Open

string.asm

```
section .data
    string1 db "Hello, World!", 0 ; Null-terminated string

section .bss
    output resb 4 ; Reserve 4 bytes for the length (up to 3 digits + null terminator)

section .text

global _start

_start:
    ; Initialize registers
    mov esi, string1  ; Move the address of string1 into ESI
    xor ecx, ecx      ; Clear ECX to use it as a counter

length_loop:
    lodsb             ; Load byte at DS:ESI into AL and increment ESI
    cmp al, 0         ; Check if the byte is the null terminator
    je done_length    ; If it is, jump to the done_length label
    inc ecx           ; Increment the length counter
    jmp length_loop   ; Repeat the loop

done_length:
    ; Store the length as a decimal number in the output buffer
    mov edi, output   ; Point EDI to the output buffer
    xor eax, eax      ; Clear EAX to use it for division
    xor ebx, ebx      ; Clear EBX

    ; Convert ECX (length) to ASCII digits
    mov eax, ecx      ; Move the length value into EAX
    mov ebx, 10       ; Set divisor to 10
print_digit:
    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 EDI backward
    mov [edi], dl     ; Store ASCII character in output buffer
    test eax, eax     ; Check if quotient is zero
    jnz print_digit   ; If not zero, repeat the loop

    ; Calculate the number of digits
    mov eax, output   ; Move base address of output buffer to EAX
    sub eax, edi      ; Calculate length of the number string
    mov edx, eax      ; Set EDX to length of the number string

    ; Write to stdout
    mov eax, 4        ; sys_write
    mov ebx, 1        ; file descriptor (stdout)
    mov ecx, edi      ; buffer to write from (current position of EDI)
    int 0x80

    ; Exit the program
    mov eax, 1        ; sys_exit
    xor ebx, ebx      ; status code 0
    int 0x80
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ gedit string.asm

┌──(dj㉿DJ)-[~]
└─$ nasm -f elf string.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 string.o -o string

┌──(dj㉿DJ)-[~]
└─$ ./string
13

┌──(dj㉿DJ)-[~]
└─$
```

Open hexbcd1.asm Save

```
section .data
    input_prompt db "Enter a 1-digit hexadecimal number: ", 0
    invalid_input db "Invalid input! Please enter a valid hex digit (0-9, A-F).", 10, 0
    result_prompt db "The BCD equivalent is: ", 0
    hex_input db 0            ; To store the input hex digit
    bcd_result db 0           ; To store the BCD result
    result_display db "BCD: 00", 10, 0 ; To display the result

section .text
global _start ; Entry point for the program

_start:
    ; Display input prompt
    mov eax, 4            ; syscall number for sys_write
    mov ebx, 1            ; file descriptor 1 (stdout)
    lea ecx, [input_prompt] ; address of input prompt
    mov edx, 34           ; length of input prompt
    int 0x80              ; invoke syscall

    ; Read user input
    mov eax, 3            ; syscall number for sys_read
    mov ebx, 0            ; file descriptor 0 (stdin)
    lea ecx, [hex_input]  ; address of input buffer
    mov edx, 1            ; number of bytes to read
    int 0x80              ; invoke syscall

    ; Convert ASCII to number
    sub byte [hex_input], '0'   ; Convert '0'-'9' to 0-9
    cmp byte [hex_input], 9
    jbe valid_hex         ; If less than or equal to '9', valid hex digit

    sub byte [hex_input], 7    ; Convert 'A'-'F' to 10-15
    cmp byte [hex_input], 15
    jbe valid_hex         ; If less than or equal to 'F', valid hex digit

invalid_hex:
    ; Handle invalid input
    mov eax, 4            ; syscall number for sys_write
    mov ebx, 1            ; file descriptor 1 (stdout)
    lea ecx, [invalid_input] ; address of invalid input message
    mov edx, 58           ; length of invalid input message
    int 0x80              ; invoke syscall

    ; Exit the program (after displaying error)
    mov eax, 1            ; syscall number for sys_exit
    xor ebx, ebx          ; Return code 0 (successful exit)
    int 0x80              ; invoke syscall

valid_hex:
    ; Convert hex to BCD using `aam` instruction
    mov al, [hex_input]   ; Load input hex digit
    mov ah, 0             ; Clear high byte of AX
    aam                   ; Adjust AX after multiply by 10 (AL -> ones, AH -> tens)
    mov [bcd_result], ax  ; Store BCD result in `bcd_result`

    ; Display result prompt
    mov eax, 4            ; syscall number for sys_write
    mov ebx, 1            ; file descriptor 1 (stdout)
    lea ecx, [result_prompt] ; address of result prompt
    mov edx, 23           ; length of result prompt
    int 0x80              ; invoke syscall

    ; Display result
    mov ax, [bcd_result]  ; Load BCD result into AX
    add al, '0'           ; Convert ones digit to ASCII
    add ah, '0'           ; Convert tens digit to ASCII
    mov [result_display + 4], ah  ; Store tens digit
    mov [result_display + 5], al  ; Store ones digit

    ; Display result (BCD value)
    mov eax, 4            ; syscall number for sys_write
    mov ebx, 1            ; file descriptor 1 (stdout)
    lea ecx, [result_display] ; address of result to display
    mov edx, 7            ; length of result display
    int 0x80              ; invoke syscall

    ; Exit the program (successful termination)
    mov eax, 1            ; syscall number for sys_exit
    xor ebx, ebx          ; Return code 0 (successful exit)
    int 0x80              ; invoke syscall
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf hexbcd1.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 hexbcd1.o -o hexbcd1

┌──(dj㉿DJ)-[~]
└─$ ./hexbcd1
Enter a 1-digit hexadecimal numberB
The BCD equivalent is: CD:11

┌──(dj㉿DJ)-[~]
└─$
```

Open | sum_input.asm | Save

```
1 global _start
2
3 section .data
4         first_print dw "Enter your value: ",10;
5         len_first_print equ $- first_print
6
7         second_print dw "Enter second your value: ",10;
8         len_second_print equ $- second_print
9
10        third_print dw "Your value is: ",10;
11        len_third_print equ $- third_print
12
13
14 section .bss
15        num1 resb 2
16        num2 resb 2
17        num3 resb 2
18
19 section .text
20
21 %macro in_out 3
22        mov eax,%1
23        mov ebx,1
24        mov ecx,%2
25        mov edx,%3
26
27        int 80h
28 %endmacro
29
30 _start:
31        in_out 4,first_print,len_first_print
32        in_out 3,num1,2
33        in_out 4,second_print,len_second_print
34        in_out 3,num2,2
35
36        mov ax,[num1]
37        sub ax,'0'
38        mov bx,[num2]
39        sub bx,'0'
40
41        add ax,bx
42        add ax,'0'
43        mov [num3],ax
44
45        in_out 4,third_print,len_third_print
46        in_out 4,num3,2
47
48        mov eax,1
49        int 80h
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf sum_input.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 sum_input.o -o sum_input

┌──(dj㉿DJ)-[~]
└─$ ./sum_input
Enter your value:
3
Enter second your value:
5
Your value is:
8

┌──(dj㉿DJ)-[~]
└─$
```

Open | input_data.asm | Save

```
1 global _start
2
3 section .data
4         first_print dw "Enter your value: ",10;
5         len_first_print equ $- first_print
6
7         second_print dw "Your entered value is: ",10;
8         len_second_print equ $- second_print
9
10 section .bss
11         num resb 2
12
13 section .text
14
15 %macro in_out 3
16         mov eax,%1
17         mov ebx,1
18         mov ecx,%2
19         mov edx,%3
20
21         int 80h
22 %endmacro
23
24 _start:
25         in_out 4,first_print,len_first_print
26         in_out 3,num,2
27         in_out 4,second_print,len_second_print
28         in_out 4,num,2
29
30         mov eax,1
31         int 80h
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf input_data.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 input_data.o -o input_data

┌──(dj㉿DJ)-[~]
└─$ ./input_data
Enter your value:
6
Your entered value is:
6

┌──(dj㉿DJ)-[~]
└─$
```

Open | macros.asm | Save

```
1 global _start
2
3 section .data
4         name db "Name: Vijay", 10; '10' is a new line character
5         name_len equ $-name
6
7         surname db "Surname: Bhilare",10
8         surname_len equ $-surname
9
10        location db "Location: Pune", 10
11        location_len equ $-location
12
13        branch db "Branch: CSE", 10
14        branch_len equ $-branch
15
16        univ db "University: MITADT", 10
17        univ_len equ $-univ
18
19 section .text
20
21 %macro print 2 ; Here 'print' is name of macro and '2' is the register that it going to change
22        mov eax,4
23        mov ebx,1
24        mov ecx,%1 ; This is the first changing register. That's why we have written %1
25        mov edx,%2 ; This is the second changing register. That's why we have written %2
26        int 80h
27
28 %endmacro
29
30 _start:
31
32        print name, name_len
33        print surname, surname_len
34        print location, location_len
35        print branch, branch_len
36        print univ, univ_len
37        int 80h;
38
39 mov eax,1
40 xor ebx, ebx
41 int 80h
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf macros.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 macros.o -o macros

┌──(dj㉿DJ)-[~]
└─$ ./macros
Name: Vijay
Surname: Bhilare
Location: Pune
Branch: CSE
University: MITADT

┌──(dj㉿DJ)-[~]
└─$
```

Open

sub.asm

Save

```
1 global _start
2
3 section .data
4
5         diff dw 0
6
7 section .text
8
9 _start:
10
11        mov ax, 8
12        mov bx,7
13        sub ax,bx
14        add ax, '0'
15        mov [diff],ax
16
17        mov eax, 4
18        mov ebx,1
19        mov ecx, diff
20        mov edx,2
21        int 80h
22
23        mov eax,1
24        int 80h
25
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf sub.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 sub.o -o sub

┌──(dj㉿DJ)-[~]
└─$ ./sub
1

┌──(dj㉿DJ)-[~]
└─$
```

Open sum.asm Save

```
1 global _start
2
3 section .data
4 sum dw 0
5
6 section .text
7 _start:
8
9         mov ax, 5 ;
10        mov bx, 4 ;
11        add ax,bx ;
12        add ax,'0';
13        mov [sum],ax;
14
15        mov eax, 4;
16        mov ebx,1;
17        mov ecx, sum;
18        mov edx,2;
19        int 80h;
20
21        mov eax,1;
22        int 80h;
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf sum.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 sum.o -o sum

┌──(dj㉿DJ)-[~]
└─$ ./sum
9

┌──(dj㉿DJ)-[~]
└─$
```

Open hello.asm Save

```
1 global _start
2 section .data
3         first db "Hello", 10
4         len_first equ $-first
5         last db "World",10
6         len_last equ $-last
7
8 section .text
9         mov eax, 4;
10        mov ebx, 1;
11        mov ecx, first;
12        mov edx, len_first;
13        int 80h;
14
15        xor ebx,ebx;
16
17        mov eax, 4;
18        mov ebx, 1;
19        mov ecx, last;
20        mov edx, len_last;
21        int 80h;
22
23        xor ebx, ebx;
24        mov eax, 1;
25        int 80h;
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf hello.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 hello.o -o hello
ld: warning: cannot find entry symbol _start; defaulting to 08049000

┌──(dj㉿DJ)-[~]
└─$ ./hello
Hello
World

┌──(dj㉿DJ)-[~]
└─$
```

Open

string_concat.asm

Save

```
section .data
    string1 db "Hello, "
    string2 db "World!", 0
    output_buffer times 20 db 0 ; Reserve 20 bytes for the concatenated result

section .text
    global _start

_start:
    ; Load addresses of string1, string2, and output_buffer
    lea esi, [string1]
    lea edi, [output_buffer]

copy_string1:
    lodsb        ; Load byte at esi into al, increment esi
    stosb        ; Store byte from al into edi, increment edi
    cmp al, 0    ; Check if we reached end of string1
    jne copy_string1  ; Repeat if not at the end

copy_string2:
    lea esi, [string2]  ; Load address of string2 into esi

concat_loop:
    lodsb        ; Load byte at esi into al, increment esi
    stosb        ; Store byte from al into edi, increment edi
    cmp al, 0    ; Check if we reached end of string2
    jne concat_loop  ; Repeat if not at the end

    ; Prepare to print the result
    mov eax, 4           ; Syscall number for sys_write
    mov ebx, 1           ; File descriptor (stdout)
    lea ecx, [output_buffer]  ; Pointer to the output buffer
    sub edi, output_buffer  ; Store length of concatenated string in edx
    mov edx, edi

    ; Invoke syscall
    int 0x80

    ; Exit program
    mov eax, 1           ; Syscall number for sys_exit
    xor ebx, ebx         ; Exit code 0
    int 0x80
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf string_concat.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 string_concat.o -o string_concat

┌──(dj㉿DJ)-[~]
└─$ ./string_concat
Hello, World!World!

┌──(dj㉿DJ)-[~]
└─$
```

Open | input_string.asm | Save

```
section .data
    prompt db 'Enter a string:', 0x0a, 0x00 ; prompt string with newline
    input_buffer times 100 db 0 ; 100 byte buffer for input

section .text
    global _start

_start:
    ; Print the prompt
    mov eax, 4 ; sys_write system call
    mov ebx, 1 ; file descriptor 1 (stdout)
    mov ecx, prompt ; pointer to prompt string
    mov edx, 16 ; length of prompt string
    int 0x80 ; make system call

    ; Read user input
    mov eax, 3 ; sys_read system call
    mov ebx, 0 ; file descriptor 0 (stdin)
    mov ecx, input_buffer ; pointer to input buffer
    mov edx, 100 ; maximum length of input
    int 0x80 ; make system call

    ; Print the input string
    mov eax, 4 ; sys_write system call
    mov ebx, 1 ; file descriptor 1 (stdout)
    mov ecx, input_buffer ; pointer to input buffer
    int 0x80 ; make system call

    ; Exit the program
    mov eax, 1 ; sys_exit system call
    xor ebx, ebx ; exit status 0
    int 0x80 ; make system call
```

dj@DJ: ~

File Actions Edit View Help

```
┌──(dj㉿DJ)-[~]
└─$ nasm -f elf input_string.asm

┌──(dj㉿DJ)-[~]
└─$ ld -m elf_i386 input_string.o -o input_string

┌──(dj㉿DJ)-[~]
└─$ ./input_string
Enter a string:
I am Vijay
I am Vijay

┌──(dj㉿DJ)-[~]
└─$
```

```
1 global _start
2
3 section .data
4
5         dividend db 26 ; Dividend (example: 1Ah)
6         divisor db 5 ; Divisor (example: 05h)
7         quotient db 0 ; Memory location to store the quotient
8         remainder db 0 ; Memory location to store the remainder
9
10 section .text
11
12 _start:
13         ; Load the 8-bit numbers
14         mov al, [dividend] ; Load dividend into AL
15         mov bl, [divisor] ; Load divisor into BL
16
17         ; Clear AH register
18         xor ah, ah ; Clear AH to ensure AH:AL is the dividend
19
20         ; Divide the numbers
21         div bl ; Divide AX by BL, quotient in AL, remainder in AH
22
23         ; Store the result
24         mov [quotient], al ; Store the quotient in memory
25         mov [remainder], ah ; Store the remainder in memory
26
27         ; Exit program using Linux system call
28         mov eax, 1 ; syscall number for exit (sys_exit)
29         xor ebx, ebx ; exit code 0
30         int 0x80 ; invoke syscall
```

```
File  Actions  Edit  View  Help
─Register group: general─
eax            0x64               100
edx            0x0                0
esp            0xffffd0c0         0xffffd0c0
esi            0x0                0
eip            0x804900d          0x804900d <_start+13>
cs             0x23               35
ds             0x2b               43
fs             0x0                0
ecx            0x0                0
ebx            0x2                2
ebp            0x0                0x0
edi            0x0                0
eflags         0x202              [ IF ]
ss             0x2b               43
es             0x2b               43
gs             0x0                0

B+  0x8049000 <_start>      mov    al,ds:0x804a000
    0x8049005 <_start+5>    mov    bl,BYTE PTR ds:0x804a001
    0x804900b <_start+11>   mul    bl
  > 0x804900d <_start+13>   mov    ds:0x804a002,ax
    0x8049013 <_start+19>   mov    eax,0x1
    0x8049018 <_start+24>   xor    ebx,ebx
    0x804901a <_start+26>   int    0x80
    0x804901c               add    BYTE PTR [eax],al
    0x804901e               add    BYTE PTR [eax],al
    0x8049020               add    BYTE PTR [eax],al
    0x8049022               add    BYTE PTR [eax],al
    0x8049024               add    BYTE PTR [eax],al
    0x8049026               add    BYTE PTR [eax],al
native process 24424 In: _start                                  L??   PC: 0x804900d
(gdb) layout regs
(gdb) nexti
0x08049005 in _start ()
(gdb) nexti
0x0804900b in _start ()
(gdb) nexti
0x0804900d in _start ()
(gdb)
```

```
1 global _start
2
3 section .data
4         name db "Name: Chirag", 10; '10'  is a new line character
5         name_len equ $-name
6
7         surname db "Surname: Gehlot",10
8         surname_len equ $-surname
9
10        location db "Location: Pune",10
11        location_len equ $-location
12
13        branch db "Branch: CSE",10
14        branch_len equ $-branch
15
16        univ db "University: MITADT",10
17        univ_len equ $-univ
18
19
20 section .text
21
22 %macro print 2 ; Here 'print' is name of macro and '2' is the register that it going to change
23        mov eax,4
24        mov ebx,1
25        mov ecx,%1 ; This is the first changing register. That's why we have written %1
26        mov edx,%2 ; This is the second changing register. That's why we have written %2
27        int 80h
28
29 %endmacro
30
31 _start:
32        print name, name_len
33        print surname, surname_len
34        print location, location_len
35        print branch, branch_len
36        print univ,univ_len
37        int 80h;
38
39
40        mov eax,1
41        xor ebx,ebx
42        int 80h
```

```
File  Actions  Edit  View  Help
Register group: general
eax            0×5                5                  ecx            0×0                0
edx            0×0                0                  ebx            0×5                5
esp            0×ffffd0c0         0×ffffd0c0         ebp            0×0                0×0
esi            0×0                0                  edi            0×0                0
eip            0×804900f          0×804900f <_start+15>  eflags     0×246              [ PF ZF IF ]
cs             0×23               35                 ss             0×2b               43
ds             0×2b               43                 es             0×2b               43
fs             0×0                0                  gs             0×0                0

B+  0×8049000 <_start>      mov    al,ds:0×804a000
    0×8049005 <_start+5>    mov    bl,BYTE PTR ds:0×804a001
    0×804900b <_start+11>   xor    ah,ah
    0×804900d <_start+13>   div    bl
  > 0×804900f <_start+15>   mov    ds:0×804a002,al
    0×8049014 <_start+20>   mov    BYTE PTR ds:0×804a003,ah
    0×804901a <_start+26>   mov    eax,0×1
    0×804901f <_start+31>   xor    ebx,ebx
    0×8049021 <_start+33>   int    0×80
    0×8049023               add    BYTE PTR [eax],al
    0×8049025               add    BYTE PTR [eax],al
    0×8049027               add    BYTE PTR [eax],al
    0×8049029               add    BYTE PTR [eax],al
native process 22603 In: _start                                   L??   PC: 0×804900f
(gdb) layout regs
(gdb) nexti
0×08049005 in _start ()
(gdb) nexti
0×0804900b in _start ()
(gdb) nexti
0×0804900d in _start ()
(gdb) nexti
0×0804900f in _start ()
(gdb)
```