MATRIX OPERATIONS

Exp No: 5 Name: Srinithyee S K

Date: 09-10-20 Register Number: 185001166

AIM:

To write assembly language programs to perform the following matrix operations:

1. Matrix Addition.
2. Matrix Subtraction.

PROGRAM 1: MATRIX ADDITION

ALGORITHM:

1. Begin.
2. Declare the data segment.
3. Initialize data segment with matrices 1 and 2, with their dimensions and resultant matrix.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Compare row1 and row2, if not equal then exit the program.
10. Compare col1 and col2, if not equal then exit the program.
11. Position SI at matrix1, and DI at matrix2.
12. Multiply row1 and col1 to find length len of the matrix.
13. Move the len to CL register.
14. Till CL goes to zero:
    i. Add values at SI and DI and push it into the stack.
    ii. Increment SI and DI.
    iii. Decrement CL.
15. Move SI to end of resultant matrix.
16. Till CL goes to zero:
    i. Pop the value from top of the stack and put it at SI.
    ii. Decrement SI.
17. Introduce an interrupt for safe exit. (INT 21h)
18. Close the code segment.
19. End

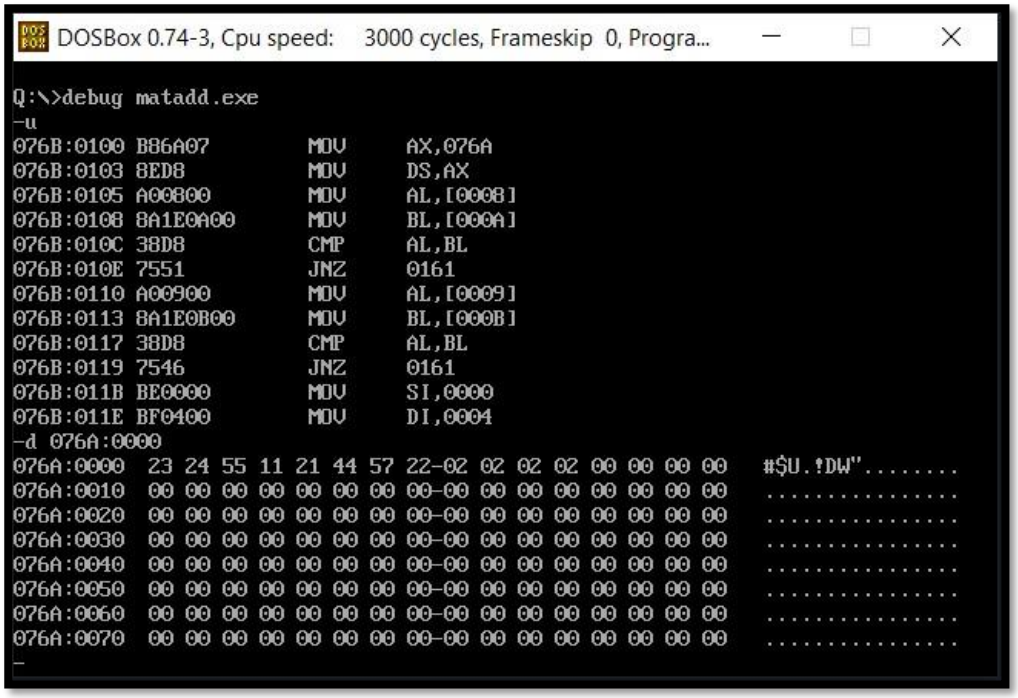
PROGRAM	COMMENTS
assume cs:code, ds:data	Declare code and data segment.
data segment	Initialize data segment with values.
mat1 db 23h,24h,55h,11h	Matrix 1.
mat2 db 21h,44h,57h,22h	Matrix 2.
row1 db 02h	Row count of matrix 1.
col1 db 02h	Column count of matrix 1.
row2 db 02h	Row count of matrix 2.
col2 db 02h	Column count of matrix 2.
len db 00h	Length of matrix.
resi dw ?	Result matrix.
data ends	
	Start the code segment.
code segment	
org 0100h	Initialize an offset address.
start: mov ax, data	Transfer data from "data" to AX.
mov ds, ax	Transfer data from memory location AX to DS.
mov al, row1	Comparing row count of both matrices.
mov bl,	
row2 cmp	Exiting if not same.
al, bl jne	
break	
mov al, col1	Comparing column count of both matrices.
mov bl, col2	
cmp al, bl	Exiting if not same.
jne break	
mov si, offset mat1	Set SI to point to Matrix 1's starting index.
mov di, offset mat2	Set DI to point to Matrix 2's starting index.
mov al, row1	
mov bl, col1	AL has the value of row1 * col1.
mul bl	
mov len, al	
mov cl, len	Finding no. of elements in the matrix.
mov ch, 00h	Clear CH.
mov ax, 0000h	Clear AX.
looper: mov al, [si]	Pushing each element-wise sum into stack
mov ah, 00h	
mov bl, [di]	Add the 2 elements from each matrix.
mov bh, 00h	
add ax, bx	

push ax inc si inc di	 Move to next element in matrix 1. Move to next element in matrix 2.
dec cx jz prewrk jmp looper	Decrement counter by 1. If addition is over, jump to “prewrk” Repeat addition for all elements.

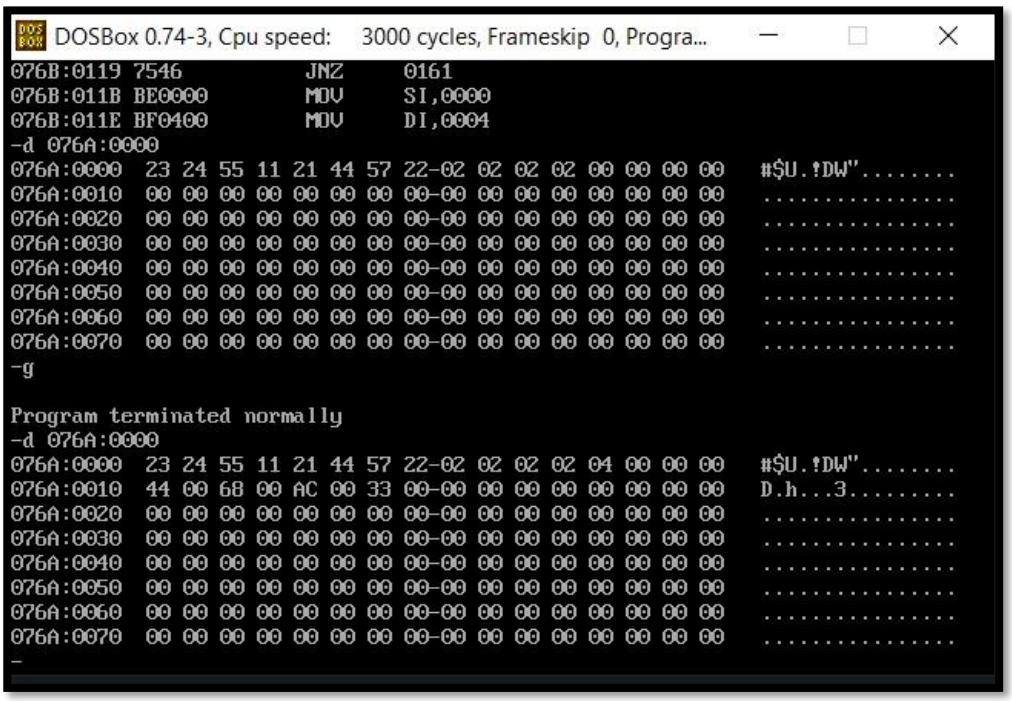
prewrk: mov si, offset resi + 0001h
mov cl, len
mov ch, 00h
add si, cx
	Set the SI to store values in result matrix “resi” properly.
	Set counter to length of the matrix.
	Clear CH.
	Set SI to point to the last location of the matrix.

retloop: pop ax mov [si], al dec si mov [si], ah dec si dec cx jz break jmp retloop	Popping each element from stack into resultant matrix. Decrement SI. Decrement counter by 1. Stop popping if all elements are popped (CX = 0) Pop the next element and put it in the matrix.
break: mov ah, 4ch int 21h code ends end start	Interrupt the process with return code and exit.

UNASSEMBLED CODE:



SAMPLE I/O SNAPSHOT:



PROGRAM – 2: MATRIX SUBTRACTION

ALGORITHM:

1. Begin.
2. Declare the data segment.
3. Initialize data segment with matrices 1 and 2, with their dimensions and resultant matrix.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Compare row1 and row2, if not equal then exit the program 10. Compare col1 and col2, if not equal then exit the program
11. Position SI at matrix1, and DI at matrix2.
12. Multiply row1 and col1 to find length len of the matrix.
13. Move the len to CL register.
14. Till CL goes to zero:
    i. Subtract values at SI and DI and push it into the stack.
    ii. Increment SI and DI.
    iii. Decrement CL.
15. Move SI to end of resultant matrix.
16. Till CL goes to zero:
    i. Pop the value from top of the stack and put it at SI.
    ii. Decrement SI.
17. Introduce an interrupt for safe exit. (INT 21h) 18. Close the code segment.
19. End.

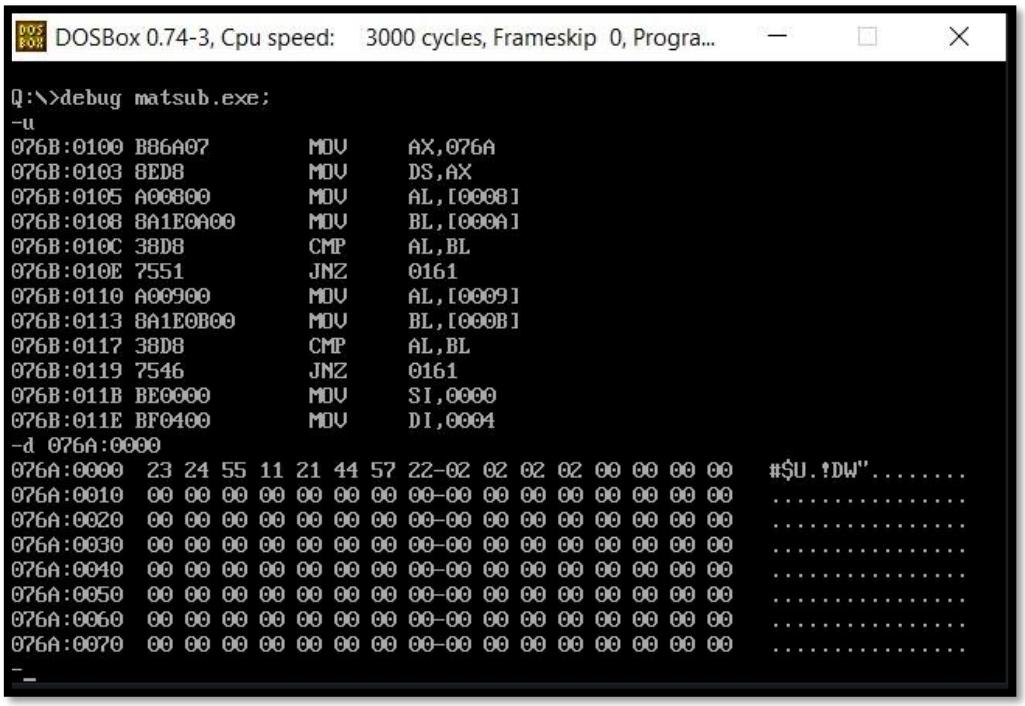
PROGRAM	COMMENTS
assume cs:code, ds:data	Declare code and data segment.
data segment	Initialize data segment with values.
mat1 db 23h,24h,55h,11h	Matrix 1.
mat2 db 21h,44h,57h,22h	Matrix 2.
row1 db 02h	Row count of matrix 1.
col1 db 02h	Column count of matrix 1.
row2 db 02h	Row count of matrix 2.
col2 db 02h	Column count of matrix 2.
len db 00h	Length of matrix.
resi dw ?	Result matrix.
data ends	
code segment	Start the code segment.
org 0100h	Initialize an offset address.
start: mov ax, data	Transfer data from "data" to AX.
mov ds, ax	Transfer data from memory location AX to DS.
mov al, row1	Comparing row count of both matrices.
mov bl, row2 cmp al, bl jne break	Exiting if not same.
mov al, col1	Comparing column count of both matrices.
mov bl, col2 cmp al, bl jne break	Exiting if not same.
mov si, offset mat1	Set SI to point to Matrix 1's starting index.
mov di, offset mat2	Set DI to point to Matrix 2's starting index.
mov al, row1 mov bl, col1 mul bl	AL has the value of row1 * col1.
mov len, al	
mov cl, len	Finding no. of elements in the matrix.
mov ch, 00h	Clear CH.
mov ax, 0000h	Clear AX.
looper: mov al, [si]	Pushing each element-wise sum into stack
mov ah, 00h mov bl, [di] mov bh, 00h sub ax, bx	Subtract the 2 elements from each matrix.

push ax inc si inc di	 Move to next element in matrix 2. Move to next element in matrix 1.
dec cx jz prewrk jmp looper	Decrement counter by 1. If addition is over, jump to “prewrk” Repeat addition for all elements.

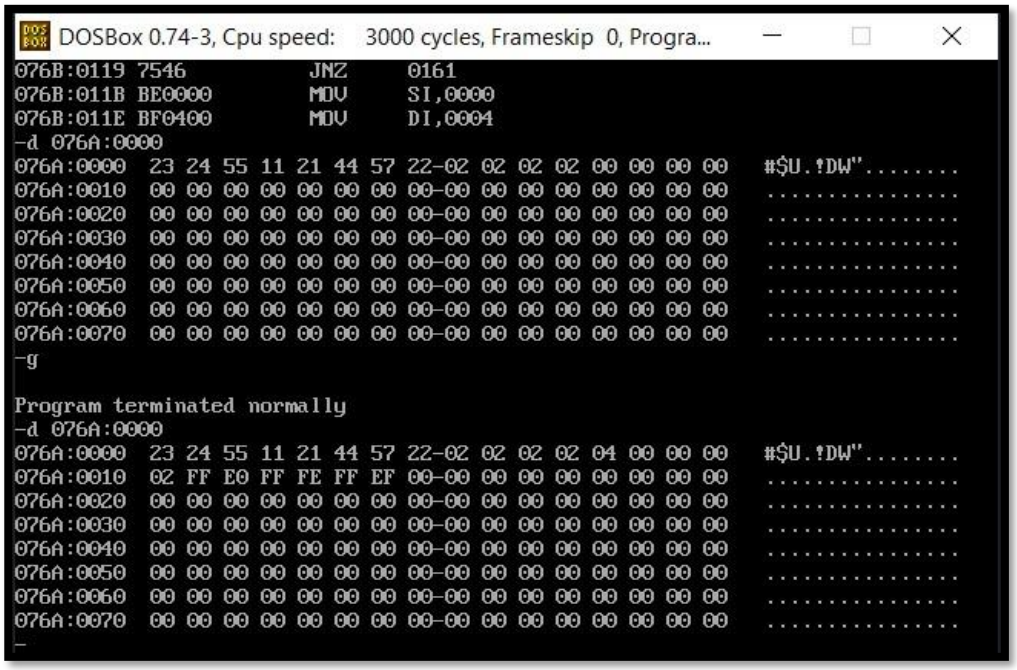
prewrk: mov si, offset resi + 0001h
mov cl, len
mov ch, 00h
add si, cx
add si, cx
	Set the SI to store values in result matrix “resi” properly.
	Set counter to length of the matrix.
	Clear CH.
	Set SI to point to the last location of the matrix.

retloop: pop ax mov [si], al dec si mov [si], ah dec si dec cx	Popping each element from stack into resultant matrix. Decrement SI. Decrement counter by 1.
jz break jmp retloop	Stop popping if all elements are popped (CX = 0) Pop the next element and put it in the matrix.
break: mov ah, 4ch int 21h code ends end start	Interrupt the process with return code and exit.

UNASSEMBLED CODE:



SAMPLE I/O SNAPSHOT:



RESULT:

The assembly level programs were written to perform the above specified matrix operations and the result was verified.

SORTING

Exp No: 6 Name: Srinithyee S K

Date: 11-10-20 Register Number: 185001166

AIM:

To write assembly language programs to perform the following experiments:

1. Ascending order sorting using Bubble Sort.
2. Descending order sorting using Bubble Sort.

PROGRAM 1: SORTING IN ASCENDING ORDER

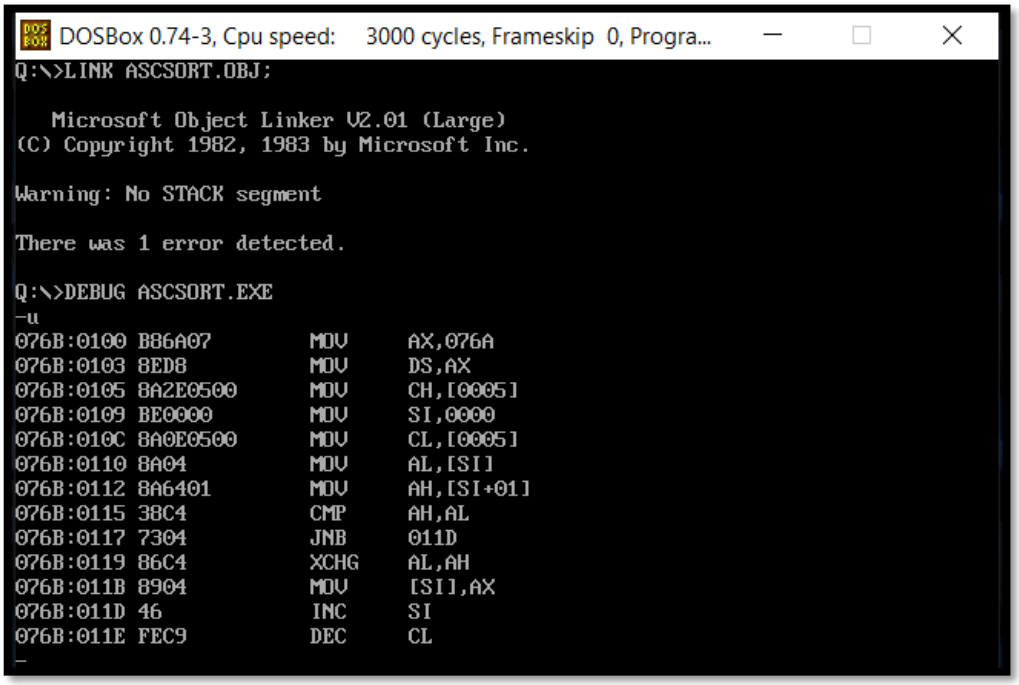
ALGORITHM:

1. Begin.
2. Declare the data segment.
3. Initialize data segment with array and its length len.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Move the length len to CH register.
10. Till CH goes to zero:
    a. Load SI with offset of list.
    b. Move the length len to CL register.
    c. Till CL goes to zero:
        i. Compare values at SI and SI+1 address.
        ii. If value at SI > value at SI+1, exchange them.
        iii. Increment SI.
        iv. Decrement CL.
    d. Decrement CH.
11. Introduce an interrupt for safe exit. (INT 21h)
12. Close the code segment.
13. End.

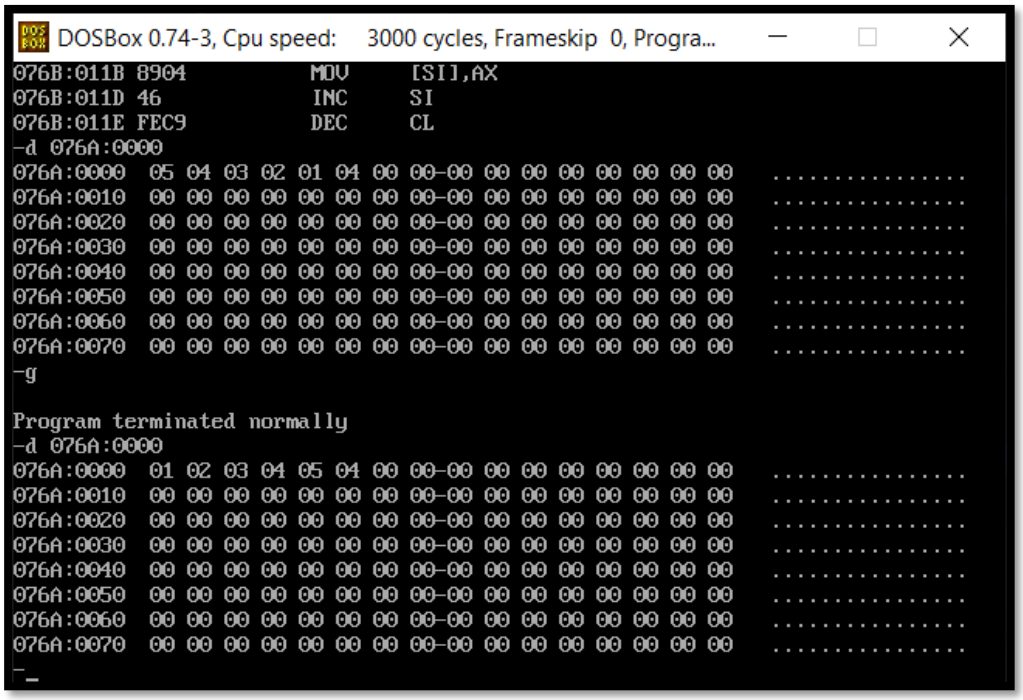
PROGRAM	COMMENTS
assume cs:code, ds:data	Declare code and data segment.
data segment	Initialize data segment with values.
list db 05h, 04h, 03h, 02h, 01h	Stores the list of elements.
len db 04h	Stores the length of the above array.
data ends	
code segment	Start the code segment.
org 0100h	Initialize an offset address.
start: mov ax, data	Transfer data from “data” to AX.
mov ds, ax	Transfer data from memory location AX to DS.
mov ch, len	
outer: mov si, offset list	Pointer at first element.
mov cl, len	Inner loop count.
inner: mov al, [si]	
mov ah, [si+1]	
cmp ah, al	Compare by AL – AH.
jnc skip	Skip if no carry occurred on AL – AH.
xchg al, ah	Exchange register contents.
mov [si], ax	Copy back moved contents to data segment (AL -> [SI], AH -> [SI + 1])
skip: inc si	Go to next element.
dec cl	Decrement inner loop count.
jnz inner	Restart inner loop.
dec ch	Decrement outer loop count.
jnz outer	Restart outer loop.
mov ah, 4ch	
int 21h	Interrupt the process with return code and exit.
code ends	

end start	

UNASSEMBLED CODE:



SAMPLE I/O SNAPSHOT:



PROGRAM 2: SORTING IN DESCENDING ORDER

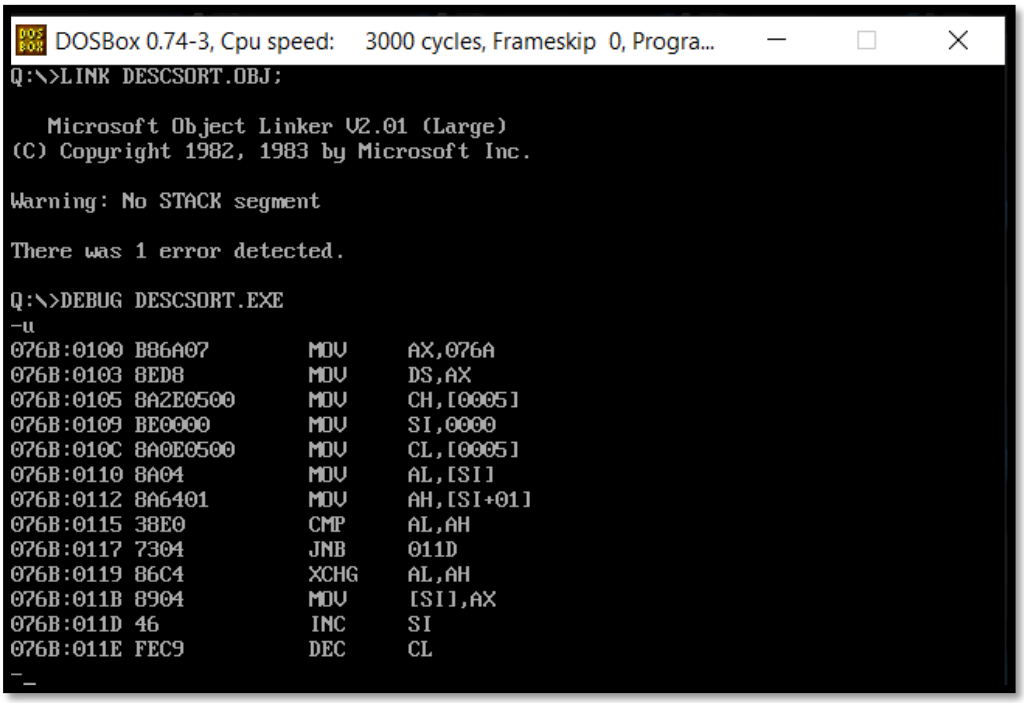
ALGORITHM:

1.
2. Begin.
3. Declare the data segment.
4. Initialize data segment with array and its length len.
5. Close the data segment.
6. Declare the code segment.
7. Set a preferred offset (preferably 100)
8. Load the data segment content into AX register.
9. Transfer the contents of AX register to DS register.
10. Move the length len to CH register.
11. Till CH goes to zero:
    a. Load SI with offset of list.
    b. Move the length len to CL register.
    c. Till CL goes to zero:
        i. Compare values at SI and SI+1 address.
        ii. If value at SI < value at SI+1, exchange them.
        iii. Increment SI.
        iv. Decrement CL.
    d. Decrement CH.
12. Introduce an interrupt for safe exit. (INT 21h)
13. Close the code segment.
14. End.

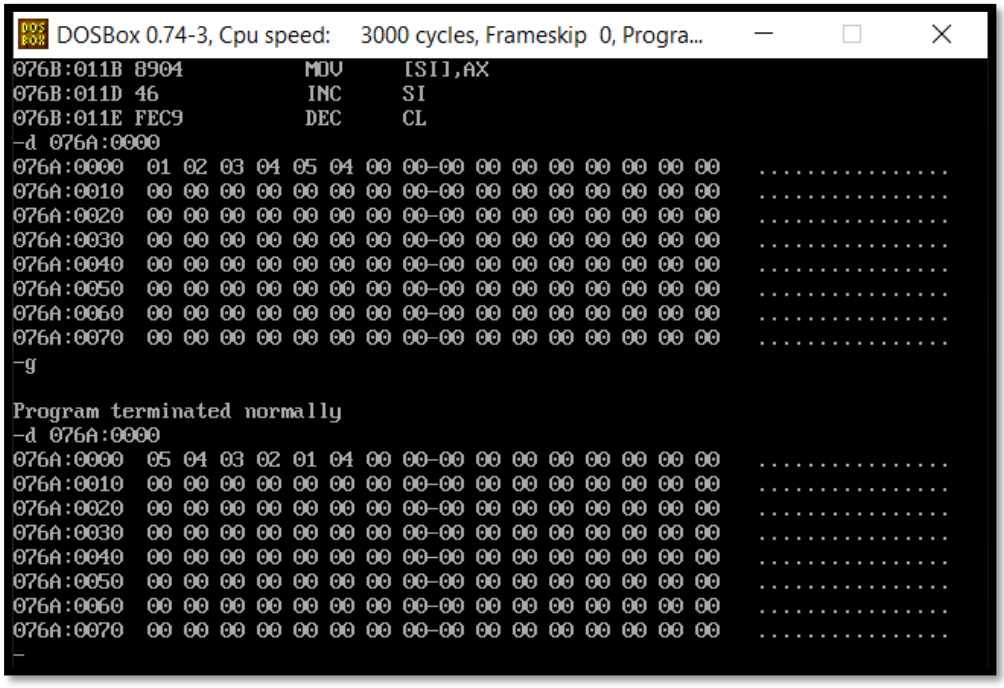
PROGRAM	COMMENTS
assume cs:code, ds:data	Declare code and data segment.
data segment	Initialize data segment with values.
list db 05h, 04h, 03h, 02h, 01h	Stores the list of elements.
len db 04h	Stores the length of the above array.
data ends	
code segment	Start the code segment.
org 0100h	Initialize an offset address.
start: mov ax, data	Transfer data from "data" to AX.
mov ds, ax	Transfer data from memory location AX to DS.
mov ch, len	
outer: mov si, offset list	Pointer at first element.
mov cl, len	Inner loop count.
inner: mov al, [si]	
mov ah, [si+1]	
cmp al, ah	Compare by AH – AL.
jnc skip	Skip if no carry occurred on AH – AL.
xchg al, ah	Exchange register contents.
mov [si], ax	Copy back moved contents to data segment (AL -> [SI], AH -> [SI + 1])
skip: inc si	Go to next element.
dec cl	Decrement inner loop count.
jnz inner	Restart inner loop.
dec ch	Decrement outer loop count.
jnz outer	Restart outer loop.
mov ah, 4ch	
int 21h	Interrupt the process with return code and exit.
code ends	

end start	

UNASSEMBLED CODE:



SAMPLE I/O SNAPSHOT:



RESULT:

The assembly level programs were written to perform the above specified sorting functions and the output was verified.

BCD ADDITION AND SUBTRACTION

Exp No: 7 Name: Srinithyee S K

Date: 12-10-20 Register Number: 185001166

AIM:

To write assembly language programs to perform the following BCD arithmetic operations:

3. BCD Addition.
4. BCD Subtraction.

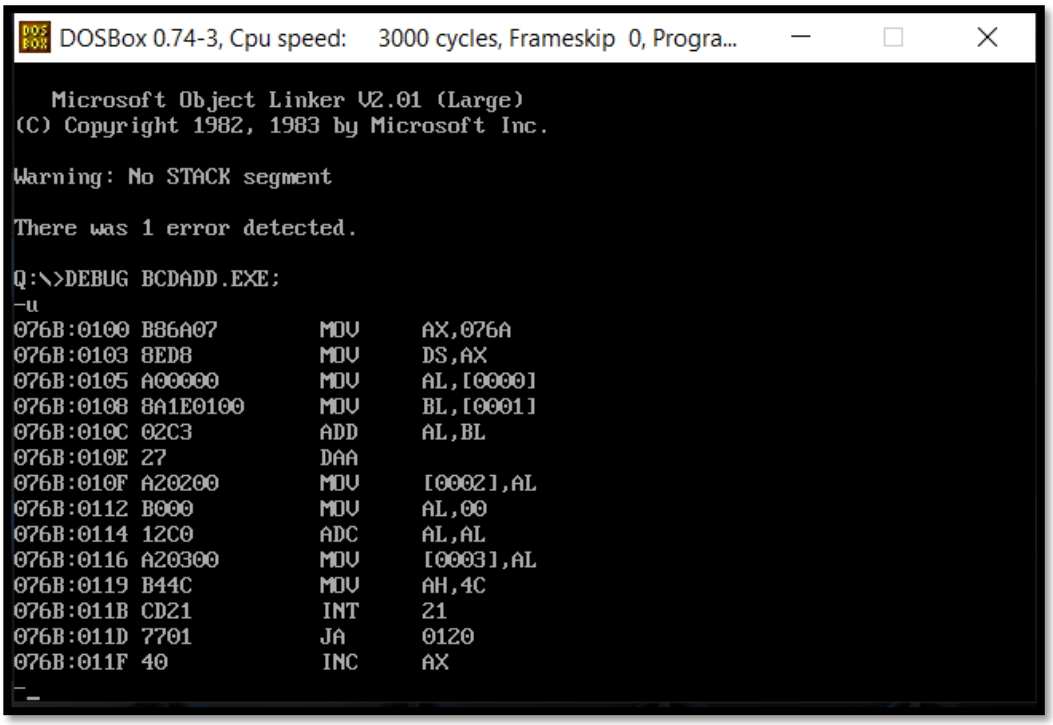
PROGRAM 1: BCD ADDITION

ALGORITHM:

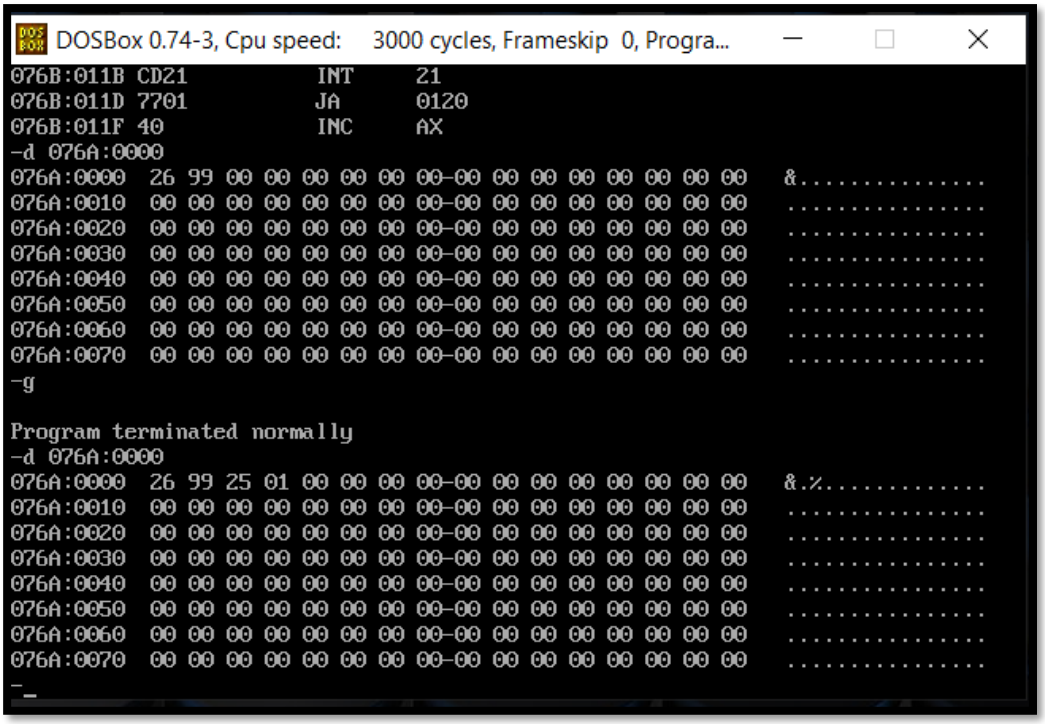
1. Begin.
2. Declare the data segment.
3. Initialize data segment with the 2 BCD numbers and variables for storing their sum and carry.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100h)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Move the contents of the two numbers num1 and num2 to AL and BL register.
10. Add them and store the value in AL.
11. Move the contents of AL to sum.
12. Perform decimal adjust after addition on AL to get BCD result (HEX to BCD)
13. Check if the above adjustment produced a carry.
    a. If carry was produced, set the variable carry to 1.
    b. Else, continue.
14. Transfer the adjusted addition result to the variable sum.
15. Introduce an interrupt for safe exit. (INT 21h)
16. Close the code segment.
17. End.

PROGRAM	COMMENTS
assume cs:code, ds:data	Declare code and data segment.
data segment	Initialize data segment with values.
num1 db 26h	Stores the first BCD number.
num2 db 99h	Stores the second BCD number.
res db ?	Variable to store the sum of the 2 numbers.
carry db ?	Variable to store the carry of the above sum.
data ends	
code segment	Start the code segment.
org 0100h	Initialize an offset address.
start: mov ax, data	Transfer data from “data” to AX.
mov ds, ax	Transfer data from memory location AX to DS.
mov al, num1	Copy num1 to AL.
mov bl, num2	Copy num2 to BL.
mov cl, 00h	Clear CL register.
add al, bl	AL = AL + BL
daa	Adjust HEX result to BCD after subtraction.
jnc resume	If carry was not produced, jump to “resume”.
inc cl	Increment CL register by 1.
resume: mov res, al	Transfer AL contents to variable res.
mov carry, cl	Transfer CL contents to variable carry.
break: mov ah, 4ch	
int 21h	Interrupt the process with return code and exit.
code ends	
end start	

UNASSEMBLED CODE:



SAMPLE I/O SNAPSHOT:



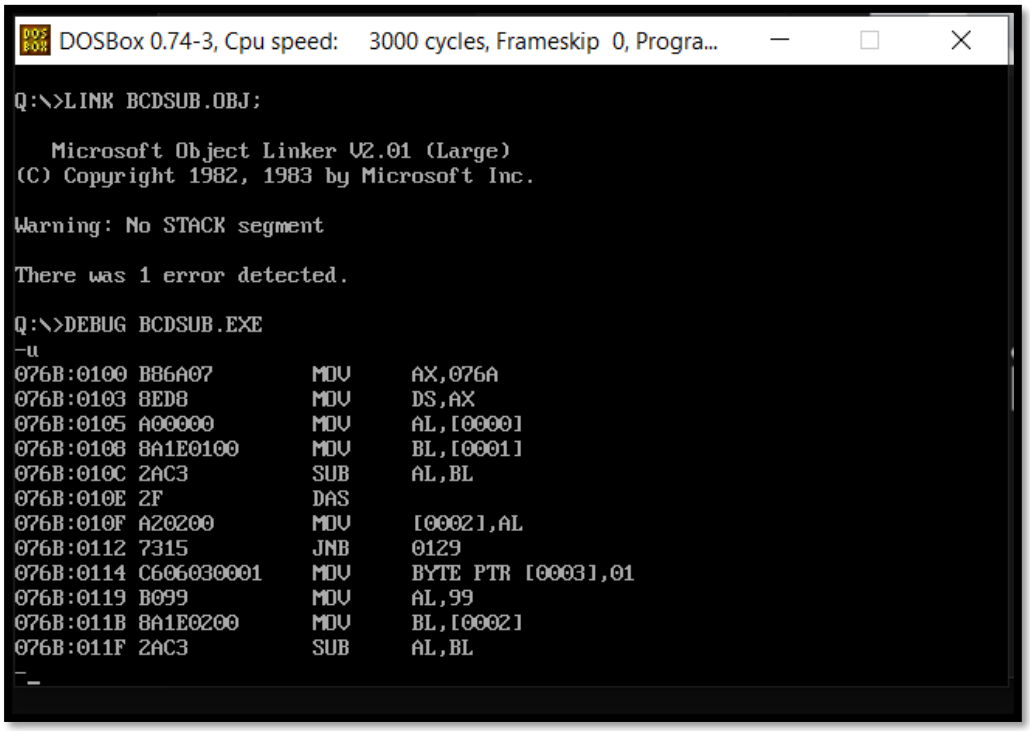
PROGRAM 1: BCD SUBTRACTION

ALGORITHM:

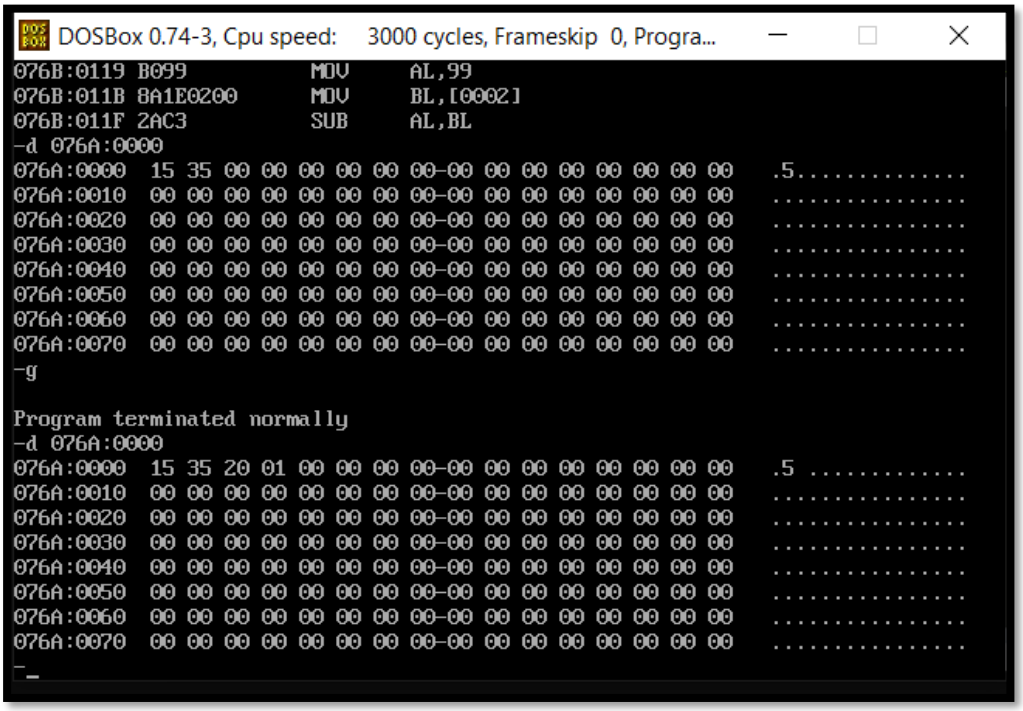
1. Begin.
2. Declare the data segment.
3. Initialize data segment with the 2 BCD numbers and variables for storing their difference (diff) and sign.
4. Close the data segment.
5. Declare the code segment.
6. Set a preferred offset (preferably 100h)
7. Load the data segment content into AX register.
8. Transfer the contents of AX register to DS register.
9. Move the contents of the two numbers num1 and num2 to AL and BL register.
10. Subtract them and store the value in AL.
11. Transfer the contents of AL to diff.
12. If carry flag is set: (Performing 10's complement)
    a. Set sign as 01h.
    b. Move the contents of diff to BL register.
    c. Move 99h to AL register.
    d. Subtract BL from AL and store the value in AL register.
    e. Move 01h to BL register.
    f. Add AL and BL.
    g. Perform decimal adjust on the addition in AL. (HEX to BCD).
    h. Transfer the contents of AL to diff.
13. Introduce an interrupt for safe exit. (INT 21h)
14. Close the code segment.
15. End.

PROGRAM	COMMENTS
assume cs:code, ds:data	Declare code and data segment.
data segment	Initialize data segment with values.
num1 db 26h	Stores the first BCD number.
num2 db 99h	Stores the second BCD number.
diff db ?	Variable to store the difference of the 2 numbers.
sign db ?	Variable to store the sign of the above difference.
data ends	
code segment	Start the code segment.
org 0100h	Initialize an offset address.
start: mov ax, data	Transfer data from “data” to AX.
mov ds, ax	Transfer data from memory location AX to DS.
mov al, num1	Copy num1 to AL.
mov bl, num2	Copy num2 to BL.
sub al, bl	AL = AL – BL
das	Adjust HEX result to BCD after subtraction.
mov diff, al	Transfer AL contents to diff.
jnc break	If carry was not produced, jump to “break”.
mov sign, 01h	If carry was produced, set sign to 1.
mov al, 99h	Set AL = 99h to perform 9’s complement.
mov bl, diff	Transfer diff to BL.
sub al, bl	AL = 99h – BL (9’s complement)
mov bl, 01h	Set BL = 01h.
add al, bl	AL = AL + BL
daa	AL value is decimal adjusted after addition (HEX to BCD)
mov diff, al	Transfer AL contents to diff.
break: mov ah, 4ch	
int 21h	Interrupt the process with return code and exit.
code ends	
end start	

UNASSEMBLED CODE:



SAMPLE I/O SNAPSHOT:



RESULT:

The assembly level programs were written to perform the above specified BCD arithmetic operations and their output was verified.