A86 Postfix Evaluator

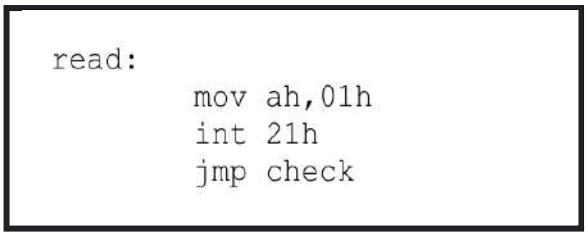
Ömer Faruk Çelik

This A86 assembly language program is evaluating postfix expressions involving hexadecimal quantities given as input.

Processes

1- Reading Phase
2- Checking Phase
3- Processing Phase
4- Printing The Result Phase

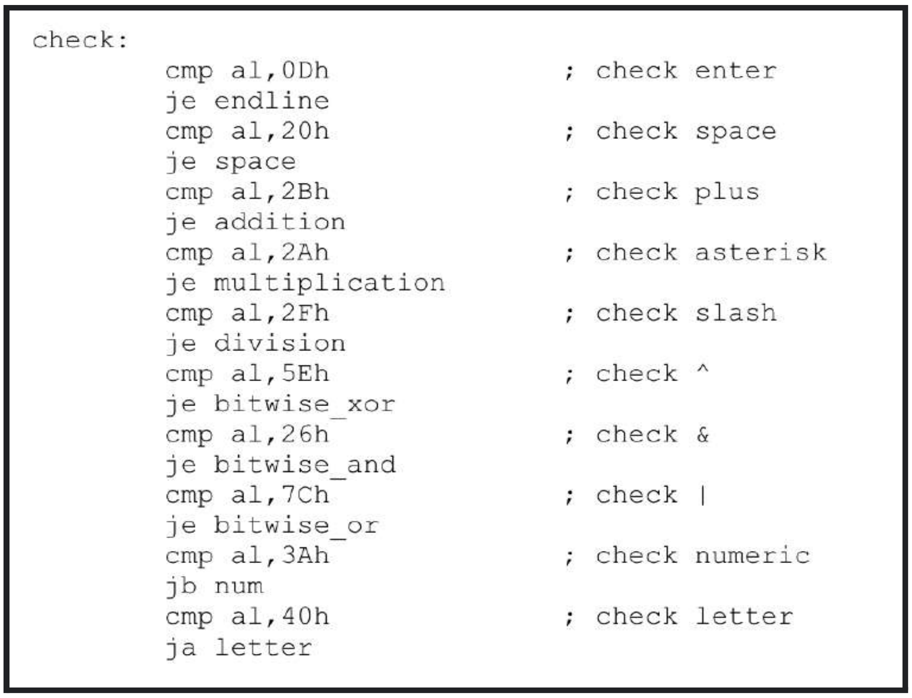
1- Reading



```
read:
        mov ah,01h
        int 21h
        jmp check
```

At this stage, the input character is read and jumped to the check part.

2- Checking



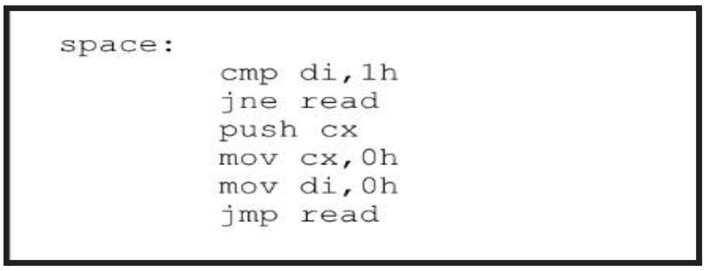
```
check:
        cmp al,0Dh                  ; check enter
        je endline
        cmp al,20h                  ; check space
        je space
        cmp al,2Bh                  ; check plus
        je addition
        cmp al,2Ah                  ; check asterisk
        je multiplication
        cmp al,2Fh                  ; check slash
        je division
        cmp al,5Eh                  ; check ^
        je bitwise_xor
        cmp al,26h                  ; check &
        je bitwise_and
        cmp al,7Ch                  ; check |
        je bitwise_or
        cmp al,3Ah                  ; check numeric
        jb num
        cmp al,40h                  ; check letter
        ja letter
```

At this stage, the read character is compared with the ascii codes and jumped to the relevant section.

3- Processing

In this phase, the read inputs are processed.

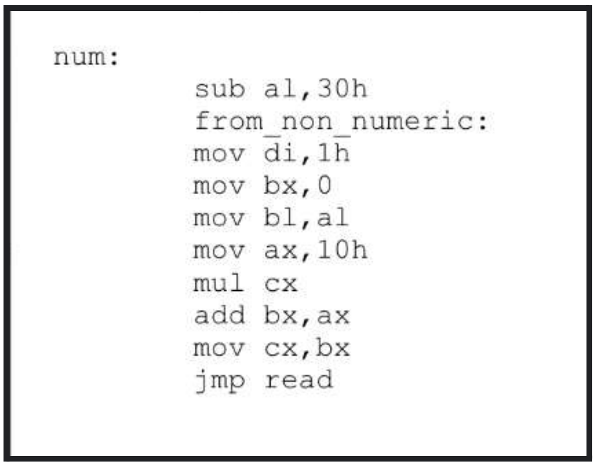
a) Space



```
space:
        cmp di,1h
        jne read
        push cx
        mov cx,0h
        mov di,0h
        jmp read
```

- Thanks to the di register, it is checked whether the last read input is an operator or an operand.
- If the character read is an operator, the reading is continued, otherwise the operand is pushed to the stack.

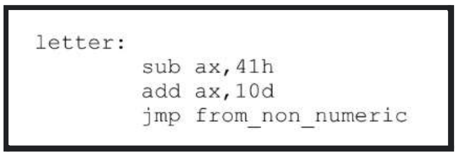
b) Num



```
num:
        sub al,30h
        from_non_numeric:
        mov di,1h
        mov bx,0
        mov bl,al
        mov ax,10h
        mul cx
        add bx,ax
        mov cx,bx
        jmp read
```

- Numeric value represented by number is converted from ascii value to numeric value.
- At the “from_non_numeric” part firstly with the di register, it is specified that the value read is the operand.
- Read value is copied to register bl.
- If another digit is read before, that digit is multiplied by 10h and the result is kept in register ax. Ax would be 0 if there are no digits before.
- The bx and ax values are added to the cx register.
- The cx register is responsible for storing the read value.
- Continue reading.

c) Letter



```
letter:
        sub ax,41h
        add ax,10d
        jmp from_non_numeric
```

- The ascii value of the numeric value represented by a letter is converted to a numeric value.
- Jump to “from_non_numeric”.

d) Addition

```
addition:
        mov di,0h
        pop ax
        pop bx
        add ax,bx
        push ax
        jmp read
```

- It is specified that the value read with the di register is an operator.
- The last 2 values in the stack are popped.
- The addition operation is performed
- The result is pushed to the stack
- Continue reading

e) Multiplication

```
multiplication:
        mov di,0h
        pop bx
        pop ax
        mul bx
        push ax
        jmp read
```

- It is specified that the value read with the di register is an operator.
- The last 2 values in the stack are popped.
- The multiplication operation is performed
- The result is pushed to the stack
- Continue reading

f) Division

```
division:
        mov di,0h
        pop bx
        pop ax
        div bx
        push ax
        jmp read
```

- It is specified that the value read with the di register is an operator.
- The last 2 values in the stack are popped.
- The division operation is performed
- The result is pushed to the stack
- Continue reading

g) Bitwise_xor

```
bitwise_xor:
        mov di,0h
        pop ax
        pop bx
        xor ax,bx
        push ax
        jmp read
```

- It is specified that the value read with the di register is an operator.
- The last 2 values in the stack are popped.
- The bitwise xor operation is performed
- The result is pushed to the stack
- Continue reading

h) Bitwise_and

```
bitwise_and:
        mov di,0h
        pop ax
        pop bx
        and ax,bx
        push ax
        jmp read
```

- It is specified that the value read with the di register is an operator.
- The last 2 values in the stack are popped.
- The bitwise and operation is performed
- The result is pushed to the stack
- Continue reading

i) Bitwise_or

```
bitwise_or:

        mov di,0h
        pop ax
        pop bx
        or ax,bx
        push ax
        jmp read
```

- It is specified that the value read with the di register is an operator.
- The last 2 values in the stack are popped.
- The bitwise or operation is performed
- The result is pushed to the stack
- Continue reading

j) Endline

```
endline:
        mov ah,02h
        mov dl,0Dh
        int 21h
        mov dl,0Ah
        int 21h
        mov cx,4h
        pop ax
        jmp handleresult
```

- Print ‘\r’.
- Print ‘\n’.
- 4 is assigned to the cx register in order to track the result digits.
- Pop result.
- Jump to “handleresult”

4- Printing The Result

```
handleresult:
            mov dx,0
            mov bx,10h
            div bx
            cmp dx,0Ah
            jb numtoasci
            add dl,41h
            sub dl,10d
            jmp fromletter


numtoasci:
            add dl,30h
fromletter:
            push dx
            dec cx
            jnz handleresult
            jmp printandexit

printandexit:
            pop dx
            mov ah,02h
            int 21h
            pop dx
            int 21h
            pop dx
            int 21h
            pop dx
            int 21h
            int 20h
```

- Dividing the result by 10h, we get the digits one by one as the remainder.
- Convert digit to its ascii value.
- Push ascii values to the stack.
- When all digits are processed jump to “printandexit”.
- Pop and print 4 digits.
- Exit the program.