%macro io 4

mov rax,%1 ; System call number (1 for write, 0 for read) mov rdi,%2 ; File descriptor (1 for stdout, 0 for stdin) mov rsi,%3 ; Buffer address

mov rdx,%4 ; Buffer size

syscall ; Invoke system call

%endmacro

%macro exit 0

mov rax,60 ; System call number for exit

mov rdi,0 ; Exit status code (0 for success)

syscall ; Invoke system call

%endmacro

section .data

msg db "Breaking out",10 msglen equ $-msg

msg1 db "Write an x86/64 ALP to perform arithmetic operations on 64\ bit hexadecimal numbers",10,"Name:yakein",10,"Roll no:7262",10,"Date Of

Performance:10/02/2025",10 msg1len equ $-msg1

menu db "0. Exit",10,"1. Add",10,"2. Subtract",10,"3. Multiply"\

,10,"4. Divide",10

menulen equ $-menu

input1 db "Enter first number: ", 10 input1len equ $-input1

input2 db "Enter second number: ", 10 input2len equ $-input2

add\_res\_msg db "The sum is: " add\_res\_msg\_len equ $-add\_res\_msg carry\_msg db "Carry is: " carry\_msg\_len equ $-carry\_msg

diff\_res\_msg db "The difference is: " diff\_res\_msg\_len equ $-diff\_res\_msg borrow\_msg db "Borrow is: " borrow\_msg\_len equ $-borrow\_msg

prod\_res\_msg db "The product is: " prod\_res\_msg\_len equ $-prod\_res\_msg

rem\_msg db "The remainder is: " rem\_msg\_len equ $-rem\_msg

quot\_msg db "The result is: " quot\_msg\_len equ $-quot\_msg

newline db 10 section .bss

choice resb 2

asciinum resb 17

num1 resq 1

num2 resq 1

carry\_value resb 1

section .text global \_start

\_start:

io 1,1,msg1,msg1len ;print msg1

io 1,1, menu,menulen ;print menu

io 0,0,choice,2 ;ask user about operation to perform

cmp byte[choice],"0" ;if user chose 0 -> exit je close

io 1,1, input1,input1len ;prompt for first number

io 0,0, asciinum,17 ;take first number as input

;and store in asciinum

call ascii\_hex64 ;convert ascii input to hex value

mov qword[num1],rbx ;store input number in num1

io 1,1,newline,1 ;print a newline

io 1,1, input2,input2len ;prompt for second number

io 0,0, asciinum,17 ;take second number as input and

;store it in asciinum

call ascii\_hex64 ;convert ascii input into hex value

mov qword[num2],rbx ;store input number in num2

cmp byte[choice],"1" ;if user chose 1 from menu -> sum je sum

cmp byte[choice],"2" ;if user chose 2 from menu-> difference je difference

cmp byte[choice],"3" ;if user chose 3 from menu-> product je product

cmp byte[choice],"4" ;if user chose 4 from menu-> division je division

close: ;label for exiting the program io 1,1,newline,1

exit

sum:

mov rbx,qword[num1] ;move first number into rbx; mov rax, qword[num2] ;move second number into rax; add rbx,rax ;rbx = rbx + rax

mov byte[carry\_value],"0" ;take 0 as default carry value

jnc result1 ;if carry flag = 0 -> print result

mov byte[carry\_value],"1" ;else set carry as 1

result1: ;print the result io 1,1,newline,1

io 1,1,add\_res\_msg,add\_res\_msg\_len call hex\_ascii64

io 1,1,newline,1

io 1,1,carry\_msg,carry\_msg\_len io 1,1,carry\_value,1

jmp close

difference:

mov rbx, qword[num1] ;move first number into rbx

mov rax, qword[num2] ;move second number into rax

sub rbx,rax ;rbx = rbx - rax

mov byte[carry\_value],"0" ;take 0 as default borrow value jnc result2 ;if carry flag = 1 -> print result mov byte[carry\_value],"1" ;else set borrow as 1

result2: ;print result

io 1,1,newline,1

io 1,1,diff\_res\_msg,diff\_res\_msg\_len call hex\_ascii64

io 1,1,newline,1

io 1,1,borrow\_msg,borrow\_msg\_len io 1,1,carry\_value,1

jmp close

product:

mov rax,qword[num1] ;move first number

mul qword[num2] ;multiply rax value by num2

;result = rdx:rax

push rax ;store rax in stack

push rdx ;store rdx in stack

io 1,1,newline,1

io 1,1,prod\_res\_msg,prod\_res\_msg\_len

pop rbx ;pop rdx value and store in rbx

call hex\_ascii64 ;print higher 64 bits of result

pop rbx ;pop rax value and store in rbx

call hex\_ascii64 ;print lower 64 bits of result jmp close

division:

mov rdx,qword[num1] ;move dividend in rdx

mov eax,edx ;move lower 32 bits of dividend in eax

shr rdx,32 ;shift higher 32 bits in eax

mov ecx,dword[num2] ;take lower 32 bits of num2 as divisor

;and store it in ecx

div ecx ;divide by ecx

;result format:

;quotient -> edx

;remainder -> eax

push rdx ;push quotient value to stack

push rax ;push remainder value to stack io 1,1,newline,1

io 1,1,quot\_msg,quot\_msg\_len pop rbx

call hex\_ascii64 io 1,1,newline,1

io 1,1,rem\_msg,rem\_msg\_len pop rbx

call hex\_ascii64 jmp close

; Function to convert ASCII to hexadecimal ascii\_hex64:

mov rsi, asciinum ; Address of input buffer

mov rbx, 0 ; Clear rbx to store the number

mov rcx, 16 ; Loop for 16 characters next3:

mov al, [rsi] ; Load a character cmp al,10

je break1

rol rbx, 4 ; Make space for the next nibble cmp al, '9'

jbe sub30h ; Convert '0'-'9'

sub al, 7h ; Adjust 'A'-'F' sub30h:

sub al, 30h ; Convert ASCII to numeric value

add bl, al ; Add to rbx

inc rsi ; Move to next character loop next3

ret break1:

ret

; Function to convert hexadecimal to ASCII and io 1,1, hex\_ascii64:

mov rsi, asciinum ; Address of output buffer

mov rcx, 16 ; Loop for 16 characters next4:

rol rbx, 4 ; Get the most significant nibble

mov al, bl ; Isolate the nibble

and al, 0Fh ; Mask the lower 4 bits cmp al, 9

jbe add30h ; Convert to '0'-'9'

add al, 7h ; Convert to 'A'-'F' add30h:

add al, 30h ; Convert to ASCII

mov [rsi], al ; Store in output buffer

inc rsi ; Move to next character loop next4

io 1,1, asciinum, 16 ; io 1,1, the converted number

ret