**VISHWAKARMA INSTITUTE OF TECHNOLOGY**

**MICROCONTROLLER AND MICROPROCESSOR:**

**CONTINUOUS ASSIGNMENT**

**Name:** Aniket Jitendra Kalbhor

**Division:** CS B

**Batch:** 1

**Roll Number:** 48

**PRN:** 12210601

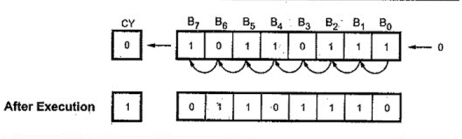
# ASSIGNMENT 1

**Aim:** Write a 64-bit ALP to count the number of positive and negative numbers from a given number (numbers can be defined in ds).

**Theory:**

1. The program systematically goes through a given set of numbers.
2. For each number, it determines whether it is positive or negative.
3. Two separate counters keep track of positive and negative numbers, respectively.
4. Once the program has checked all the numbers, it displays the counts of positive and negative numbers.

**Mathematical Model:**

****If the Carry Flag generated 0, it is Positive Number

If the Carry Flag generated 1, it is Negative Number

**Algorithm:**

1. Start

2. Accept Number from User

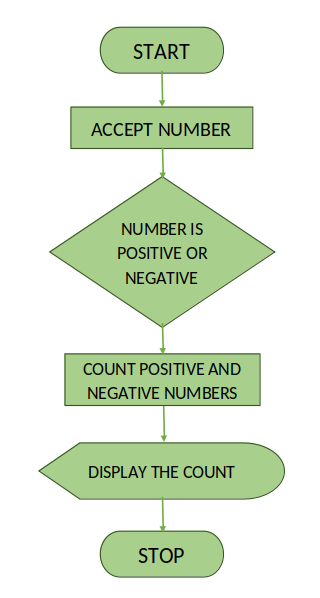
3. Check whether the number is positive or negative.

4. Count the positive and negative numbers.

5. Display the Count

6. Stop

**Flowchart:**

****

**Conclusion:**

The program effectively calculates the amount of positive and negative numbers within a given set. Using a straightforward algorithm, it iterates through each number, incrementing the appropriate counter.

**Code:**

%macro println 2

mov rax,01h

mov rdi,01h

mov rsi,%1

mov rdx,%2

syscall

%endmacro

printName:

mov rax, 01h

mov rdi, 01h

mov rsi, name

mov rdx, nameLength

syscall

ret

section .data

name db 'Aniket Kalbhor - 48'

nameLength equ $-name

array db 07Fh,40h,3Fh,41h,80h

msg0 db 10,'Array Elements Are :: '

len0 equ $-msg0

msg1 db 10,'Positive no Count :: '

len1 equ $-msg1

msg2 db 10,'Negative no Count :: '

len2 equ $-msg2

msg3 db ' '

len3 equ $-msg3

section .bss

positive resb 1

negative resb 1

display resb 2

section .text

global \_start

\_start:

call printName

mov cl,05h

mov rsi,array

mov bh,00h ;positive

mov bl,00h ;negative

chkNext:

mov al,[rsi]

shl al,01h

jnc posi

inc bl

jmp skip

posi:

inc bh

skip:

inc rsi

dec cl

jnz chkNext

mov [positive],bh

mov [negative],bl

println msg0,len0

mov cl,05h

mov rsi,array

dispNext:

mov bl,[rsi]

push rcx

push rsi

call displayNo

println msg3,len3

pop rsi

pop rcx

inc rsi

dec cl

jnz dispNext

println msg1,len1

mov bl,[positive]

call displayNo

println msg2,len2

mov bl,[negative]

call displayNo

mov rax,60

syscall

displayNo:

mov al,bl

and al,0f0h

mov cl,04h

shr al,cl

add al,30h

cmp al,39h

jle dontAdd

add al,07h

dontAdd:

mov [display],al

mov al,bl

and al,0fh

add al,30h

cmp al,39h

jle dontAddd

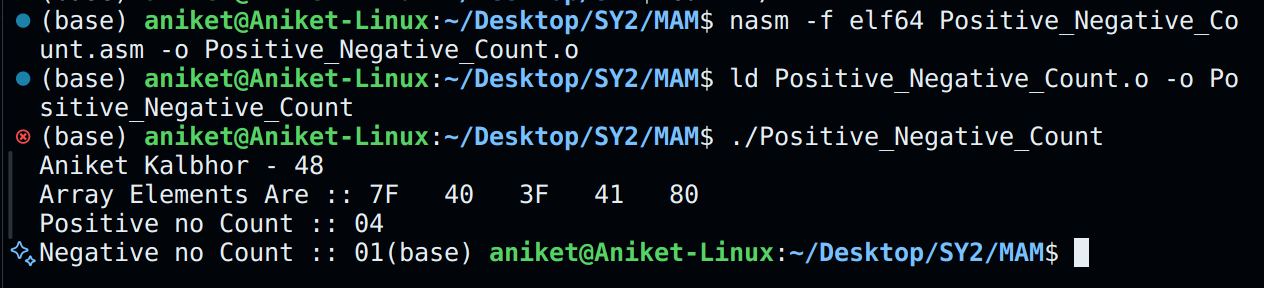
add al,07h

dontAddd:

mov [display+1],al

println display,02h

ret

OUTPUT-

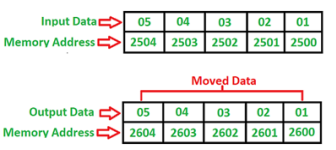
# ASSIGNMENT 2

**Aim:** Write a 64-bit ALP to transfer a block of data without overlapping data that can be defined in the data segment.

**Theory:**

1. The assembly program facilitates the transfer of an array of bytes in memory.
2. The user first inputs the array, which is then displayed before and after the transfer operation.
3. The program is tailored for 64-bit systems and employs system calls for input/output operations.
4. Specifically, the program begins by soliciting the number of elements in the array from the user.
5. Subsequently, the program prompts the user to input the array elements and displays them after receiving the input.
6. Next, the program computes the number of overlapping elements in the array based on user input.
7. Upon calculation, it executes the transfer operation, ensuring that no overlapping data is lost.
8. Finally, the program displays the array again after the transfer operation.

**Mathematical Model:**

****

**Algorithm:**

1. Start

2. Accept Source Elements from User

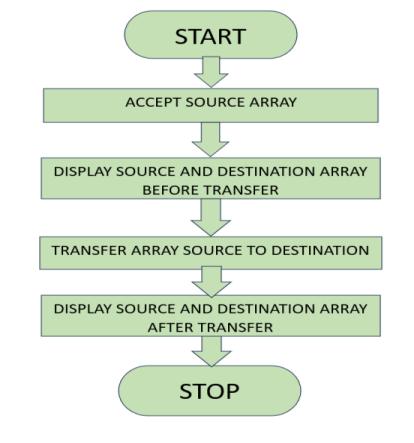
3. Display Source and Destination array before Transfer.

4. Transfer Array Source to Destination.

5. Display the Source and Destination Array.

6. Stop

**Flowchart:**

****

**Conclusion:**

Explained the necessity of preventing overlapping in data transfer activities, emphasizing the critical role of efficient data movement.

**Code:**

%macro println 2

mov rax,01h

mov rdi,01h

mov rsi,%1

mov rdx,%2

syscall

%endmacro

printName:

mov rax, 01h

mov rdi, 01h

mov rsi, name

mov rdx, nameLength

syscall

ret

section .data

name db 'Aniket Kalbhor - 48'

nameLength equ $-name

msg0 db 10,'How Many Ele :: '

len0 equ $-msg0

msg1 db 10,'Enter 5 array Elements :'

len1 equ $-msg1

msg2 db 10,'Before Transfer :'

len2 equ $-msg2

msg3 db 10,'SRC is ::'

len3 equ $-msg3

msg4 db 10,'Dest is::'

len4 equ $-msg4

msg5 db 10,'After Transfer :'

len5 equ $-msg5

space db ' '

section .bss

src resb 25

dest resb 25

cnt resb 1

display resb 2

accept resb 3

section .text

global \_start

\_start:

call printName

println msg0,len0

call acceptNo

mov [cnt],bl

println msg1,len1

mov cl,[cnt]

mov rsi,src

movNext:

push rcx

push rsi

call acceptNo

pop rsi

pop rcx

mov [rsi],bl

inc rsi

dec cl

jnz movNext

println msg2,len2

println msg3,len3

mov cl,[cnt]

mov rsi,src

dispNext:

mov bl,[rsi]

push rcx

push rsi

call displayNo

println space,02h

pop rsi

pop rcx

inc rsi

dec cl

jnz dispNext

println msg4,len4

mov cl,[cnt]

mov rsi,dest

dispNext1:

mov bl,[rsi]

push rcx

push rsi

call displayNo

println space,02h

pop rsi

pop rcx

inc rsi

dec cl

jnz dispNext1

mov cl,[cnt]

mov rsi,src

mov rdi,dest

next:

mov al,[rsi]

mov [rdi],al

inc rsi

inc rdi

dec cl

jnz next

println msg5,len5

println msg3,len3

mov cl,[cnt]

mov rsi,src

dispNext2:

mov bl,[rsi]

push rcx

push rsi

call displayNo

println space,02h

pop rsi

pop rcx

inc rsi

dec cl

jnz dispNext2

println msg4,len4

mov cl,[cnt]

mov rsi,dest

dispNext3:

mov bl,[rsi]

push rcx

push rsi

call displayNo

println space,02h

pop rsi

pop rcx

inc rsi

dec cl

jnz dispNext3

mov rax,60

syscall

acceptNo:

mov rax,00h

mov rdi,00h

mov rsi,accept

mov rdx,03h

syscall

mov al,[accept]

sub al,30h

cmp al,09h

jle dontSub

sub al,07h

dontSub:

mov cl,04h

shl al,cl

mov bl,al

mov al,[accept+1]

sub al,30h

cmp al,09h

jle dontSubb

sub al,07h

dontSubb:

or bl,al

ret

displayNo:

mov al,bl

and al,0f0h

mov cl,04h

shr al,cl

add al,30h

cmp al,39h

jle dontAdd

add al,07h

dontAdd:

mov [display],al

mov al,bl

and al,0fh

add al,30h

cmp al,39h

jle dontAddd

add al,07h

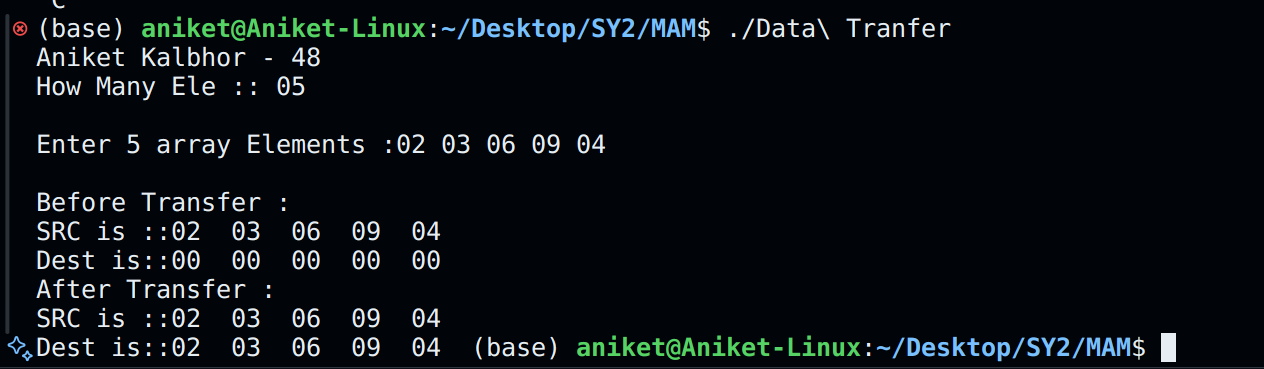
dontAddd:

mov [display+1],al

println display,02h

ret

**Output:**

****

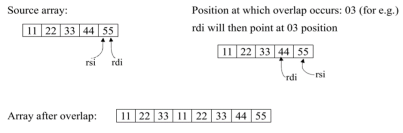
# ASSIGNMENT 3

**Aim:** Write a 64-bit ALP to transfer a block of data with overlapping data that can be defined in data segment.

**Theory:**

1. The assembly language program facilitates the transfer of a byte array between memory locations in a 64-bit system, ensuring efficient handling of potential data overlaps.
2. The program begins by prompting the user to specify the desired number of elements in the array, followed by the input of the array elements themselves.
3. The program displays the contents of the array both before and after the transfer operation, providing a clear view of the changes made.
4. At the start of the program, the user is prompted to provide the number of elements in the array, which is essential for subsequent operations.
5. The program assists the user in entering the array elements and subsequently displays the contents of the array after the input process.
6. Based on the user-provided information, the program computes the potential overlap within the array, ensuring efficient handling of data during the transfer.
7. Utilizing the computed overlap, the program executes the transfer operation, prioritizing the preservation of all data without any loss.
8. Upon completion of the transfer the program, this time reflecting the outcomes of the transfer operation.

**Mathematical Model:**

****

**Algorithm:**

1. Start

2. Accept Source Elements from User

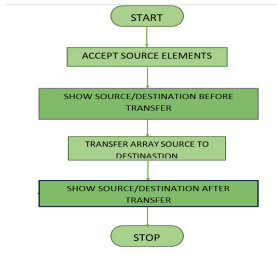
3. Display Source and Destination array before Transfer.

4. Transfer Array Source to Destination with Overlap.

5. Display the Source and Destination Array.

6. Stop

**Flowchart:**

****

**Conclusion:**

The program achieves the goal of transferring a block of data with overlapping data efficiently and accurately.

**Code:**

%macro println 2

mov rax,01h

mov rdi,01h

mov rsi,%1

mov rdx,%2

syscall

%endmacro

printName:

mov rax, 01h

mov rdi, 01h

mov rsi, name

mov rdx, nameLength

syscall

ret

section .data

name db 'Aniket Kalbhor - 48'

nameLength equ $-name

msg0 db 10,'How Many Array Elements :'

len0 equ $-msg0

msg1 db 'Enter Array Elements :'

len1 equ $-msg1

msg2 db 10,'Before Transfer :'

len2 equ $-msg2

msg3 db 10,'Array is :'

len3 equ $-msg3

msg4 db 10,'After Transfer :'

len4 equ $-msg4

msg5 db 10,'Overlap Count :'

len5 equ $-msg5

space db ' '

section .bss

array resb 20

accept resb 3

display resb 2

cnt resb 1

ovcnt resb 1

section .text

global \_start

\_start:

call printName

println msg0,len0

call acceptNo

mov [cnt],bl

println msg1,len1

mov rsi,array

mov cl,[cnt]

acceptNext:

push rcx

push rsi

call acceptNo

pop rsi

pop rcx

mov [rsi],bl

inc rsi

dec cl

jnz acceptNext

println msg5,len5

call acceptNo

mov [ovcnt],bl

println msg2,len2

println msg3,len3

mov rsi,array

mov cl,20

dispNext:

push rsi

push rcx

mov bl,[rsi]

call displayNo

pop rcx

pop rsi

inc rsi

dec cl

jnz dispNext

mov rdi,array

mov rcx,00h

mov cl,[cnt]

add cl,cl

sub cl,[ovcnt]

add rdi,rcx

dec rdi

mov rcx,00h

mov cl,[cnt]

mov rsi,array

add rsi,rcx

dec rsi

; std

; rep movsb

movAgain:

mov al,[rsi]

mov [rdi],al

dec rsi

dec rdi

dec cl

jnz movAgain

println msg4,len4

println msg3,len3

mov rsi,array

mov cl,20

dispNext1:

push rsi

push rcx

mov bl,[rsi]

call displayNo

pop rcx

pop rsi

inc rsi

dec cl

jnz dispNext1

mov rax,60

syscall

acceptNo:

mov rax,00h

mov rdi,00h

mov rsi,accept

mov rdx,03h

syscall

mov al,[accept]

sub al,30h

cmp al,09h

jle dontSub

sub al,07h

dontSub:

mov cl,04h

shl al,cl

mov bl,al

mov al,[accept+1]

sub al,30h

cmp al,09h

jle dontSubb

sub al,07h

dontSubb:

or bl,al

ret

displayNo:

mov al,bl

and al,0f0h

mov cl,04h

shr al,cl

add al,30h

cmp al,39h

jle dontAdd

add al,07h

dontAdd:

mov [display],al

mov al,bl

and al,0fh

add al,30h

cmp al,39h

jle dontAddd

add al,07h

dontAddd:

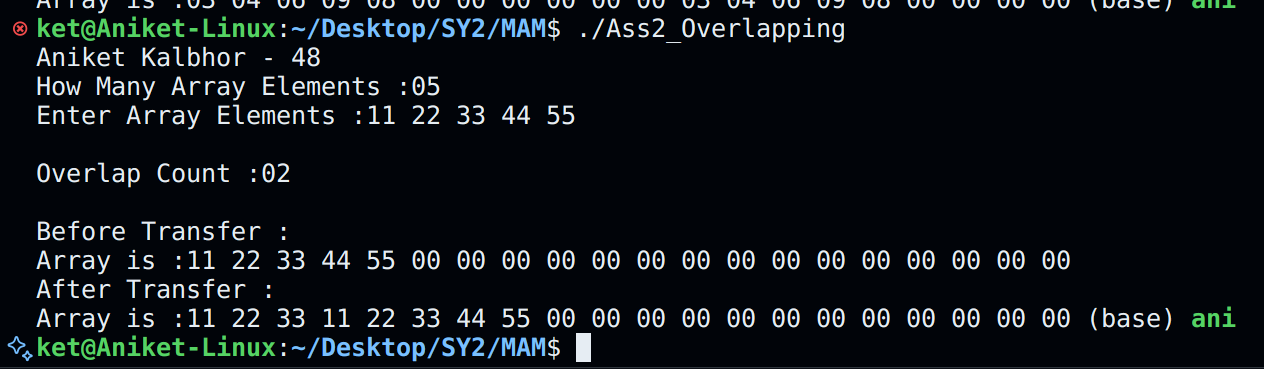
mov [display+1],al

println display,02h

println space,01h

ret

**Output-**

****

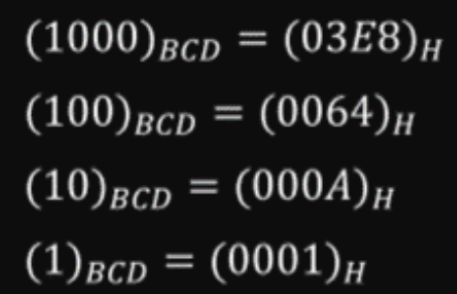
# ASSIGNMENT 4

**Aim:** Write a 64-bit ALP to convert the given BCD number to Hexadecimal Number.

**Theory:**

1. The program prompts the user to enter a five-digit BCD number.
2. The BCD number is read digit by digit, and its corresponding numeric value is obtained by subtracting the ASCII code of '0'.
3. To convert each digit to its decimal equivalent, it is multiplied by its corresponding weight (10000, 1000, 100, 10, 1).
4. The total decimal value is obtained by adding the individual decimal values.
5. The total decimal value is then converted into its hexadecimal equivalent.
6. The hexadecimal result is stored in the "hex" variable.

**Mathematical Model:**

****

4789= 4\*1000 + 7\*100 + 8\*10 + 9\*1

4 \* 03E8 = 0FA0

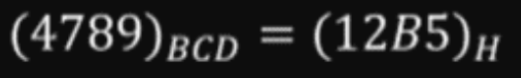
7 \* 0064 = 02BC

8 \*000A = 0050

9 \* 0001 = 0009

Total: 12B5

Therefore,



**Algorithm:**

1. Start

2. Accept BCD Number from User

3. Convert the BCD Number to Hex Number.

4. Display the Number

5. Stop

**Flowchart**

****

**Conclusion**

In x86-64 assembly language, the assembly code efficiently converts a 5-digit Binary Coded Decimal (BCD) number into its hexadecimal equivalent, enhancing readability and maintainability. It accomplishes this through the seamless integration of macros and system calls, which enable smooth user interaction and effective output handling.

**Code:**

%macro println 2

mov rax,01h

mov rdi,01h

mov rsi,%1

mov rdx,%2

syscall

%endmacro

printName:

mov rax, 01h

mov rdi, 01h

mov rsi, name

mov rdx, nameLength

syscall

ret

section .data

name db 'Aniket Kalbhor - 48'

nameLength equ $-name

msg1 db 10,'Enter 5 digit BCD Number :'

len1 equ $-msg1

msg2 db 10,'Eq Hex is :: '

len2 equ $-msg2

section .bss

display resb 2

number resb 1

hex resb 2

carry resb 1

section .text

global \_start

\_start:

call printName

println msg1,len1

mov rax,00h

mov rdi,00h

mov rsi,number

mov rdx,01h

syscall

mov al,[number]

sub al,30h

mov ah,00h

mov bx,2710h

mul bx

mov [hex],ax

cmp dx,0000h

je goAhead

inc byte[carry]

goAhead:

mov rax,00h

mov rdi,00h

mov rsi,number

mov rdx,01h

syscall

mov al,[number]

sub al,30h

mov ah,00h

mov bx,03e8h

mul bx

add [hex],ax

jnc goAhead1

inc byte[carry]

goAhead1:

mov rax,00h

mov rdi,00h

mov rsi,number

mov rdx,01h

syscall

mov al,[number]

sub al,30h

mov ah,00h

mov bx,0064h

mul bx

add [hex],ax

jnc goAhead2

inc byte[carry]

goAhead2:

mov rax,00h

mov rdi,00h

mov rsi,number

mov rdx,01h

syscall

mov al,[number]

sub al,30h

mov ah,00h

mov bx,000ah

mul bx

add [hex],ax

jnc goAhead3

inc byte[carry]

goAhead3:

mov rax,00h

mov rdi,00h

mov rsi,number

mov rdx,01h

syscall

mov al,[number]

sub al,30h

mov ah,00h

add [hex],ax

jnc goAhead4

inc byte[carry]

goAhead4:

println msg2,len2

cmp byte[carry],00h

je dispNext

add byte[carry],30h

println carry,01h

dispNext:

mov bl,[hex+1]

call displayNo

mov bl,[hex]

call displayNo

mov rax,60

syscall

displayNo:

mov al,bl

and al,0f0h

mov cl,04h

shr al,cl

add al,30h

cmp al,39h

jle dontAdd

add al,07h

dontAdd:

mov [display],al

mov al,bl

and al,0fh

add al,30h

cmp al,39h

jle dontAddd

add al,07h

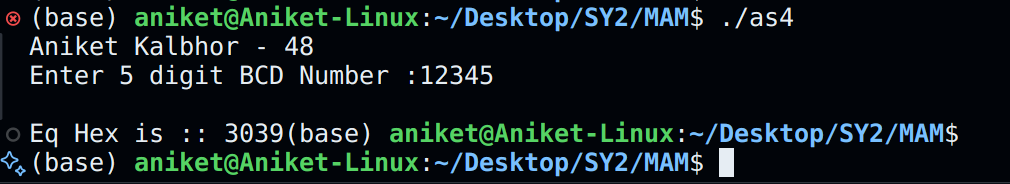
dontAddd:

mov [display+1],al

println display,02h

ret

**Output-**

****

# ASSIGNMENT 5

**Aim:** Write a 64-bit ALP to convert the given HEX number to BCD Number.

**Theory:**

1. Converting a hexadecimal number to Binary Coded Decimal (BCD) involves converting each hexadecimal digit to binary and then arranging these binary digits into groups of four to represent each decimal digit.
2. This maintains the numerical value while adjusting the encoding format.
3. The process entails decomposing the hexadecimal number into binary digits and arranging them in BCD format, ensuring the precise representation of each decimal digit.
4. This conversion is crucial in various applications that require BCD representation, such as digital systems where accurate decimal display or BCD arithmetic operations are necessary.
5. Through bitwise functions and arithmetic operations, the hexadecimal input is transformed to comply with BCD format, ensuring accuracy and compatibility during further processing or display.

**Mathematical Model:**

4789 = 4 \* 1000 + 7 \*100 + 8\*10 + 9\*1 i.e.,

12B5 = 4 \*03E8 + 7 \*0064 + 8\*000A + 9\*0001.

So, by dividing the hex number by respective place gives the decimal number.

**Algorithm:**

1. Start

2. Accept HEX Number from User

3. Convert the HEX Number to BCD Number.

4. Display the Number

5. Stop

**Flowchart:**

****

**Conclusion:**

**Output Text:**

In low-level programming tasks and embedded systems, the conversion of hexadecimal numbers to Binary Coded Decimal (BCD) format is crucial. This conversion process alters the encoding scheme, ensuring accuracy and facilitating compatibility with BCD systems. As a result, handling and manipulating numerical data becomes more efficient, providing enhanced performance in various applications.

**Code:**

%macro println 2

mov rax,01h

mov rdi,01h

mov rsi,%1

mov rdx,%2

syscall

%endmacro

printName:

mov rax, 01h

mov rdi, 01h

mov rsi, name

mov rdx, nameLength

syscall

ret

section .data

name db 'Aniket Kalbhor - 48'

nameLength equ $-name

msg1 db 10,'Enter 4 digit Hex ::'

len1 equ $-msg1

msg2 db 10,'Eq BCd :: '

len2 equ $-msg2

section .bss

quo resb 1

accept resb 2

hex resb 2

remd resb 2

section .text

global \_start

\_start:

call printName

println msg1,len1

call acceptNo

mov [hex+1],bl

call acceptNo

mov [hex],bl

println msg2,len2

mov dx,0000h

mov ax,[hex]

mov bx,2710h

div bx

mov [remd],dx

add al,30h

mov [quo],al

println quo,01h

mov dx,0000h

mov ax,[remd]

mov bx,03e8h

div bx

mov [remd],dx

add al,30h

mov [quo],al

println quo,01h

mov dx,0000h

mov ax,[remd]

mov bx,0064h

div bx

mov [remd],dx

add al,30h

mov [quo],al

println quo,01h

mov dx,0000h

mov ax,[remd]

mov bx,000Ah

div bx

mov [remd],dx

add al,30h

mov [quo],al

println quo,01h

mov ax,[remd]

add al,30h

mov [quo],al

println quo,01h

mov rax,60

syscall

acceptNo:

mov rax,00h

mov rdi,00h

mov rsi,accept

mov rdx,02h

syscall

mov al,[accept]

sub al,30h

cmp al,09h

jle dontSub

sub al,07h

dontSub:

mov cl,04h

shl al,cl

mov bl,al

mov al,[accept+1]

sub al,30h

cmp al,09h

jle dontSubb

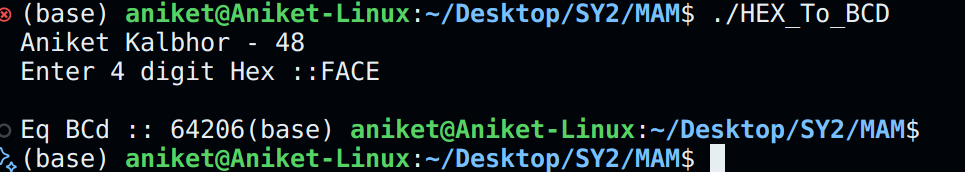
sub al,07h

dontSubb:

or bl,al

ret

**OUTPUT-**

****

# 

# ASSIGNMENT 6

**Aim:** Write a 64-bit ALP for Multiplication of two 8 bit Binary Numbers.

**Theory:**

In assembly language, multiplying two 8-bit binary numbers involves several steps. First, the binary values are loaded into registers. Then, the multiplication is performed using the mul instruction. Finally, the result is stored in a memory location.

In a given ALP, two 8-bit binary numbers are stored in memory locations num1 and num2. These values are loaded into registers AL and BL, respectively. The contents of AL and BL are multiplied using the mul instruction, and the result is stored in AX. Since we are working with 8-bit values, only the lower 8 bits of the result (stored in AL) are considered. This result is then stored in memory location result for further processing or display. Finally, the program ends.

The ALP provided is an implementation of binary multiplication in assembly language.

**Successive Addition Method**

* Initialize a result variable to zero.
* Set a loop counter equal to the value of the multiplier.
* In a loop, repeatedly add the multiplicand to the result and decrement the loop counter with each iteration until it reaches zero.
* Consider potential overflow and handle it by discarding higher order bits or using larger data types.

**Add and Shift Method**

* Initialize a result variable to zero.
* Enter a loop that iterates through each bit of the multiplier, starting from the least significant bit and moving towards the most significant bit.
* Examine the current bit of the multiplier and if it's set (equal to 1), add the multiplicand to the result.
* Shift the result one position to the left, effectively doubling it.
* Repeat this process for all bits in the multiplier until all bits have been processed.
* The result variable holds the product of the two 8-bit numbers.

**Mathematical Model:**

**Successive Addition Method:**

For example: AL = 12 H, BL = 10 H Result = 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H

Result = 0120 H

**Add and Shift Method**

For example: AL = 54 H, BL = 65 H

AL 0101 0100

BL 0110 0101

0101 0100

00000 000

010101 00

0000000 0

00000000

01010100

01010100

00000000

0010 0001 0010 0100

**Algorithm:**

1. Start

2. Accept binary number from User

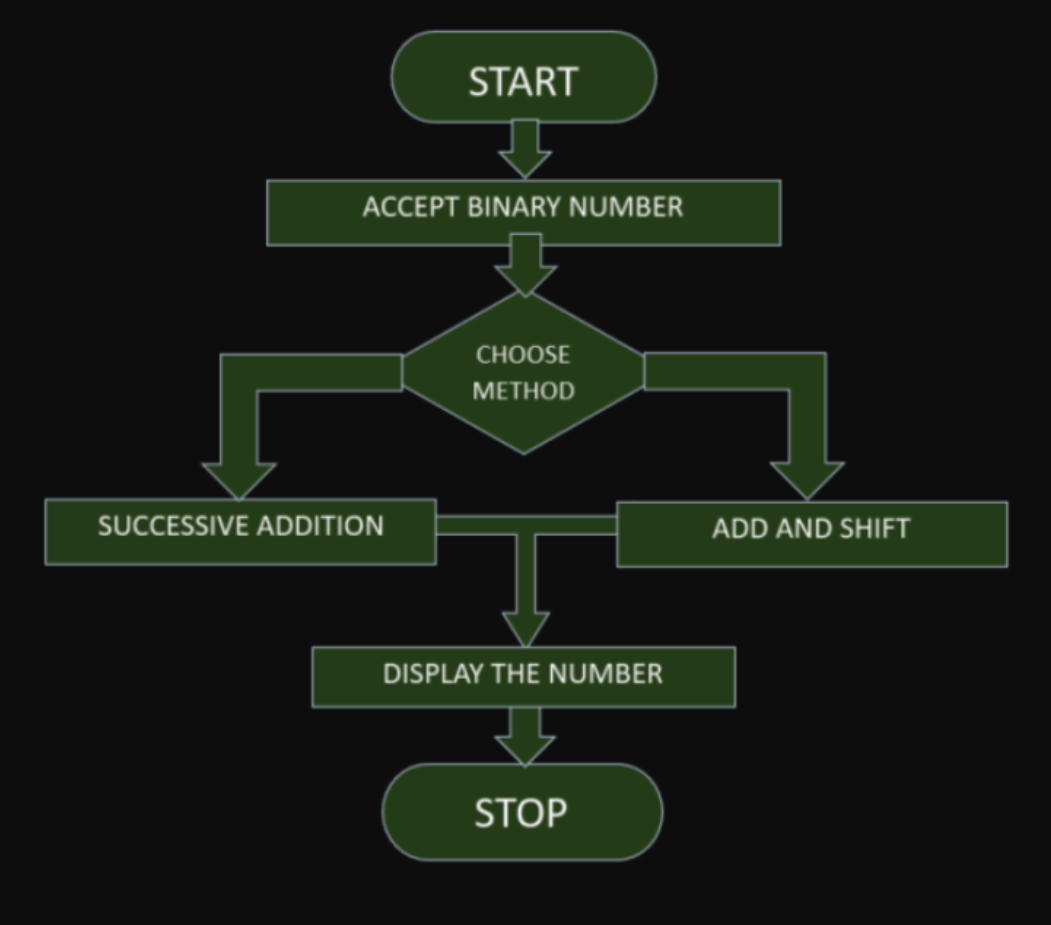
3. Successive Addition

4. Add and Shift

5. Display the Number

6. Stop

**Flowchart:**

****

**Conclusion:**

Multiplication of two 8-bit binary numbers is done effectively by the provided assembly language software. In it, the process of loading binary numbers into registers, multiplying them using the mul instruction, and storing the outcome for later use or display is demonstrated. This ALP illustrates the basic concepts of binary manipulation and arithmetic inside the x86 architecture, and can be used as a basis for more intricate binary arithmetic operations in assembly language programming.

**Code:**

%macro println 2

mov rax,01h

mov rdi,01h

mov rsi,%1

mov rdx,%2

syscall

%endmacro

printName:

mov rax, 01h

mov rdi, 01h

mov rsi, name

mov rdx, nameLength

syscall

ret

section .data

name db 'Aniket Kalbhor - 48'

nameLength equ $-name

msg1 db 10,'Menu: '

db 10,'1.S Addition'

db 10,'2.A & S'

db 10,'3. exit'

db 10,'Choice :'

len1 equ $-msg1

msg2 db 10,'Enter Multiplicand : '

len2 equ $-msg2

msg3 db 10,'Enter Multiplier :'

len3 equ $-msg3

msg4 db 10,'Product is :'

len4 equ $-msg4

section .bss

mulr resb 1

muld resb 1

result resb 2

accept resb 3

display resb 2

choice resb 2

section .text

global \_start

\_start:

call printName

menuAgain:

println msg1,len1

mov rax,00h

mov rdi,00h

mov rsi,choice

mov rdx,02h

syscall

mov al,[choice]

cmp al,31h

jne next

call SAddition

jmp menuAgain

next:

cmp al,32h

jne next1

call AddShift

jmp menuAgain

next1:

cmp al,33h

je exit

jmp menuAgain

exit:

mov rax,60

syscall

SAddition:

println msg2,len2

call acceptNo

mov [muld],bl

println msg3,len3

call acceptNo

mov [mulr],bl

mov dx,0000h

mov cl,[mulr]

mov bl,[muld]

mov bh,00h

addNext:

add dx,bx

dec cl

jnz addNext

mov [result],dx

println msg4,len4

mov bl,[result+1]

call displayNo

mov bl,[result]

call displayNo

ret

AddShift:

println msg2,len2

call acceptNo

mov [muld],bl

println msg3,len3

call acceptNo

mov [mulr],bl

mov al,[mulr]

mov cl,08h

mov bl,[muld]

mov bh,00h

mov dx,0000h

chkNextBit:

shr al,01h

jnc onlyShift

add dx,bx

onlyShift:

shl bx,01h

dec cl

jnz chkNextBit

mov [result],dx

println msg4,len4

mov bl,[result+1]

call displayNo

mov bl,[result]

call displayNo

ret

acceptNo:

mov rax,00h

mov rdi,00h

mov rsi,accept

mov rdx,03h

syscall

mov al,[accept]

sub al,30h

cmp al,09h

jle dontSub

sub al,07h

dontSub:

mov cl,04h

shl al,cl

mov bl,al

mov al,[accept+1]

sub al,30h

cmp al,09h

jle dontSubb

sub al,07h

dontSubb:

or bl,al

ret

displayNo:

mov al,bl

and al,0f0h

mov cl,04h

shr al,cl

add al,30h

cmp al,39h

jle dontAdd

add al,07h

dontAdd:

mov [display],al

mov al,bl

and al,0fh

add al,30h

cmp al,39h

jle dontAddd

add al,07h

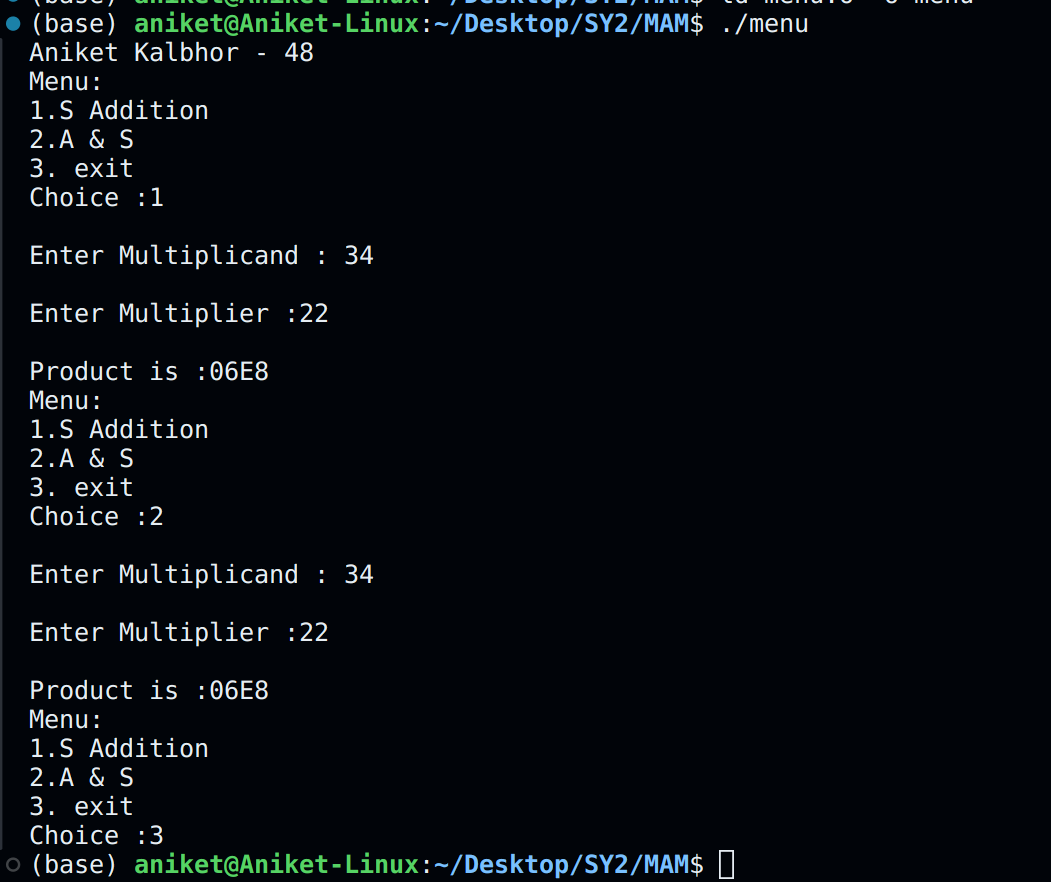
dontAddd:

mov [display+1],al

println display,02h

ret

**OUTPUT-**

****