expected result: 10*	
Multiplication inputs (2, 2, 2) and (2, -2, -1) *Expected result: 8 and 4*	 
Division inputs (2, 19, 9) *Expected result:* *Enter the first number: 2* *Enter the second number: 19* *Quotient: **0*** *Remainder: 2* *Enter the third number: 9* *Quotient: **0***	
ERRORS	*RESULTS*
Expected numbers ranging from -99 to 99	
Division Error	

ASSEMBLERS INSTRUCTIONS

Selection Statements	***Iterative Statements***
cmp – Compares two values to determine the next action. It is used to check the user's menu choice to decide which arithmetic operation to execute and to validate input values within the allowed range (-99 to 99). *je (jump if equal) – Transfers control to a specific section if the compared values are equal. For example, it jumps to the addition, subtraction, multiplication, or division routines based on the user's choice.* *jne (jump if not equal) – Transfers control when the compared values are not equal. It is used to ensure the divisor is not zero before performing division.* *jl / jg (jump if less / jump if greater) – Used to enforce input constraints. If a number is less than -99 or greater than 99, the program jumps to a routine that prompts for a valid input.*	*jmp – Used to repeat a section of code until a condition is met. Examples:* *menu_loop – Loops the main menu until the user chooses to exit.* *get_input_loop – Repeats the input prompt until a valid number is entered.* *divisor_check – Repeats the second number input if the divisor is zero.* *thirdnum_check – Repeats the third number input if it is zero.* *Labeled loops (menu_loop, get_input_loop, divisor_check, thirdnum_check) – Ensure repeated execution of prompts and validation, preventing invalid inputs or division by zero, while maintaining smooth program flow.*

PROGRAM CODE

```
section .data
    header_msg db 'Hi! This is Joshua Jericho D. Barja, I am here to help you perform operations for 3 signed two-digit numbers (-99 to 99).', 10, \
                  '==== SIMPLE CALCULATOR by Joshua Jericho D. Barja ====', 10, 0
    options db '[0] Exit', 10, \
               '[1] Add', 10, \
               '[2] Subtract', 10, \
               '[3] Multiply', 10, \
               '[4] Divide', 10, 0
    choice db 'Enter choice: ', 0
    firstNum db 'Enter the first number: ', 0
    secondNum db 'Enter the second number: ', 0
    thirdNum db 'Enter the third number: ', 0
    input_format db '%d', 0
    exit_msg db 'Thank you!', 10, 0
    not_valid db 'Input should only be between -99 to 99. Please enter again a valid input.', 10, 0
    zero_divisor db 'You cannot divide by 0. Please enter a valid divisor again.', 10, 0
    sum_msg db 'Sum: %d', 10, 0
    sub_msg db 'Difference: %d', 10, 0
    mul_msg db 'Product: %d', 10, 0
    div_msg db 'Quotient: %d', 10, 0
    div_remainder db 'Remainder: %d', 10, 0
    new_line db 10, 0

section .bss
    choice_num resd 1
    fNum resd 1
    sNum resd 1
    tNum resd 1
    result resd 1
```

```
    remainder resd 1

section .text
    global _main
    extern _printf
    extern _scanf

_main:

menu_loop:
    ; print header message
    push header_msg
    call _printf
    add esp, 4

    ; print options
    push options
    call _printf
    add esp, 4

    ; ask for choice
    push choice
    call _printf
    add esp, 4

    ; read choice
    push choice_num
    push input_format
    call _scanf
    add esp, 8

    ; load option into eax register
    mov eax, [choice_num]

    ; exit program if 0
```

```
    cmp eax, 0
    je exit_loop

    ; addition
    cmp eax, 1
    je addition

    ; subtraction
    cmp eax, 2
    je subtraction

    ; multiplication
    cmp eax, 3
    je multiplication

    ; division
    cmp eax, 4
    je division

    ; invalid choice, loop again
    jmp menu_loop

;---------------------------------------------- ADDITION ------------
---------------------------------
addition:
    ; first number
    push fNum
    push firstNum
    call get_valid_input
    add esp, 8

    ; second number
    push sNum
    push secondNum
    call get_valid_input
```

```
    add esp, 8

    ; third number
    push tNum
    push thirdNum
    call get_valid_input
    add esp, 8

    ; compute result
    mov eax, [fNum]
    add eax, [sNum]
    add eax, [tNum]
    mov [result], eax

    ; print the result
    push dword [result]
    push sum_msg
    call _printf
    add esp, 8

    ; newline for spacing
    push new_line
    call _printf
    add esp, 4

    ; jump back to menu
    jmp menu_loop

;---------------------------------------------- SUBTRACTION ---------
-----------------------------------
subtraction:
    ; first number
    push fNum
    push firstNum
    call get_valid_input
```

```
    add esp, 8

    ; second number
    push sNum
    push secondNum
    call get_valid_input
    add esp, 8

    ; third number
    push tNum
    push thirdNum
    call get_valid_input
    add esp, 8

    ; compute result
    mov eax, [fNum]
    sub eax, [sNum]
    sub eax, [tNum]
    mov [result], eax

    ; print the result
    push dword [result]
    push sub_msg
    call _printf
    add esp, 8

    ; newline for spacing
    push new_line
    call _printf
    add esp, 4

    ; jump back to menu
    jmp menu_loop
```

```
;------------------------------------------- MULTIPLICATION ------
--------------------------------------
multiplication:
    ; first number
    push fNum
    push firstNum
    call get_valid_input
    add esp, 8

    ; second number
    push sNum
    push secondNum
    call get_valid_input
    add esp, 8

    ; third number
    push tNum
    push thirdNum
    call get_valid_input
    add esp, 8

    ; multiply
    mov eax, [fNum]
    imul eax, [sNum]
    imul eax, [tNum]
    mov [result], eax

    ; print the result
    push dword [result]
    push mul_msg
    call _printf
    add esp, 8

    ; newline for spacing
    push new_line
```

```
    call _printf
    add esp, 4

    jmp menu_loop

;----------------------------------------- DIVISION ------------
---------------------------------
division:
    ; first number
    push fNum
    push firstNum
    call get_valid_input
    add esp, 8

divisor_check:
    ; second number (divisor)
    push sNum
    push secondNum
    call get_valid_input
    add esp, 8

    ; check if second number is zero
    mov eax, [sNum]
    cmp eax, 0
    jne perf_division1   ; if not zero, proceed
    push zero_divisor
    call _printf
    add esp, 4
    jmp divisor_check    ; repeat only the second number

perf_division1:
    ; divide first number by second number
    mov eax, [fNum]
    cdq
    mov ebx, [sNum]
```

```
    idiv ebx
    mov [result], eax
    mov [remainder], edx

    ; print quotient
    push dword [result]
    push div_msg
    call _printf
    add esp, 8

    ; print remainder if non-zero
    cmp dword [remainder], 0
    je skip1
    push dword [remainder]
    push div_remainder
    call _printf
    add esp, 8
skip1:
    push new_line
    call _printf
    add esp, 4

thirdnum_check:
    ; third number
    push tNum
    push thirdNum
    call get_valid_input
    add esp, 8

    ; check if third number is zero
    mov eax, [tNum]
    cmp eax, 0
    jne perf_division2   ; if not zero, proceed
    push zero_divisor
    call _printf
```

```
    add esp, 4
    jmp thirdnum_check   ; repeat only the third number

perf_division2:
    ; divide previous result by third number
    mov eax, [result]
    cdq
    mov ebx, [tNum]
    idiv ebx
    mov [result], eax
    mov [remainder], edx

    ; print final quotient
    push dword [result]
    push div_msg
    call _printf
    add esp, 8

    ; print remainder if non-zero
    cmp dword [remainder], 0
    je skip2
    push dword [remainder]
    push div_remainder
    call _printf
    add esp, 8
skip2:
    push new_line
    call _printf
    add esp, 4

    jmp menu_loop

; ------------------------------------------- VALIDATION FUNCTION
-------------------------------------------
get_valid_input:
```

```
    push ebp
    mov ebp, esp

get_input_loop:
    ; display prompt
    push dword [ebp + 8]      ; prompt message
    call _printf
    add esp, 4

    ; read number
    push dword [ebp + 12]     ; variable address
    push input_format
    call _scanf
    add esp, 8

    ; load number to eax
    mov ebx, [ebp + 12]
    mov eax, [ebx]

    ; check range -99 to 99
    cmp eax, -99
    jl invalid_input
    cmp eax, 99
    jg invalid_input
    jmp valid_input

invalid_input:
    push not_valid
    call _printf
    add esp, 4
    jmp get_input_loop

valid_input:
    pop ebp
    ret
```

```
;-------------------------------------------- EXIT PROGRAM CODE :>
-------------------------------------------
exit_loop:
    push exit_msg
    call _printf
    add esp, 4
    ret
```