Research Question: Assembly Language in Modern Computing

Assembly language is a major component of modern computing, especially in areas that require high degrees of efficiency and close hardware connection. Assembly language plays a crucial role in embedded systems, which are essential to industrial automation, automotive, and consumer electronics because it provides accurate control over microcontrollers and microprocessors. The construction of extremely effective and performance-optimized systems is made easier by this direct hardware manipulation, which is essential in situations where computational resources are few.

Additionally, assembly language plays a crucial role in the development of device drivers, which serve as a conduit between an operating system and a computer's hardware. Assembly language is required in this field because it can carry out low-level operations that are inefficiently handled by high-level languages, guaranteeing optimal hardware. both reliability and efficacy.

Another area where assembly language is quite useful is performance optimisation. Assembly language's efficiency can result in significant performance and responsiveness gains in situations where microseconds matter, such high-frequency trading systems or real-time data processing applications. Assembly language lets programmers write code that directly modifies hardware resources, which makes it possible to optimise crucial code segments that high-level languages are unable to accomplish because of their abstraction from the hardware.

Even with its great potential, writing in assembly language is typically more difficult and time-consuming than utilising high-level languages and necessitates a thorough understanding of computer architecture. As a result of this complexity and the development of more potent and effective compilers, assembly language utilisation has decreased. for the creation of generic applications. Assembly language is still a vital tool in the specialised domains of embedded systems, device driver development, and performance-critical applications.

In summary, even though high-level programming languages are becoming more and more common, assembly language's special qualities guarantee that it will always be important in some fields of computer technology. It is a crucial tool for the creation of embedded systems, device drivers, and performance-critical application optimisation due to its direct and effective control over hardware.

Sources:

V. Kanade, "What Is Assembly Language? Working, Features, and Advantages," Spiceworks, 15 June 2023. [Online]. Available: https://www.spiceworks.com/tech/tech-general/articles/what-is-assembly-language.

A. Taylor, "Exploring Assembly Language: An Introduction to Programming," ReversePCB,

November 12, 2022. [Online]. Available: https://reversepcb.com/exploring-assembly-language-an-introduction-to-programming.

AuroraSolutionsAS, "Assembly Language: The Bridge Between Human and Machine," Medium, Mar 15, 2023. [Online]. Available: https://medium.com/@aurorasolutionsas/assembly-language-the-bridge-between-human-and-machine-8624a0d2051c.

New Project
My Projects
Execute History
Collaborate
Save
Save As
Editable Share
Instant Share (No Login/Save required)
Copy to Clipboard
Open (from local file)
Save (to local file)
Pretty Print
IDE Settings
How To/FAQ

```
section .text
    global _start

_start:
    ; Subtract 57 from 95
    mov eax, 95         ; Move the value 95 into eax
    sub eax, 57         ; Subtract 57 from eax

    ; Convert the result to ASCII characters
    mov ebx, 10         ; Store the base 10 divisor
    xor ecx, ecx        ; Clear ecx for storing character count
    mov edi, buffer + 19 ; Load address of the end of buffer for storing ASCII result
    mov byte [edi], 0   ; Null-terminate the string

convert_loop:
    xor edx, edx        ; Clear edx for division
    div ebx             ; Divide eax by 10
    add dl, '0'         ; Convert remainder to ASCII character
    dec edi             ; Move to the previous position in buffer
    mov [edi], dl       ; Store ASCII character in buffer
    inc ecx             ; Increment character count
    test eax, eax       ; Check if quotient is zero
    jnz convert_loop    ; If not, continue conversion

    ; Prepare for printing
    mov edx, ecx        ; Load character count for sys_write
    mov ecx, edi        ; Load address of the start of the ASCII result
    mov ebx, 1          ; File descriptor 1: STDOUT
    mov eax, 4          ; System call number for sys_write
    int 80h             ; Call kernel

    ; Exit the program
    mov eax, 1          ; The system call for exit (sys_exit)
    xor ebx, ebx        ; Exit with return code of 0 (no error)
    int 80h             ; Call kernel

section .bss
    buffer resb 20      ; Buffer to store ASCII result
```

Execute

Upload Files | Input | Output | JDroid

38

New Project
My Projects
Execute History
Collaborate
Save
Save As
Editable Share
Instant Share (No Login/Save required)
Copy to Clipboard
Open (from local file)
Save (to local file)
Pretty Print
IDE Settings
How To/FAQ

```
section .text
    global _start

_start:
    ; Multiply 95 by 57
    mov eax, 95           ; Move the value 95 into eax
    mov ebx, 57           ; Move the value 57 into ebx
    imul eax, ebx         ; Multiply eax by ebx, result stored in eax

    ; Convert the result to ASCII characters
    mov ebx, 10           ; Store the base 10 divisor
    xor ecx, ecx          ; Clear ecx for storing character count
    mov edi, buffer + 19  ; Load address of the end of buffer for storing ASCII result
    mov byte [edi], 0     ; Null-terminate the string

convert_loop:
    xor edx, edx          ; Clear edx for division
    div ebx               ; Divide eax by 10, quotient in eax, remainder in edx
    add dl, '0'           ; Convert remainder to ASCII character
    dec edi               ; Move to the previous position in buffer
    mov [edi], dl         ; Store ASCII character in buffer
    inc ecx               ; Increment character count
    test eax, eax         ; Check if quotient is zero
    jnz convert_loop      ; If not, continue conversion

    ; Prepare for printing
    mov edx, ecx          ; Load character count for sys_write
    mov ecx, edi          ; Load address of the start of the ASCII result
    mov ebx, 1            ; File descriptor 1: STDOUT
    mov eax, 4            ; System call number for sys_write
    int 80h               ; Call kernel

    ; Exit the program
    mov eax, 1            ; The system call for exit (sys_exit)
    xor ebx, ebx          ; Exit with return code of 0 (no error)
    int 80h               ; Call kernel

section .bss
    buffer resb 20        ; Buffer to store ASCII result
```

Execute

Upload Files | Input | Output | JDroid

5415

New Project
My Projects
Execute History
Collaborate
Save
Save As
Editable Share
Instant Share (No Login/Save required)
Copy to Clipboard
Open (from local file)
Save (to local file)
Pretty Print
IDE Settings
How To/FAQ

```
section .text
    global _start
;(note: this code only shows integers without decimals)
_start:
    ; Divide 95 by 57
    mov eax, 95             ; Move the dividend 95 into eax
    mov ebx, 57             ; Move the divisor 57 into ebx
    xor edx, edx            ; Clear edx for high-order bits of the dividend
    div ebx                 ; Divide eax by ebx, quotient in eax, remainder in edx

    ; Convert the quotient to ASCII characters
    mov ebx, 10             ; Store the base 10 divisor
    xor ecx, ecx            ; Clear ecx for storing character count
    mov edi, buffer + 19    ; Load address of the end of buffer for storing ASCII result
    mov byte [edi], 0       ; Null-terminate the string

convert_loop:
    xor edx, edx            ; Clear edx for division
    div ebx                 ; Divide eax by 10, quotient in eax, remainder in edx
    add dl, '0'             ; Convert remainder to ASCII character
    dec edi                 ; Move to the previous position in buffer
    mov [edi], dl           ; Store ASCII character in buffer
    inc ecx                 ; Increment character count
    test eax, eax           ; Check if quotient is zero
    jnz convert_loop        ; If not, continue conversion

    ; Prepare for printing
    mov edx, ecx            ; Load character count for sys_write
    mov ecx, edi            ; Load address of the start of the ASCII result
    mov ebx, 1              ; File descriptor 1: STDOUT
    mov eax, 4              ; System call number for sys_write
    int 80h                 ; Call kernel

    ; Exit the program
    mov eax, 1              ; The system call for exit (sys_exit)
    xor ebx, ebx            ; Exit with return code of 0 (no error)
    int 80h                 ; Call kernel

section .bss
    buffer resb 20          ; Buffer to store ASCII result
```

Execute

Upload Files | Input | Output | JDroid

```
1
```

- New Project
- My Projects
- Execute History
- Collaborate
- Save
- Save As
- Editable Share
- Instant Share (No Login/Save required)
- Copy to Clipboard
- Open (from local file)
- Save (to local file)
- Pretty Print
- IDE Settings
- How To/FAQ

```
section .data
    msg db "Original String: "
    len equ $-msg   ; Calculates msg string's length

    msg2 db  0ah ,"Reversed string: " ; Adds newline before "Reversed String: "
    lenMsg2 equ $ - msg2    ; Computes length of msg2 string

    msg3 db 0ah, "Uppercase String: "
    lenMsg3 equ $-msg3    ; Determines length of msg3 string

    firstString db "team"
    lenFirst equ $-firstString   ; Measures length of firstString

    SecondString db "mate"
    lenSecond equ $-SecondString ; Measures length of SecondString

    result db 50    ; Allocates 50 bytes for result buffer


section .text
    global _start

    ; Writes the second message to output
    writeSecondMsg:
        mov ecx, msg2
        mov edx, lenMsg2
        mov eax, 4
        mov ebx, 1
        int 80h
        ret

    ; Outputs the first message
    writeFirstMsg:
        mov ecx, msg
        mov edx, len
        mov eax, 4
        mov ebx, 1
        int 80h
        ret

    ; Displays the third message
    writeThirdMsg:
        mov ecx, msg3
        mov edx, lenMsg3
        mov eax, 4
        mov ebx, 1
        int 80h
        ret

    ; Exits the program cleanly
    exit_program:
        mov eax, 1
        xor ebx, ebx    ; Resets ebx to 0
        int 80h

    ; Copies characters from source to destination until count reaches zero
    copy_loop:
        mov al, [esi]       ; Retrieves character from source
        mov [edi], al       ; Stores character in destination
        inc esi             ; Advances source pointer
        inc edi             ; Advances destination pointer
        dec ecx             ; Reduces loop count
        jnz copy_loop       ; Continues if count is not zero
    ret

    ; Sends the result string to output
    print_result:
        mov ecx, result
        mov edx, lenFirst + lenSecond  ; Sets output length
        mov eax, 4
        mov ebx, 1
        int 80h
        ret

    ; In-place string reversal
    reverse_string_loop:
        mov al, [esi]       ; Gets character from start of string
        mov ah, [edi]       ; Gets character from end of string
        mov [esi], ah       ; Swaps end character to start
        mov [edi], al       ; Swaps start character to end
        inc esi             ; Moves start pointer forward
        dec edi             ; Moves end pointer backward
        dec ecx             ; Decreases count
        jnz reverse_string_loop

        ret

_start:
    call writeFirstMsg

    ; Prepares to copy firstString into result
    mov esi, firstString
    mov edi, result
    mov ecx, lenFirst     ; Loop count set to firstString's length
    call copy_loop

    ; Appends SecondString to result after firstString
    mov esi, SecondString
    mov edi, result
    mov ecx, lenFirst     ; Moves edi to end of firstString in result
    add edi, ecx
    mov ecx, lenSecond    ; Loop count set to SecondString's length
    call copy_loop

    call print_result  ; Outputs concatenated string

    call writeSecondMsg
```

```
    ; Sets up for reversing the concatenated string
    mov esi, result
    mov edi, result
    dec edi              ; Adjusts edi to point at string's end
    mov ecx, lenFirst + lenSecond  ; Count set to concatenated string length
    call reverse_string_loop
    call print_result

    call writeThirdMsg

    ; Converts concatenated and reversed string to uppercase
    mov esi, result
    convert_to_uppercase:
        mov al, [esi]   ; Loads current character
        cmp al, 0       ; Checks for end of string
        je convert_done  ; Ends loop if end of string found

        ; Checks if character is lowercase and converts if true
        cmp al, 'a'
        jb not_lowercase
        cmp al, 'z'
        ja not_lowercase

        ; Converts lowercase to uppercase by adjusting ASCII value
        sub al, 32

not_lowercase:
        mov [esi], al   ; Saves converted character

        inc esi         ; Moves to next character
        jmp convert_to_uppercase  ; Repeats for next character

convert_done:

    call print_result

    call exit_program
```

Upload Files | Input | Output | JDroid

```
Original String: teammate
Reversed string: etammaet
Uppercase String: ETAMMAET
```

New Project
My Projects
Execute History
Collaborate
Save
Save As
Editable Share
Instant Share (No Login/Save required)
Copy to Clipboard
Open (from local file)
Save (to local file)
Pretty Print
IDE Settings
How To/FAQ

```
section .data
    buffer: times 10 db 0    ; Increased buffer size to accommodate "TeamMate"
    team: db "Team"      ; Define the string "Team"
    mate: db "Mate"  ; Define the string "Mate"

section .text
    global _start

_start:
    mov eax, [team]           ; Move the address of the string "Team" into the eax register
    mov [buffer], eax         ; Move the content of eax (the address of "Team") into the beginning of the buffer

    mov eax, [mate]         ; Move the address of the string "Mate" into the eax register
    mov [buffer+4], eax     ; Move the content of eax (the address of "Mate") into the buffer starting at offset 6

    ; Prepare for printing the concatenated string
    mov eax, 4                ; System call number for sys_write
    mov ebx, 1                ; File descriptor 1 (stdout)
    mov ecx, buffer           ; Pointer to the concatenated string in buffer
    mov edx, 8              ; Length of the concatenated string (6 bytes for "Team" + 8 bytes for "Mate")
    int 0x80                  ; Invoke syscall

    ; Exit
    mov eax, 1                ; System call number for sys_exit
    xor ebx, ebx              ; Exit code 0
    int 0x80                  ; Invoke syscall
```

Execute ▶

Upload Files | Input | Output | JDroid

```
TeamMate
```

Generated Files

New Project
My Projects
Execute History
Collaborate
Save
Save As
Editable Share
Instant Share (No Login/Save required)
Copy to Clipboard
Open (from local file)
Save (to local file)
Pretty Print
IDE Settings
How To/FAQ

```
section .text
    global _start

_start:
    ; Add 95 and 57
    mov eax, 95       ; Move the value 95 into eax
    add eax, 57     ; Add 57 to eax

    ; Convert the result to ASCII characters
    mov ebx, 10         ; Store the base 10 divisor
    xor ecx, ecx        ; Clear ecx for storing character count
    mov edi, buffer + 19 ; Load address of the end of buffer for storing ASCII result
    mov byte [edi], 0   ; Null-terminate the string

convert_loop:
    xor edx, edx        ; Clear edx for division
    div ebx             ; Divide eax by 10
    add dl, '0'         ; Convert remainder to ASCII character
    dec edi             ; Move to the previous position in buffer
    mov [edi], dl       ; Store ASCII character in buffer
    inc ecx             ; Increment character count
    test eax, eax       ; Check if quotient is zero
    jnz convert_loop    ; If not, continue conversion

    ; Prepare for printing
    mov edx, ecx        ; Load character count for sys_write
    mov ecx, edi        ; Load address of the start of the ASCII result
    mov ebx, 1          ; File descriptor 1: STDOUT
    mov eax, 4          ; System call number for sys_write
    int 80h             ; Call kernel

    ; Exit the program
    mov eax, 1          ; The system call for exit (sys_exit)
    xor ebx, ebx        ; Exit with return code of 0 (no error)
    int 80h             ; Call kernel



section .bss
    buffer resb 20      ; Buffer to store ASCII result
```

Execute

Upload Files | Input | Output | JDroid

```
152
```