**Assembly Program I**

.data

plain BYTE ?

code BYTE ?

prompt BYTE "Enter a string: ", 0

msg BYTE "The encoded string is:", 0

buff BYTE ?100 ;comparable to x86's DUP(100)?.

.code ;Keep procedures here.

encode PROC ; Encode a character.

;if(( plain >= 'a' && plain <= 'x') ||

; (plain => 'A' && plain <= 'X')).

cond1:

; ( plain >= 'a' && plain <= 'x').

compare\_ge plain, 'a'

compare\_le plain, 'x'

branch\_and cond1\_true

compare\_ge plain, 'A'

compare\_le plain, 'X'

branch\_and cond1\_true

jumpto else\_L

cond1\_true:

add plain, 2

jumpto end\_proc

else\_L:

cond2:

compare\_eq plain, 'y'

compare\_eq plain, 'z'

compare\_eq plain, 'Y'

compare\_eq plain, 'Z'

branch cond2\_else

jumpto end\_proc

cond2\_else:

subtract plain, 24

end\_proc:

move plain, code

move code, RV1

return

END PROC

getline PROC ; Implementation of the STL getline() method.

; Take in a pointer to a bytestring and store it into buff

move 0, GP1

l:

;While the current character is not null and

;the current index is less than AR2, store the byte

move [AR3+GP1], [AR1+GP1] ; Move the current character

;at the GP1 index of AR3 into

;the same index at AR1.

add GP1, 1 ; Increment the counter.

compare\_lt GP1, AR2 ; Check if we're within our limit.

compare\_eq 0, [AR3+GP1] ; Or that the current character

; is not null.

branch e

jumpto l

e:

return

END PROC

.run ; Main code run here. Use procedures in .code.

; Print out the prompt.

move [prompt], AR1

call WriteString

call GetString ; GetString returns an offset to the BYTE

;array it created into RV1

move [buff], AR1

move 100, AR2

move RV1, AR3

call getline ; Move the line into buff

; Do a loop (not a barrel roll) to encode each character in

;the buf array by passing it to the encode procedure.

move 0, GP1

top\_loop:

move buff[GP1], plain

call encode ; Encode has a return value in RV1.

move RV1, buff[GP1] ; Move that return value to GP1.

compare\_eq 0, buff[GP1]

branch end\_of\_loop

add GP1, 1

jumpto top\_loop

end\_of\_loop:

; Print out the encoded string.

move [msg], AR1

call WriteString

move [buff], AR1

call WriteString

call Newline

**Assembly Program II**

; Have the user enter in 10 non-negative integers and sum up those

; non-negative integers and find the average.

.data

sum DWORD ?

average DWORD ?

prompt BYTE "Enter a non-negative integer: ",0 ;Prompt for user to

; enter an integer.

message BYTE "Average= ",0

.run

move 0, GP4

top\_loop:

move [prompt], AR1 ; Move the address of the prompt variable

; to the first argument register for the

; WriteString method to read.

call WriteString ; WriteString looks at the value of AR1 as a

; memory location reference and reads the

; values in sequential memory until it hits a

; null value signifying the end of a string.

call GetInt ; GetInt gets an integer value from

; the user and returns it to RV1.

add sum, RV1 ; Add the value to the sum.

call WriteNewline ; Assuming built-in method. Writes a newline

; character to the output.

add GP4, 1 ; increment the loop counter.

compare\_lt GP4, 9 ; Check for loop bounds.

branch top\_loop ; Relative label addressing.

divide2 sum, 10, average ; Do an integer division operation of

; average=sum/10.

move [message], AR1

call WriteString ; Assuming built-in method.

move [average], AR1

call WriteInt ; Assuming built-in method.

call WriteNewline ; Assuming built-in method.