

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

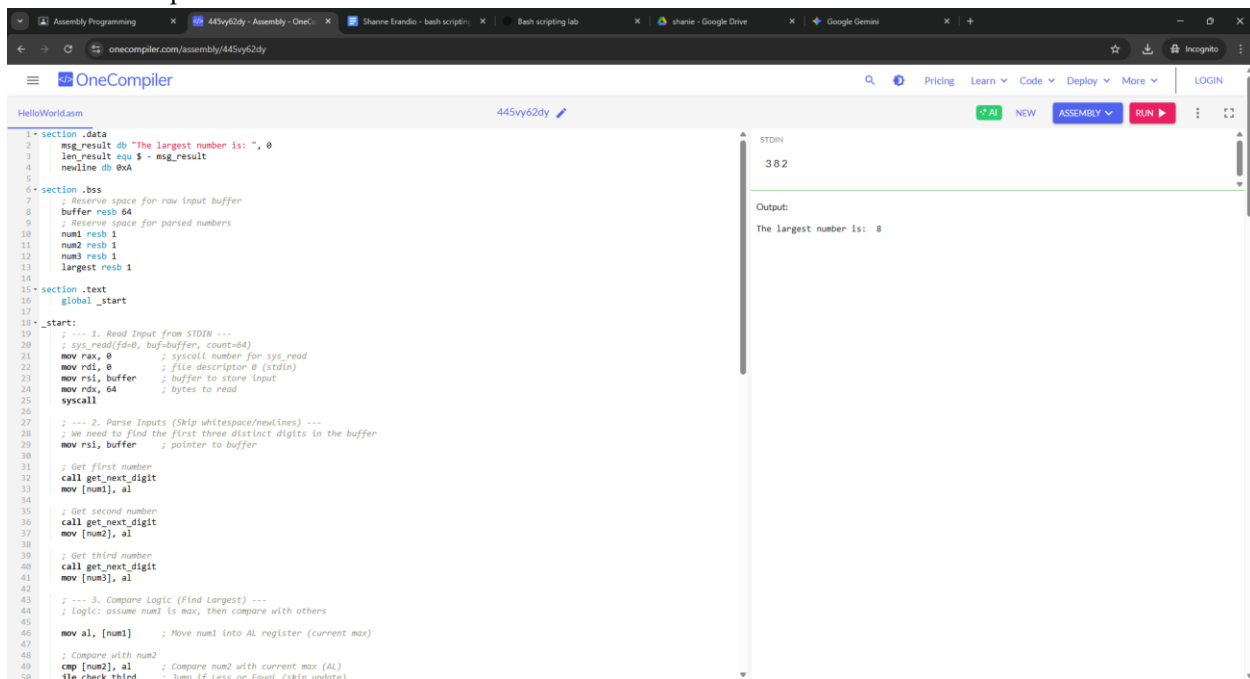
Date: 11/27/2025

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



```
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 ; sys_read(fd=0, buf=buffer, count=64)
21 mov rax, 0 ; syscall number for sys_read
22 mov rdi, 0 ; file descriptor 0 (stdin)
23 mov rsi, buffer ; buffer to store input
24 mov rdx, 64 ; bytes to read
25 syscall
26
27 ; --- 2. Parse Inputs (Skip whitespace/newlines) ---
28 ; We need to find the first three distinct digits in the buffer
29 mov rsi, buffer ; pointer to buffer
30
31 ; Get first number
32 call get_next_digit
33 mov [num1], al
34
35 ; Get second number
36 call get_next_digit
37 mov [num2], al
38
39 ; Get third number
40 call get_next_digit
41 mov [num3], al
42
43 ; --- 3. Compare Logic (Find Largest) ---
44 ; Logic: assume num1 is max, then compare with others
45
46 mov al, [num1] ; Move num1 into AL register (current max)
47
48 ; Compare with num2
49 cmp [num2], al ; Compare num2 with current max (AL)
50 ; If num2 is less, or equal, skip update
51 jle check_third
52
53 ; Update max to num2
54 mov al, [num2]
55
56 ; Compare with num3
57 cmp [num3], al ; Compare num3 with current max (AL)
58 ; If num3 is less, or equal, skip update
59 jle check_third
60
61 ; Update max to num3
62 mov al, [num3]
63
64 ; Print the result
65 mov rdi, msg_result
66 mov rdx, len_result
67 syscall
68
69 ; Print newline
70 mov rdi, newline
71 mov rdx, 1
72 syscall
73
74 ; Exit
75 mov rax, 0
76 syscall
```

STDIN
382

Output:
The largest number is: 8

```
47 ; Compare with num2
48
49 cmp [num2], al ; Compare num2 with current max (AL)
50 jle check_third ; Jump if Less or Equal (skip update)
51 mov al, [num2] ; Update max: AL = num2
52
53 * check_third:
54 ; Compare with num3
55 cmp [num3], al ; Compare num3 with current max (AL)
56 jle store_result ; Jump if Less or Equal (skip update)
57 mov al, [num3] ; Update max: AL = num3
58
59 * store_result:
60 mov [largest], al ; Store the final Largest value
61
62 ; --- 4. Display Output ---
63
64 ; Print "The largest number is: "
65 mov rax, 1 ; sys_write
66 mov rdi, 1 ; stdout
67 mov rsi, msg_result
68 mov rdx, len_result
69 syscall
70
71 ; Print the largest number
72 mov rax, 1 ; sys_write
73 mov rdi, 1 ; stdout
74 mov rsi, largest ; address of the number
75 mov rdx, 1 ; length (1 byte)
76 syscall
77
78 ; Print newline
79 mov rax, 1 ; sys_write
80 mov rdi, 1 ; stdout
81 mov rsi, newline
82 mov rdx, 1 ; length (1 byte)
83 syscall
84
85 ; --- 5. Exit ---
86 mov rax, 60 ; sys_exit
87 xor rdi, rdi ; status 0
88 syscall
89
90 ; --- Helper Subroutine: get_next_digit ---
91 ; Scans RSI until it finds a digit (ASCII '0'-'9').
92 ; Returns character in AL.
93 ; Updates RSI to point to next position.
94 * get_next_digit:
95 * .scan_loop:
96 * mov al, [rsi] ; Load byte from buffer
97 * inc rsi ; increment pointer
98 *
99 * ; Check if it's the end of buffer (null or 0 length read assumption)
100 * cmp al, 0
101 * je .done
102 *
103 * ; Check if it's a digit (ASCII 48 to 57)
104 * cmp al, '0'
105 * jl .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 * ret
```

```
63
64 ; Print "The largest number is: "
65 mov rax, 1 ; sys_write
66 mov rdi, 1 ; stdout
67 mov rsi, msg_result
68 mov rdx, len_result
69 syscall
70
71 ; Print the largest number
72 mov rax, 1 ; sys_write
73 mov rdi, 1 ; stdout
74 mov rsi, largest ; address of the number
75 mov rdx, 1 ; length (1 byte)
76 syscall
77
78 ; Print newline
79 mov rax, 1 ; sys_write
80 mov rdi, 1 ; stdout
81 mov rsi, newline
82 mov rdx, 1 ; length (1 byte)
83 syscall
84
85 ; --- 5. Exit ---
86 mov rax, 60 ; sys_exit
87 xor rdi, rdi ; status 0
88 syscall
89
90 ; --- Helper Subroutine: get_next_digit ---
91 ; Scans RSI until it finds a digit (ASCII '0'-'9').
92 ; Returns character in AL.
93 ; Updates RSI to point to next position.
94 * get_next_digit:
95 * .scan_loop:
96 * mov al, [rsi] ; Load byte from buffer
97 * inc rsi ; increment pointer
98 *
99 * ; Check if it's the end of buffer (null or 0 length read assumption)
100 * cmp al, 0
101 * je .done
102 *
103 * ; Check if it's a digit (ASCII 48 to 57)
104 * cmp al, '0'
105 * jl .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 * ret
```

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.