Group Members:

- Furkan KADIOĞLU 2015400051
- Emilcan ARICAN 2016400231

CODE EXPLANATION

TAKING INPUT

Takes a character input via 01h interrupt.

```
mov ah,01h                  ;reads a character input
int 21h
```

“ENTER” CHARACTER

If the incoming character is “enter” that means expression ended so it jumps to the output

```
cmp al,13                   ;if the incoming character is "enter" jumps to output2
je output2
```

SPACE CHARACTER

If the incoming character is “space” there could be two situations: a number token is ended or a operation token is ended. If the ended token is a number token, currently calculated number is pushed to the stack. Otherwise, jumps to process.

```
cmp al,32                   ;controls if incoming character is a "space" or not
jne not_space
cmp di,1                    ;controls "number reading" flag
jne process
mov di,0                    ;sets di to 0 (number token ended)
push bx                     ;pushes number to stack
mov bx,00                   ;sets bx back to 00
jmp process
not_space:
```

OPERATIONS

If the incoming character is an operation handles depending on the type of the operation.

ADD

- First, pops two numbers and places them into bx and dx registers.
- Subsequently, sums them up.
- After that, pushes result that is in bx register.

```
cmp al,43                   ;Controls if its an addition operation or not
jne not_add
pop bx                      ;pops two numbers
pop dx
add bx,dx                   ;sums them up
push bx                     ;pushes result to the stack
mov bx,00                   ;sets bx back to 00
jmp process
not_add:
```

MUL

- First, pops two numbers and places them into bx and ax registers.
- Multiplies those two numbers.
- Then pushes result that is in the ax register.

```
cmp al,42                   ;Controls if its an multiplication operation or not
jne not_mul
pop bx                      ;pops two numbers
pop ax
mul bx                      ;multiplies those numbers
push ax                     ;pushes result to the stack
mov bx,00                   ;sets bx back to 00
jmp process
not_mul:
```

DIV

- First, pops one number (which is the divisor) and places it into bx.
- Then, pops another number (which is the divident) and places it into ax.
- After the call of div operation with bx, quotient and remainder appears at ax and dx registers respectively.
- Since we are aiming to integer division, result in the ax register is pushed to stack.

```
cmp al,47                   ;Controls if its an division operation or not
jne not_div
pop bx                      ;pops divisor
pop ax                      ;pops divident
div bx                      ;divides divident to divisor
push ax                     ;pushes quotient (result) to stack
mov bx,00                   ;sets bx back to 00
jmp process
not_div:
```

OR

- First, pops two numbers and places them into ax and bx registers.
- Subsequently, applies bitwise or operation on these two numbers.
- After that, pushes result that is in ax register.

```
cmp al,124                  ;Controls if its an xor operation or not
jne not_or
pop bx                      ;pops two numbers
pop ax
or ax,bx                    ;applies or operation
push ax                     ;pushes result to the stack
mov bx,00                   ;sets bx back to 00
jmp process
not_or:
```

XOR

- First, pops two numbers and places them into ax and bx registers.
- Subsequently, applies bitwise xor operation on these two numbers.
- After that, pushes result that is in ax register.

```
cmp al,'^'                  ;Controls if its an xor operation or not
jne not_xor
pop bx                      ;pops two numbers
pop ax
xor ax,bx                   ;applies xor operation
push ax                     ;pushes result to the stack
mov bx,00                   ;sets bx back to 00
jmp process
not_xor:
```

AND

- First, pops two numbers and places them into ax and bx registers.
- Subsequently, applies bitwise and operation on these two numbers.
- After that, pushes result that is in ax register.

```
cmp al,38                   ;Controls if its an and operation or not
jne not_and
pop bx                      ;pops two numbers
pop ax
and ax,bx                   ;applies and operation
push ax                     ;pushes result to the stack
mov bx,00                   ;sets bx back to 00
jmp process
not_and:
```

NUMBER CONVERSION

Characters are processed differently whether its between 0-9 or A-F.

HEXADECIMAL

This part handles characters that are A to F.

- First, subtracts ‘A’ from the character, and adds 10 to it, by doing so hexadecimal value of character is derived.
- Then multiplies pre-calculated number by 16 to create a space for the new decimal.
- Adds the number derived from the ASCII character.
- And stores the current number at bx register.

```
cmp al,70                   ;Controls if character in the interval of A-F or not
jg not_hexa
cmp al,64
jle not_hexa
mov ah,00h
sub ax,'A'                  ;Deriving numerical value from the ASCII
add ax,10
mov dx,ax                   ;register manuevers
mov ax,bx
mov bx,dx
mov cx,16                   ;setting cx to 16
mul cx                      ;creating a space for new digit
mov cx,10                   ;setting cx back to 10
add ax,bx                   ;add new digit
mov bx,ax                   ;store current number at bx register
mov di,1                    ;sets "number reading" flag to 1
jmp process
not_hexa:
```

DECIMAL

- First strips the ASCII value.
- Then multiplies pre-calculated number by 16 to create a space for the new decimal.
- Adds the number derived from the ASCII character to pre-calculated one.
- And stores the current number at bx register.

```
sub al,'0'                  ;striping ASCII
mov ah,00h
mov dx,ax                   ;register maneuvers
mov ax,bx
mov bx,dx
mul cx                      ;creating a space for the new digit
add ax,bx                   ;add new digit
mov bx,ax                   ;store current number at bx register
mov di,1                    ;sets "number reading" flag to 1
jmp process
```

OUTPUT

```
output:
pop ax                      ;pops the number(the last number) from the stack
push 0                      ;pushes 0 to stack
```

DECIMAL TO HEXADECIMAL

- Takes 4 bits of the final number (that is corresponding to last digit of hexa decimal form)
- Derives corresponding ASCII character.
- Pushes character to the stack.
- Then, shifts 4 bits to the right (dividing by 16) to remove processed digit
- Finally, controls if all the number been processed.

```
out_stack:
    mov dx, ax              ;creates a copy of the value that is in ax
    and dx, 0fh             ;takes last 4 bits of the number (last digit in hexadecimal)
    mov si, dx              ;assigns dx to si
    mov dl, [convert + si]  ;derives corresponding ASCII character
    push dx                 ;pushes this character to stack

    shr ax, 4               ;shifts 4 bits to the right (divides by 16)
    cmp ax,0                ;checks if number is processed completely
    jne out_stack
```

PRINT

Pops and prints all characters in stack one by one until it encounters the 0 that is pushed at the begining of the output process.

```
print:
    pop dx                  ;pops one character from the stack
    cmp dx,0                ;controls if it is zero, if so printing process ends
    je exit
    mov ah,02h              ;print to console
    int 21h
    jmp print
```

INPUT AND OUTPUT

How to execute

- First step to run the code is mounting C drive.

```
Z:>mount c C:\8086
Drive C is mounted as local directory C:\8086\
```

- After that, switch to c drive.

```
Z:>mount c C:\8086
Drive C is mounted as local directory C:\8086\

Z:\>C:
```

- Then, via a86 command create .com and .sym files.

```
Z:>mount c C:\8086
Drive C is mounted as local directory C:\8086\

Z:\>C:

C:\>a86 TEST.ASM
A86 macro assembler, V4.05 Copyright Eric Isaacson
Source:
TEST.ASM
Object: TEST.COM
SYMBOLS: TEST.SYM
```

- Finally, you can execute the code. At this point, it waits for the input.

```
Z:>mount c C:\8086
Drive C is mounted as local directory C:\8086\

Z:\>C:

C:\>a86 TEST.ASM
A86 macro assembler, V4.05 Copyright Eric Isaacson
Source:
TEST.ASM
Object: TEST.COM
SYMBOLS: TEST.SYM

C:\>TEST

```

Examplary Input

```
2 3 + 4 5 + * 2 /
```

First character is 2. Derives numerical corresponding from ASCII character and multiplies bx by (0*16=0) 16 to open a room for the new digit. Subsequently, adds 2 to the bx (0+2=2).
Then encounters with space and pushes bx (2) to stack. Then, sets bx to 00.

```
Stack:[2
```

Next character is 3. Derives numerical corresponding from ASCII character and multiplies bx by (0*16=0) 16 to open a room for the new digit. Subsequently, adds 3 to the bx (0+3=3).
Then encounters with space and pushes bx (3) to stack. Then, sets bx to 00.

```
Stack:[2,3
```

Next character is +. So pops two numbers (3,2) from the stack and sum them up. Afterwards pushes the result to the stack (which is 5). Then encounters with space character and continues to read characters.

```
Stack:[5
```

Next character is 4. Derives numerical corresponding from ASCII character and multiplies bx by (0*16=0) 16 to open a room for the new digit. Subsequently, adds 4 to the bx (0+4=4).
Then encounters with space and pushes bx (4) to stack. Then, sets bx to 00.

```
Stack:[5,4
```

Next character is 5. Derives numerical corresponding from ASCII character and multiplies bx by (0*16=0) 16 to open a room for the new digit. Subsequently, adds 5 to the bx (0+5=5).
Then encounters with space and pushes bx (5) to stack. Then, sets bx to 00.

```
Stack:[5,4,5
```

Next character is +. So pops two numbers (5,4) from the stack and sum them up. Afterwards pushes the result to the stack (which is 9). Then encounters with space character and continues to read characters.

```
Stack:[5,9
```

Next character is *. So pops two numbers (9,5) from the stack and multiplies these two number. Afterwards pushes the result to the stack (which is 45).
Then encounters with space character and continues to read characters.

```
Stack:[45
```

Next character is 2. Derives numerical corresponding from ASCII character and multiplies bx by (0*16=0) 16 to open a room for the new digit. Subsequently, adds 2 to the bx (0+2=2).
Then encounters with space and pushes bx (2) to stack. Then, sets bx to 00.

```
Stack:[45,2
```

Next character is /. So pops two numbers (2,45) from the stack (former is divisor latter is divident). Then divides second one to first one. Afterwards pushes the result (quotient) to the stack (which is 22).
Then encounters with enter jumps to output stage.

```
Stack:[22
```

Firstly, last value in the stack is popped and a ‘0’ inserted (it will remaind us when to stop). A copy of the last value is created. Then bitwise and operation applied by 0fh (it derives numerical value of the last digit in hexadecimal form). Then calculated last digit is pushed into stack.After that, output value divided by 16 to remove the processed digit.

```
Stack:[48,6
```

And these procedure keeps going until the all the digits are processed.

```
Stack:[48,6,1
```

Finally, pops and prints all values in the stack until it encounters with 48.

```
Output:16
```