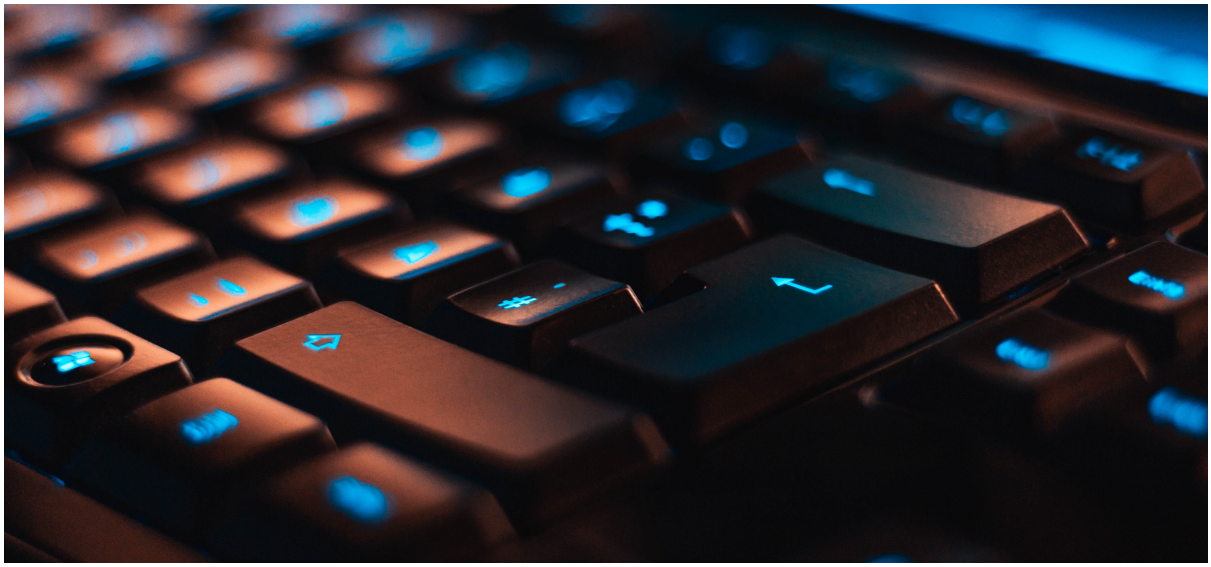


COA Lab Assignment 4

Roll No.: CS20B1016

Name: V Nagasai

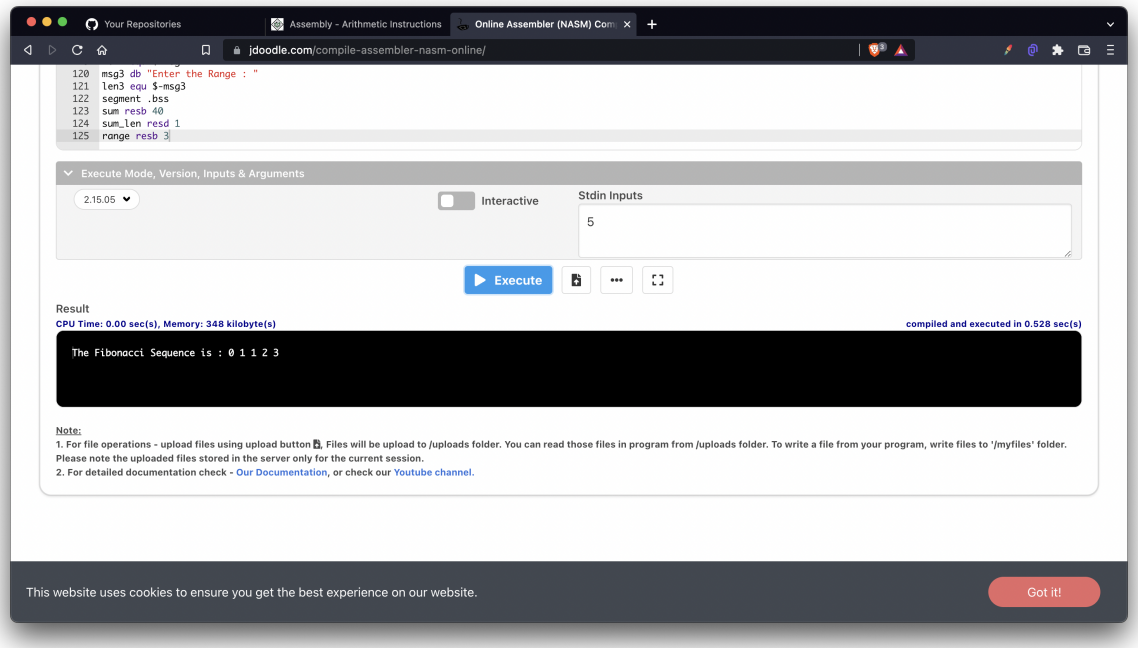
1. **Write a program to generate the Fibonacci series in ALP.**

Solution:

```
section .text
global _start
_start:
mov eax , 3
mov ebx , 0
mov ecx , range
mov edx , 3
int 0x80
; Printing the initial message
mov eax , 4
mov ebx , 1
mov ecx , msg1
mov edx , len1
int 0x80
; Printing the 1st number
```

```
mov eax , [x]
mov edi, sum
call int2str
sub edi, sum
mov [sum_len], edi
mov eax , 4
mov ebx , 1
mov ecx , sum
mov edx , [sum_len]
int 0x80
mov eax , 4
mov ebx , 1
mov ecx , msg2
mov edx , len2
int 0x80
; Printing the 2nd number
mov eax , [y]
mov edi, sum
call int2str
sub edi, sum
mov [sum_len], edi
mov eax , 4
mov ebx , 1
mov ecx , sum
mov edx , [sum_len]
int 0x80
mov eax , 4
mov ebx , 1
mov ecx , msg2
mov edx , len2
int 0x80
; The Actual program
mov edx, range ; our string
call str2int
mov ecx , eax
sub ecx , 2
fib_loop:
mov [range] , ecx
mov eax , [x]
mov ebx , [y]
add eax , ebx
mov ebx , [y]
mov [x] , ebx
mov [y] , eax
mov edi, sum
call int2str
sub edi, sum
mov [sum_len], edi
mov eax , 4
mov ebx , 1
mov ecx , sum
mov edx , [sum_len]
int 0x80
mov eax , 4
mov ebx , 1
mov ecx , msg2
mov edx , len2
int 0x80
```

```
mov ecx , [range]
dec ecx
jne fib_loop
mov eax, 1
int 0x80
str2int: ; Converts an string in EDX to a integer stored in eax.
xor eax, eax ; clear off anything in eax
.top:
movzx ecx, byte [edx] ; get each character of string in each iteration
inc edx
cmp ecx, '0' ; If you reach end of string, then you're done.
jb .done
cmp ecx, '9'
ja .done
sub ecx, '0' ; converting character to integer
imul eax, 10 ; multiply the result obtained till now by ten
add eax, ecx ; add in current digit
jmp .top ; loop until done.
.done:
ret
int2str: ; Converts an positive integer to a string
xor ecx, ecx
mov ebx, 10
.LL1:
xor edx, edx
div ebx
push dx
inc ecx
test eax, eax
jnz .LL1
.LL2:
pop dx
or dl, '0'
mov [edi], dl
inc edi
loop .LL2
mov byte [edi], 0
ret
section .data
x dd 0
y dd 1
;range dd 15
msg1 db "The Fibonacci Sequence is : "
len1 equ $-msg1
msg2 db " "
len2 equ $-msg2
msg3 db "Enter the Range : "
len3 equ $-msg3
segment .bss
sum resb 40
sum_len resd 1
range resb 3
```



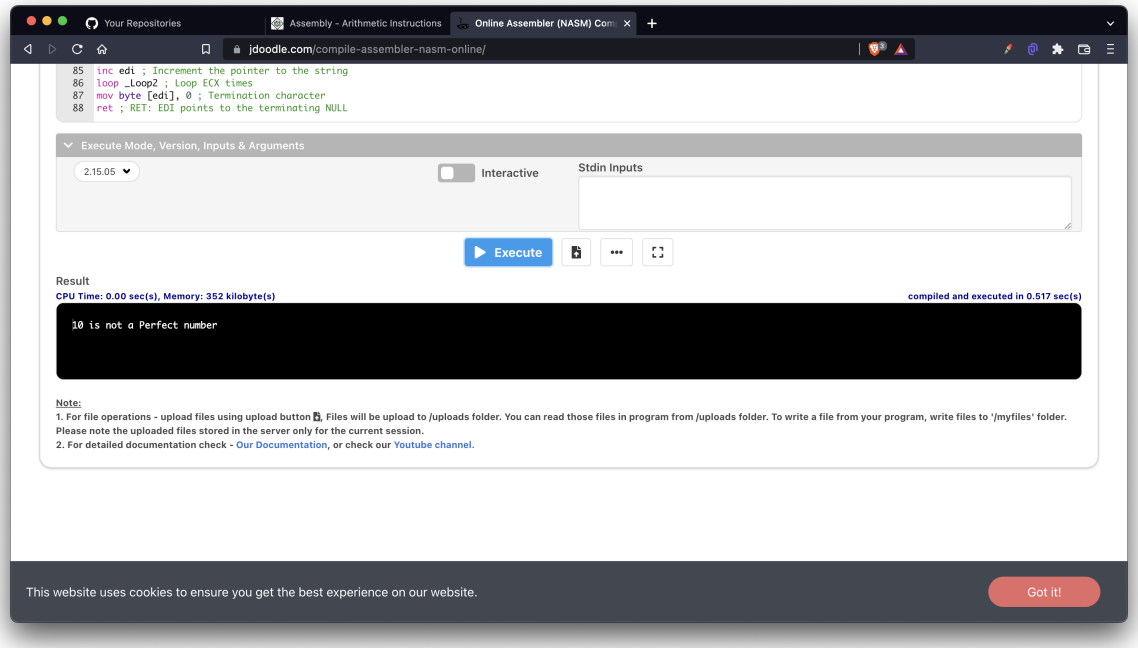
2. **Write an ALP to check if the given number is a Perfect number or not.**

Solution:

```
section .data
msg1 db " is a Perfect number"
len1 equ $- msg1
msg2 db " is not a Perfect number"
len2 equ $- msg2
num dd 6
section .bss
res resb 10
res_len resb 10
section .text
global _start ;must be declared for using gcc
_start: ;tell linker entry point
mov eax,[num]
mov ebx,2
mov ecx,1
mov edx,0
perfect_Num : ;To find the sum of the positive divisors
cmp eax,1
```

```
je print
cmp ebx,eax
je print
push eax
push edx
div ebx
cmp edx,0
je sum
end:
pop edx
pop eax
inc ebx
jmp perfect_Num
print:
push eax
push ecx
; Convert the number to a string
mov edi, res ; Argument: Address of the target string
call intTOstr ; Get the digits of EAX and store it as ASCII
sub edi, res ; getting length of the string
mov [res_len], edi
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
mov ecx,res
mov edx, [res_len]
int 0x80 ;call kernel
pop ecx
pop eax
cmp eax,1
je print2
cmp ecx,eax
je print1
print2:
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
mov ecx,msg2
mov edx, len2
int 0x80 ;call kernel
jmp exit
print1:
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
mov ecx,msg1
mov edx, len1
int 0x80 ;call kernel
exit:
mov eax, 1
mov ebx,0
int 80h
sum:
add ecx,ebx
jmp end
intTOstr: ; Converts an positive integer in EAX to a string pointed to by EDI
xor ecx, ecx
mov ebx, 10
_Loop1: ; First loop: Save the remainders
xor edx, edx ; Clear EDX for div
div ebx ; EDX:EAX/EBX -> EAX Remainder EDX
```

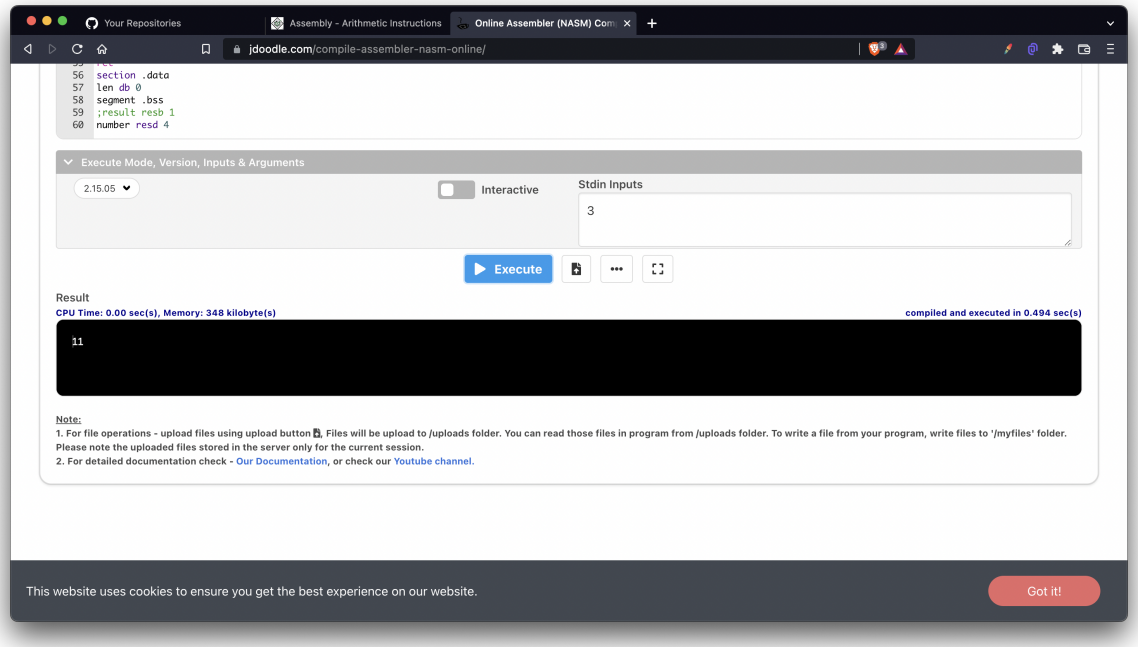
```
push dx ; Save remainder
inc ecx ; Increment push counter
test eax, eax ; Anything left to divide?
jnz _Loop1 ; Yes: loop once more
_Loop2: ; Second loop: Retrieve the remainders
pop dx ; In DL is the value
or dl, '0' ; To ASCII
mov [edi], dl ; Save it to the string
inc edi ; Increment the pointer to the string
loop _Loop2 ; Loop ECX times
mov byte [edi], 0 ; Termination character
ret ; RET: EDI points to the terminating NULL
```



3. Write an ALP program to convert a decimal number to binary.

```
section .text
global _start
_start:
```

```
mov eax , 3
mov ebx , 0
mov ecx , number
mov edx , 4
int 0x80
mov edx , number
call str2int
; Loop for implementing the iterative division to obtain remainders
div_loop:
inc dword[len] ; Keep track of word length of binary string
xor edx, edx
mov ecx, 2 ; put denominator in ecx
div ecx ; Divide to get numerator and remainder
push edx
cmp eax , 0 ; Check if the given number has reached zero or not => If yes
exit, else continue
jne div_loop
jmp _print
; Function to print the obtained string in reverse order
_print:
dec dword[len]
pop edx
push eax
mov eax , edx
add eax , 48
push eax
mov eax , esp
mov ecx, eax
mov edx, 1
mov ebx, 1
mov eax, 4
int 0x80
pop eax
pop eax
cmp dword [len], 0
jne _print
mov eax, 1
int 0x80
str2int: ; Converts an string in EDX to a integer stored in eax.
xor eax, eax ;clear off anything in eax
.top:
movzx ecx, byte [edx] ; get each character of string in each iteration
inc edx
cmp ecx, '0' ;If you reach end of string, then you're done.
jb .done
cmp ecx, '9'
ja .done
sub ecx, '0' ;converting character to integer
imul eax, 10 ;multiply the result obtained till now by ten
add eax, ecx ;add in current digit
jmp .top ;loop until done.
.done:
ret
section .data
len db 0
segment .bss
;result resb 1
number resd 4
```


Your Repositories
Assembly - Arithmetic Instructions
Online Assembler (NASM) Com
jdoodle.com/compile-assembler-nasm-online/
section .data
len db 0
segment .bss
;result resb 1
number resd 4
Execute Mode, Version, Inputs & Arguments
2.15.05
Interactive
Stdin Inputs
3
Execute
Result
CPU Time: 0.00 sec(s), Memory: 348 kilobyte(s)
compiled and executed in 0.494 sec(s)
11
Note:
1. For file operations - upload files using upload button. Files will be upload to /uploads folder. You can read those files in program from /uploads folder. To write a file from your program, write files to '/myfiles' folder. Please note the uploaded files stored in the server only for the current session.
2. For detailed documentation check - Our Documentation, or check our Youtube channel.
This website uses cookies to ensure you get the best experience on our website.
Got it!