COA PROJECT

# Design and implement an arbitrary precision four function calculator.

**Abstraction:**

* + Microprocessors and their applications course are considered as a significant core course for electrical engineering students due to its potential impact into several reallife applications such as complex calculations, interfacing, control and automation technology.
  + A simple **Calculator** was performed using the Assembly language of 8086, which performs algebraic functions such as addition, subtraction, division and multiplication. It makes calculations easier and faster.

# Working of Calculator:

**Step 1:** Input first number from the user

**Step 2:** Choose the appropriate operator (+,-,\*,/) as of your choice.

**Step 3:** Now input the Second number.

# Step 4:

* If you select the Addition Operator(+), addition of numbers will be calculated.
* If you select the Substraction Operator(-) difference of the numbers will be calculated.
* If you select the Multiplication Operator(\*) ,Multiplication of numbers will be calculated.
* If you select the Division Operator(/) . Quotient of the two numbers will be calculated.

**Step 5:** Finally, Output is displayed.

**Note:** This Calculator works for integer values only.

# Code:

name "calc2"

; command prompt based simple calculator (+,-,\*,/) for 8086.

; example of calculation:

; input 1 <- number: 10

; input 2 <- operator:

- ; input 3 <- number: 5

;

; 10 - 5 = 5

; output -> number: 5

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; this maro is copied from emu8086.inc ;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; this macro prints a char in AL and advances ; the current cursor position: PUTC MACRO char

PUSH AX

MOV AL, char MOV AH, 0Eh INT 10h

POP AX

ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

org 100h jmp start ; define variables:

msg0 db "note: calculator works with integer values only.",0Dh,0Ah db "to learn how to output the result of a float division see

float.asm in examples",0Dh,0Ah,'$' msg1 db 0Dh,0Ah,

0Dh,0Ah, 'enter first number: $'

msg2 db "enter the operator: + - \* / : $" msg3 db "enter second number: $"

msg4 db 0dh,0ah , 'the approximate result of my calculations is : $' msg5 db 0dh,0ah ,'thank you for using the calculator! press any key... ', 0Dh,0Ah, '$' err1 db "wrong operator!", 0Dh,0Ah , '$' smth db " and something $"

; operator can be: '+','-','\*','/' or 'q' to exit in the middle. opr db '?'

; first and second number: num1 dw ? num2 dw ?

start:

mov dx, offset msg0 mov ah, 9 lea dx, msg1

mov ah, 09h ; output string at ds:dx int 21h

; get the multi-digit signed number

; from the keyboard, and store ; the result in cx register:

call scan\_num

; store first number: mov num1, cx ; new line: putc 0Dh putc 0Ah lea dx, msg2 mov ah, 09h

; output string at ds:dx ; get operator: mov ah, 1 ; single char input to AL.

int 21h mov opr, al ; new line: putc 0Dh putc 0Ah

cmp opr, 'q' ; q - exit in the middle. je exit

cmp opr, '\*' jb wrong\_opr cmp opr, '/'

ja wrong\_opr

; output of a string at ds:dx lea dx, msg3 mov ah, 09h

int 21h

; get the multi-digit signed number

; from the keyboard, and store ; the result in cx register

call scan\_num ; store second number: mov num2, cx lea dx, msg4 mov ah, 09h ; output string at ds:dx int 21h ; calculate: cmp opr, '+' je do\_plus cmp opr, '-'

je do\_minus cmp opr, '\*' je

do\_mult cmp opr, '/' je do\_div ; none

of the above....

wrong\_opr:

lea dx, err1 mov ah, 09h ; output string at ds:dx int 21h exit:

; output of a string at ds:dx lea dx, msg5 mov ah, 09h

int 21h ; wait for any key...

mov ah, 0 int 16h ret ; return back to os. do\_plus: mov ax, num1 add ax, num2

call print\_num ; print ax value. jmp exit

do\_minus:

mov ax, num1 sub ax, num2

call print\_num ; print ax value. jmp exit

do\_mult:

mov ax, num1

imul num2 ; (dx ax) = ax \* num2. call print\_num ; print ax value.

; dx is ignored (calc works with tiny numbers only). jmp exit

do\_div:

; dx is ignored (calc works with tiny integer numbers only). mov dx, 0

mov ax, num1

idiv num2 ; ax = (dx ax) / num2. cmp dx, 0 jnz approx call print\_num ; print ax

value. jmp exit approx:

call print\_num ; print ax value. lea dx, smth mov ah, 09h

; output string at ds:dx int 21h jmp exit

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; these functions are copied from emu8086.inc ;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; gets the multi-digit SIGNED number from the keyboard,

; and stores the result in CX register: SCAN\_NUM PROC NEAR

PUSH DX PUSH AX PUSH SI

MOV CX, 0

; reset flag:

MOV CS:make\_minus, 0

next\_digit:

; get char from keyboard

; into AL:

MOV AH, 00h INT 16h

; and print it: MOV AH, 0Eh INT 10h

; check for MINUS: CMP AL, '-'

JE set\_minus

; check for ENTER key:

CMP AL, 0Dh ; carriage return? JNE not\_cr

JMP stop\_input not\_cr:

CMP AL, 8 ; 'BACKSPACE' pressed?

JNE backspace\_checked

MOV DX, 0 ; remove last digit by

MOV AX, CX ; division:

DIV CS:ten ; AX = DX:AX / 10

(DX-rem).

MOV CX, AX

PUTC ' ' ; clear position.

PUTC 8 ; backspace again. JMP next\_digit backspace\_checked:

; allow only digits:

CMP AL, '0'

JAE ok\_AE\_0 JMP remove\_not\_digit ok\_AE\_0:

CMP AL,

'9' JBE

ok\_digit remove\_not\_digit:

PUTC 8 ; backspace.

PUTC ' ' ; clear last entered not digit. PUTC 8 ; backspace again.

JMP next\_digit ; wait for next input. ok\_digit:

; multiply CX by 10 (first time the result is zero) PUSH AX

MOV AX, CX

MUL CS:ten ; DX:AX = AX\*10 MOV CX, AX

POP AX

; check if the number is too big

; (result should be 16 bits) CMP DX, 0

JNE too\_big

; convert from ASCII code: SUB AL, 30h

; add AL to CX:

MOV AH, 0

MOV DX, CX ; backup, in case the result will be too big. ADD CX, AX

JC too\_big2 ; jump if the number is too big.

JMP next\_digit

set\_minus:

MOV CS:make\_minus, 1 JMP next\_digit

too\_big2:

MOV CX, DX ; restore the backuped value before add.

MOV DX, 0 ; DX was zero before backup! too\_big:

MOV AX, CX

DIV CS:ten ; reverse last DX:AX = AX\*10, make AX = DX:AX / 10 MOV CX, AX

PUTC 8 ; backspace.

PUTC ' ' ; clear last entered digit. PUTC 8 ; backspace again.

JMP next\_digit ; wait for Enter/Backspace. stop\_input:

; check flag:

CMP CS:make\_minus, 0

JE not\_minus NEG CX

not\_minus:

POP SI

POP AX POP

DX RET

make\_minus DB ? ; used as a flag. SCAN\_NUM ENDP

; this procedure prints number in AX,

; used with PRINT\_NUM\_UNS to print signed numbers: PRINT\_NUM PROC NEAR

PUSH DX PUSH AX CMP AX, 0

JNZ not\_zero

PUTC '0'

JMP printed

not\_zero:

; the check SIGN of AX,

; make absolute if it's negative: CMP AX, 0

JNS positive

NEG AX

PUTC '-'

positive:

CALL

PRINT\_NUM\_UNS printed: POP AX

POP DX

RET

PRINT\_NUM ENDP

; this procedure prints out an unsigned

; number in AX (not just a single digit)

; allowed values are from 0 to 65535 (FFFF) PRINT\_NUM\_UNS PROC NEAR

PUSH AX

PUSH BX PUSH CX PUSH DX

; flag to prevent printing zeros before number: MOV CX, 1

; (result of "/ 10000" is always less or equal to 9). MOV BX, 10000 ; 2710h - divider.

; AX is zero? CMP AX,

0 JZ

print\_zero

begin\_print:

; check divider (if zero go to end\_print): CMP BX,0

JZ end\_print

calc:

; avoid printing zeros before number:

CMP CX, 0

JE calc

; if AX<BX then result of DIV will be zero: CMP AX, BX

JB skip

MOV CX, 0 ; set flag. MOV DX, 0

DIV BX ; AX = DX:AX / BX (DX=remainder).

; print last digit

; AH is always ZERO, so it's ignored

ADD AL, 30h ; convert to ASCII code. PUTC AL

MOV AX, DX ; get remainder from

last div. skip:

; calculate BX=BX/10 PUSH AX

MOV DX, 0 MOV AX, BX

DIV CS:ten ; AX = DX:AX / 10 (DX=remainder). MOV BX, AX

POP AX

JMP begin\_print print\_zero:

PUTC '0'

end\_print:

POP DX

POP CX

POP BX

POP AX RET

PRINT\_NUM\_UNS ENDP

ten DW 10 ; used as multiplier/divider by SCAN\_NUM & PRINT\_NUM\_UNS. GET\_STRING PROC NEAR

|  |  |  |
| --- | --- | --- |
| PUSH PUSH PUSH  PUSH | AX CX DI  DX |  |
| MOV | CX, 0 | ; char counter. |
| CMP | DX, 1 | ; buffer too small? |
| JBE | empty\_buffer | ; |
| DEC | DX | ; reserve space for last zero. |

;============================

; Eternal loop to get ; and processes key presses wait\_for\_key:

MOV AH, 0 ; get pressed key. INT 16h

CMP AL, 0Dh ; 'RETURN' pressed? JZ exit\_GET\_STRING

CMP AL, 8 ; 'BACKSPACE' pressed?

JNE add\_to\_buffer

JCXZ wait\_for\_key ; nothing to remove!

|  |  |  |  |
| --- | --- | --- | --- |
| DEC  DEC | CX  DI |  | |
| PUTC | 8 | ; backspace. | |
| PUTC | ' ' | ; clear position. | |
| PUTC | 8 | ; backspace | |
| again. | JMP | wait\_for\_key | |
| add\_to\_buffer: | | | |
| CMP | | CX, DX | ; buffer is full? |
| JAE | | wait\_for\_key | ; if so wait for 'BACKSPACE' or 'RETURN'... |
| MOV | | [DI], AL |  |
| INC | | DI |  |
| INC | | CX |  |

; print the key: MOV AH, 0Eh INT 10h

JMP wait\_for\_key

;============================

exit\_GET\_STRING:

; terminate by null:

MOV [DI], 0

empty\_buffer:

POP DX

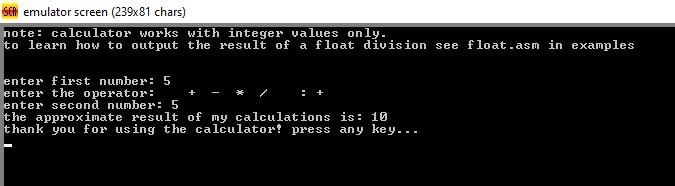
POP DI

POP CX

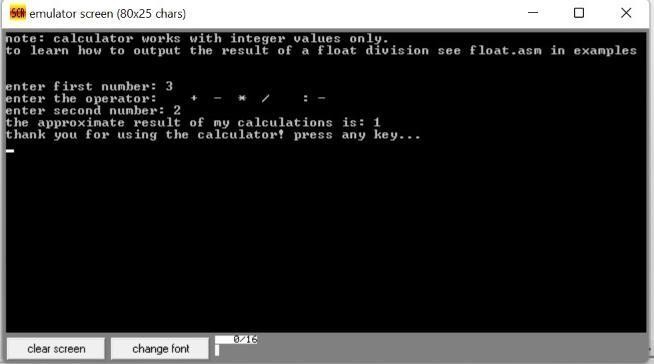
POP AX RET

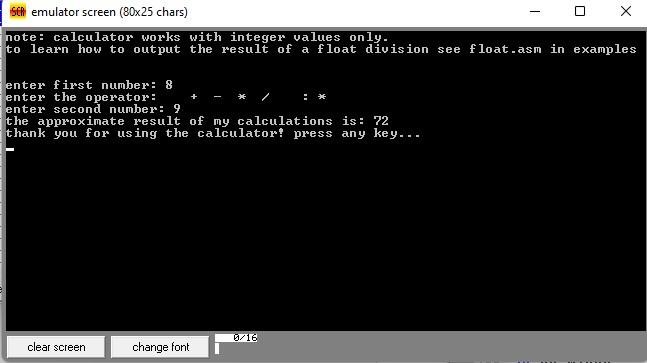
GET\_STRING ENDP

# Output:

**Output when Addition is choosed:**

# Output when Substraction is choosed:



**Output when Multiplication is used:**

# Output when division is choosed:

**Application** :

* This Calculator was very beneficial in enhancing the student' skills in mathematics, engineering and computer programming which can be employed in designing a useful application for users as well as the ability to apply numerical techniques .
* This calculator were designed over the virtual machine for Intel 8086 microprocessor using EMU8086 emulator software .
* It is a portable device that can use anywhere to perform simple mathematical operations. We use a scientific or sophisticated calculator in some situations, where we need to solve complex calculations like trigonometry functions, exponential.