

**Name:** Shanne Edralyn N. Erandio  
**Course, Year and Section:** BSIT 2A

**Date: 11/27/2025**

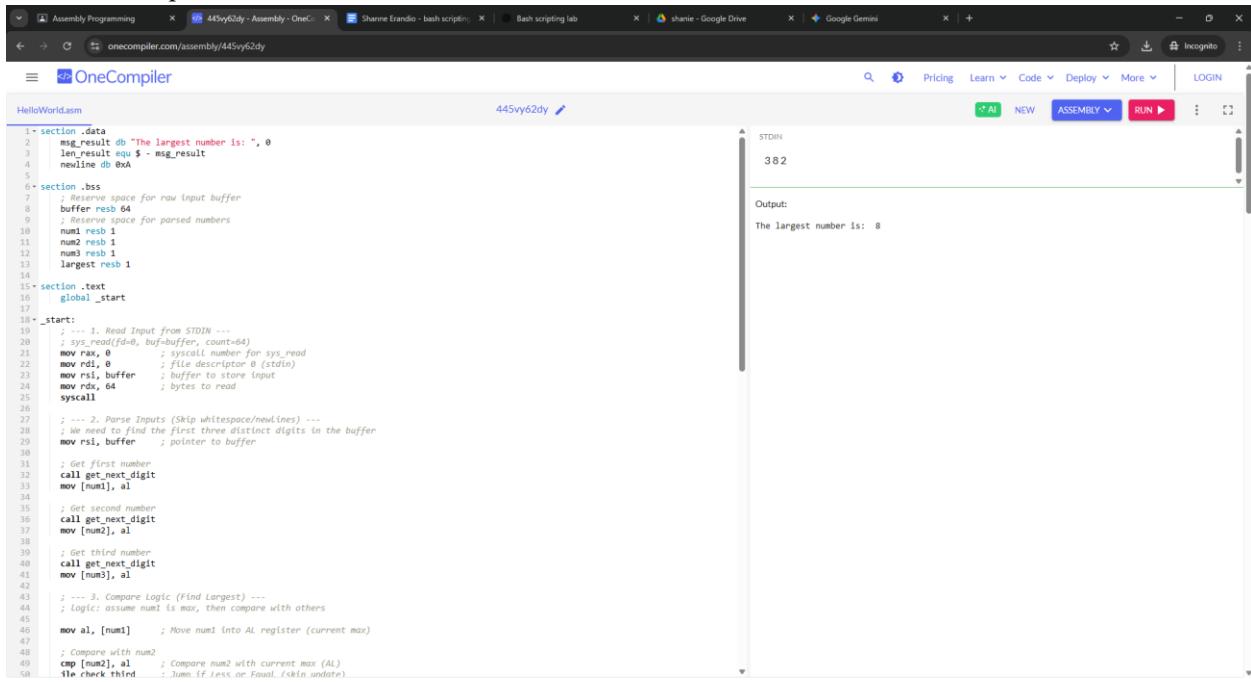
## Assembly Programming

### Case Study 5: Largest of Three Numbers

Write an Assembly program that:

- Takes three single-digit inputs
- Compares them
- Displays the largest number
- Use conditional jumps (JG, JL, JE) or CMP instructions.

#### Code and Output



The screenshot shows a web-based assembly compiler interface. The code editor contains assembly code for a program named "HelloWorld.asm". The code reads three single-digit numbers from standard input (stdin), parses them, and finds the largest. The output window shows the result "The largest number is: 8".

```
1 - section .data
2   msg_result db "The largest number is: ", 0
3   len_result equ $ - msg_result
4   newline db 0xA
5
6 - section .bss
7   ; Reserve space for raw input buffer
8   buffer resb 64
9   ; Reserve space for parsed numbers
10  num1 resb 1
11  num2 resb 1
12  num3 resb 1
13  largest resb 1
14
15 - section .text
16   global _start
17
18 - _start:
19   ; --- 1. Read Input from STDIN ---
20   mov rdi, 0           ; syscall number for sys_read
21   mov rsi, buffer      ; file descriptor 0 (stdin)
22   mov rdx, 64          ; buffer to store input
23   mov rax, 0            ; bytes to read
24   syscall
25
26   ; --- 2. Parse Inputs (Skip whitespace/newlines) ---
27   ; We need to find the first three distinct digits in the buffer
28   mov rsi, buffer      ; pointer to buffer
29
30   ; Get first number
31   call get_next_digit
32   mov [num1], al
33
34   ; Get second number
35   call get_next_digit
36   mov [num2], al
37
38   ; Get third number
39   call get_next_digit
40   mov [num3], al
41
42   ; --- 3. Compare Logic (Find Largest) ---
43   ; logic: assume num1 is max, then compare with others
44
45   mov al, [num1]         ; Move num1 into AL register (current max)
46
47   ; Compare with num2
48   cmp [num2], al        ; Compare num2 with current max (AL)
49   jle check_third       ; Jump if less or equal. If not, skip update
50
51   ; Check if num3 is larger than current max (AL)
52   cmp [num3], al
53   jle exit              ; If not, skip update
54
55   ; Update max if num3 is larger
56   mov al, [num3]
57
58   ; Print result
59   mov rsi, msg_result
60   mov rdi, 1             ; syscall number for sys_write
61   mov rdx, len_result
62   syscall
63
64   ; Exit program
65   mov rax, 60
66   mov rdi, 0
67   syscall
```

The screenshot shows the OneCompiler interface with the assembly code for `HelloWorld.asm`. The code implements a program to find the largest number from three inputs. It uses `cmp`, `jle`, and `mov` instructions to compare and update the maximum value. The `sys_write` and `sys_exit` system calls are used for output and termination. The output window shows the result `382`.

```

47      ; Compare with num2
48      cmp [num2], al          ; Compare num2 with current max (AL)
49      jle check_third         ; Jump if Less or Equal (skip update)
50      mov al, [num2]           ; Update max: AL = num2
51
52      .check_third:
53      ; Compare with num3
54      cmp [num3], al          ; Compare num3 with current max (AL)
55      jle store_result        ; Jump if Less or Equal (skip update)
56      mov al, [num3]           ; Update max: AL = num3
57
58      store_result:
59      mov [largest], al         ; Store the final Largest value
60
61      ; --- 4. Display Output ---
62
63      ; Print "The Largest number is: "
64      mov rax, 1               ; sys_write
65      mov rdi, 1               ; stdio
66      mov rsi, msg_result       ; address of the message
67      mov rdx, len_result       ; length (1 byte)
68
69      syscall
70
71      ; Print the Largest number
72      mov rax, 1               ; sys_write
73      mov rdi, 1               ; stdio
74      mov rsi, largest          ; address of the number
75      mov rdx, 1               ; Length (1 byte)
76
77      syscall
78
79      ; Print newline
80      mov rax, 1               ; sys_write
81      mov rdi, 1               ; stdio
82      mov rsi, newline          ; address of the newline character
83      mov rdx, 1               ; Length (1 byte)
84
85      syscall
86
87      ; --- 5. Exit ---
88      mov rax, 60               ; sys_exit
89      xor rdi, rdi             ; status 0
90
91      syscall
92
93      ; Helper Subroutine: get_next_digit
94      ; Scans RSI until it finds a digit (ASCII '0'-'9').
95      ; Returns character in AL.
96      ; Updates RSI to point to next position.
97      ; Updates RSI to point to next position.
98      ; Updates RSI to point to next position.
99      ; Updates RSI to point to next position.
100     ; Updates RSI to point to next position.
101     ; Updates RSI to point to next position.
102     ; Updates RSI to point to next position.
103     ; Checks if it's the end of buffer (NULL or @ Length read assumption)
104     ; CMP AL, 0
105     ; JLE .scan_loop ; If < '0', it's garbage/whitespace, keep scanning
106     ; CMP AL, '9'
107     ; JG .scan_loop ; If > '9', it's garbage/whitespace, keep scanning
108
109     ; If we are here, it's a valid digit
110     ret
111     .done:
112     reti

```

This screenshot shows the same assembly code as the first one, but with some line numbers removed and some comments added. The code remains identical to the first version.

```

63
64      ; Print "The Largest number is: "
65      mov rax, 1               ; sys_write
66      mov rdi, 1               ; stdio
67      mov rsi, msg_result       ; address of the message
68      mov rdx, len_result       ; length (1 byte)
69
70      syscall
71
72      ; Print the Largest number
73      mov rax, 1               ; sys_write
74      mov rdi, 1               ; stdio
75      mov rsi, largest          ; address of the number
76      mov rdx, 1               ; Length (1 byte)
77
78      syscall
79
80      ; Print newline
81      mov rax, 1               ; sys_write
82      mov rdi, 1               ; stdio
83      mov rsi, newline          ; address of the newline character
84
85      syscall
86
87      ; --- 5. Exit ---
88      mov rax, 60               ; sys_exit
89      xor rdi, rdi             ; status 0
90
91      syscall
92
93      ; Helper Subroutine: get_next_digit
94      ; Scans RSI until it finds a digit (ASCII '0'-'9').
95      ; Returns character in AL.
96      ; Updates RSI to point to next position.
97      ; Updates RSI to point to next position.
98      ; Updates RSI to point to next position.
99      ; Updates RSI to point to next position.
100     ; Updates RSI to point to next position.
101     ; Updates RSI to point to next position.
102     ; Updates RSI to point to next position.
103     ; Checks if it's the end of buffer (NULL or @ Length read assumption)
104     ; CMP AL, 0
105     ; JLE .scan_loop ; If < '0', it's garbage/whitespace, keep scanning
106     ; CMP AL, '9'
107     ; JG .scan_loop ; If > '9', it's garbage/whitespace, keep scanning
108
109     ; If we are here, it's a valid digit
110     ret
111     .done:
112     reti

```

In this activity, I learned how Assembly handles input, comparisons, and output at a very low level using system interrupts. Writing a program to determine the largest number helped me understand the use of `CMP` and conditional jump instructions like `JG` and `JL`. I also learned how ASCII values must be converted before comparing numbers. Overall, this exercise improved my understanding of how high-level logic is translated into step-by-step CPU operations.

The primary goal of this case study was to implement conditional logic in a low-level language. Unlike high-level languages where `if/else` structures are built-in, Assembly requires the programmer to manually manage control flow using the processor's status flags and jump instructions. The specific task was to determine the largest of three single-digit integers provided via standard input.

This exercise highlighted the trade-offs of Assembly language. While it offers precise control over registers and execution flow, it demands explicit management of trivial tasks like parsing ASCII characters and skipping whitespace. Mastering `CMP` and conditional jumps (`JG`, `JL`, `JE`) provides a fundamental understanding of how computer processors actually make decisions at the hardware level.