

NAME: GINI CHACKO

SEMESTER: IV

CLASS: SE COMPS B

BATCH: B

ROLL: 8942

TOPIC: MP EXPERIMENT 2 :

Menu driven code for

16 bit ADDITION

16 bit SUBTRACTION

16 bit MULTIPLICATION

16 bit DIVISION

CODE:

```
.8086
.model small
.data
num1 dw 0506h
num2 dw 0106h
result dw ?
rem dw ?
quot dw ?
msg db 'Enter the options$'
msg1 db '1. Addition$'
msg2 db '2. Subtraction$'
msg3 db '3. Multiplication$'
msg4 db '4. Division$'
.code
start:
mov ax, @data
mov ds, ax
lea dx, msg
mov ah, 09h
int 21h
lea dx, msg1
mov ah, 09h
int 21h
lea dx, msg2
mov ah, 09h
int 21h
lea dx, msg3
mov ah, 09h
int 21h
lea dx, msg4
mov ah, 09h
int 21h
```

```
mov ah,08h  
int 21h
```

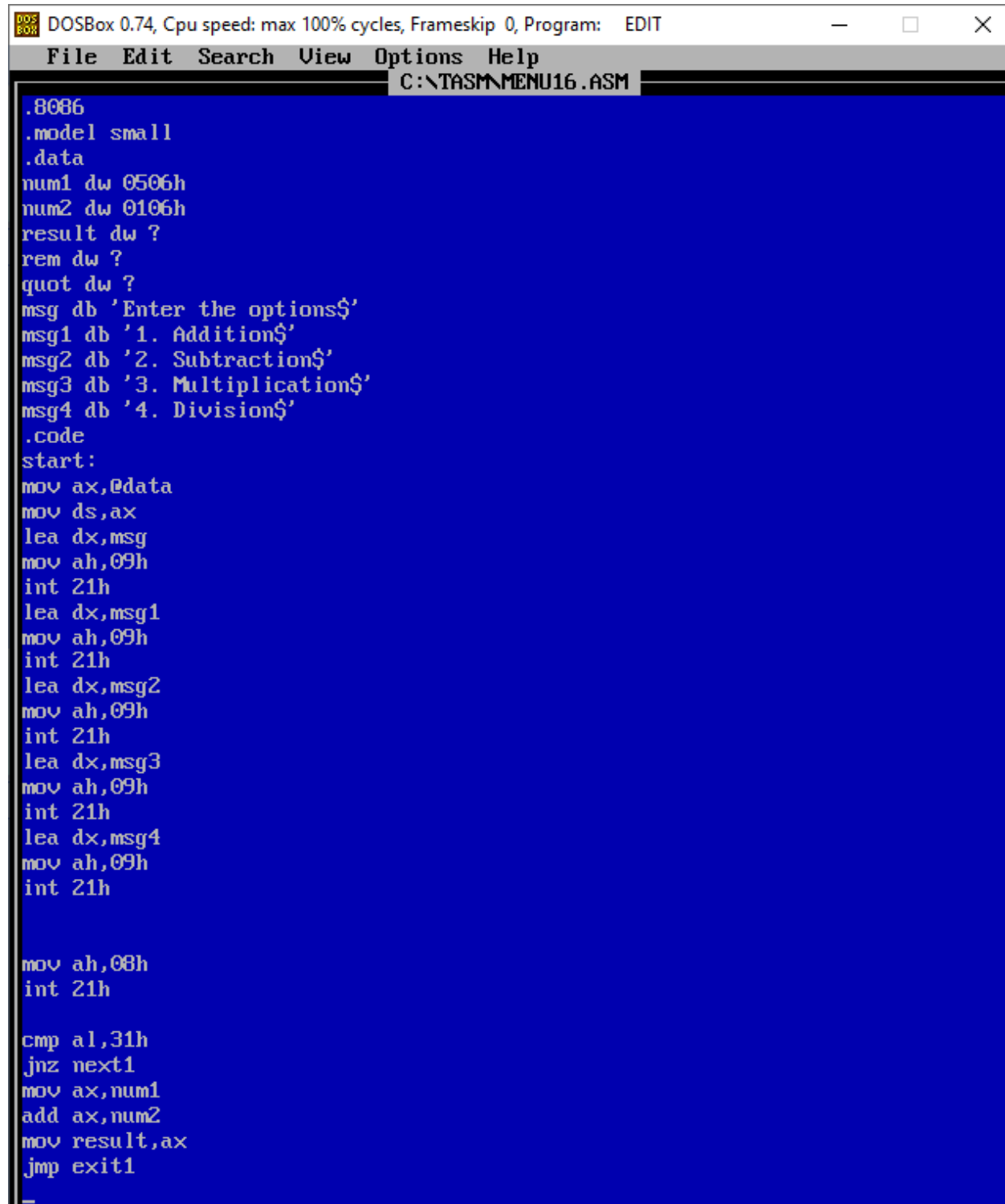
```
cmp al,31h  
jnz next1  
mov ax,num1  
add ax,num2  
mov result,ax  
jmp exit1
```

```
next1: cmp al,32h  
jnz next2  
mov ax,num1  
sub ax,num2  
mov result,ax  
jmp exit1
```

```
next2: cmp al,33h  
jnz next3  
mov ax,num1  
mov bx,num2  
mul bx  
mov result,ax  
mov ax,dx  
jmp exit1
```

```
next3: cmp al,34h  
jnz exit1  
mov ax,0000h  
mov bx,0000h  
mov ax,num1  
mov bx,num2  
div bx  
mov quot,ax  
mov rem,dx
```

```
exit1:int 3h
int 21h
end start
```

A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: EDIT". The menu bar includes "File", "Edit", "Search", "View", "Options", and "Help". The file name "C:\TASM\MENU16.ASM" is displayed in the title bar. The main window has a blue background and contains assembly code in white text. The code defines data for a menu, including numbers and messages for addition, subtraction, multiplication, and division. It includes a loop to display the menu and a section for performing addition based on user input.

```
.8086
.model small
.data
num1 dw 0506h
num2 dw 0106h
result dw ?
rem dw ?
quot dw ?
msg db 'Enter the options$'
msg1 db '1. Addition$'
msg2 db '2. Subtraction$'
msg3 db '3. Multiplication$'
msg4 db '4. Division$'
.code
start:
mov ax,@data
mov ds,ax
lea dx,msg
mov ah,09h
int 21h
lea dx,msg1
mov ah,09h
int 21h
lea dx,msg2
mov ah,09h
int 21h
lea dx,msg3
mov ah,09h
int 21h
lea dx,msg4
mov ah,09h
int 21h

mov ah,08h
int 21h

cmp al,31h
jnz next1
mov ax,num1
add ax,num2
mov result,ax
jmp exit1
```

```

next1: cmp al,32h
jnz next2
mov ax,num1
sub ax,num2
mov result,ax
jmp exit1

next2: cmp al,33h
jnz next3
mov ax,num1
mov bx,num2
mul bx
mov result,ax
mov ax,dx
jmp exit1

next3: cmp al,34h
jnz exit1
mov ax,0000h
mov bx,0000h
mov ax,num1
mov bx,num2
div bx
mov quot,ax
mov rem,dx

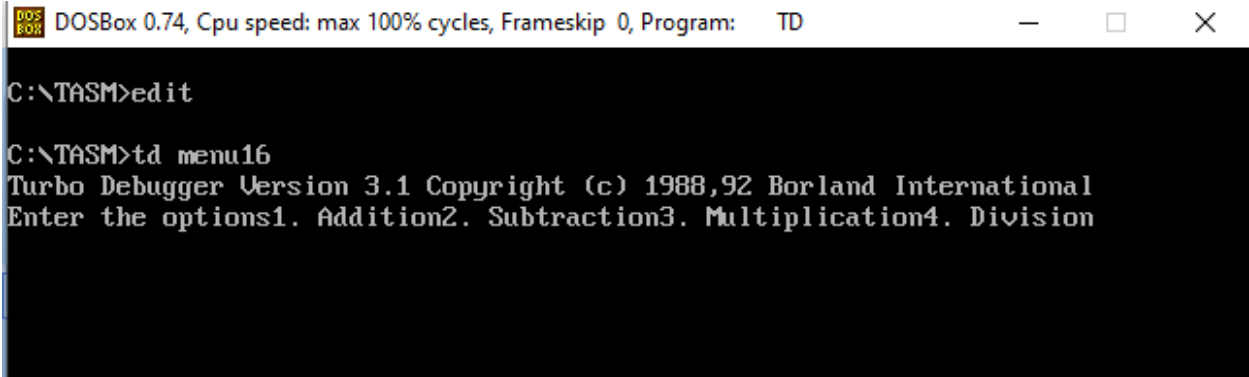
exit1: int 3h
int 21h
end start

```

F1=Help

Line:74 Col:1

OUTPUT:



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

```

C:\TASM>edit
C:\TASM>td menu16
Turbo Debugger Version 3.1 Copyright (c) 1988,92 Borland International
Enter the options1. Addition2. Subtraction3. Multiplication4. Division

```

1.Addition :

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options Window Help

CPU 80486

cs:007D CC	int 03	ax 060C	c=0
cs:007E CD21	int 21	bx 0000	z=0
cs:0080 06	push es	cx 0000	s=0
cs:0081 050601	add ax,0106	dx 0049	o=0
cs:0084 0C06	or al,06	si 0000	p=1
cs:0086 0000	add [bx+sil,al]	di 0000	a=0
cs:0088 0000	add [bx+sil,al]	bp 0000	i=1
cs:008A 45	inc bp	sp 0000	d=0

[F7]-Dump 2=[F7][F7]

ds:0000 06 05 06 01 0C 06 00 00 Enter

ds:0008 00 00 45 6E 74 65 72 20 the opti

ds:0010 74 68 65 20 6F 70 74 69 ons\$1. A

ds:0018 6F 6E 73 24 31 2E 20 41

es:0000 CD 20 FF 9F 00 EA FF FF = f

es:0008 AD DE E0 01 C5 15 AA 01 ; [ax+sil]

es:0010 C5 15 89 02 20 10 92 01 ; [bx+sil]

es:0018 01 03 01 00 02 FF FF FF

ss:0002 6474

ss:0000 0000

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

2.Subtraction :

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options Window Help

CPU 80486

cs:007D CC	int 03	ax 0400	c=0
cs:007E CD21	int 21	bx 0000	z=0
cs:0080 06	push es	cx 0000	s=0
cs:0081 050601	add ax,0106	dx 0049	o=0
cs:0084 0004	add [sil,al]	si 0000	p=1
cs:0086 0000	add [bx+sil,al]	di 0000	a=0
cs:0088 0000	add [bx+sil,al]	bp 0000	i=1
cs:008A 45	inc bp	sp 0000	d=0

[F7]-Dump 2=[F7][F7]

ds:0000 06 05 06 01 00 04 00 00 Enter

ds:0008 00 00 45 6E 74 65 72 20 the opti

ds:0010 74 68 65 20 6F 70 74 69 ons\$1. A

ds:0018 6F 6E 73 24 31 2E 20 41

es:0000 CD 20 FF 9F 00 EA FF FF = f

es:0008 AD DE E0 01 C5 15 AA 01 ; [ax+sil]

es:0010 C5 15 89 02 20 10 92 01 ; [bx+sil]

es:0018 01 03 01 00 02 FF FF FF

ss:0002 6474

ss:0000 0000

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

3. Multiplication :

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options Window Help READY

CPU 80486

cs:007D CC	int	03	ax	0005	c=1
cs:007E CD 21	int	21	bx	0106	z=0
cs:0080 06	push	es	cx	0000	s=0
cs:0081 050601	add	ax,0106	dx	0005	o=1
cs:0084 2424	and	al,24	si	0000	p=1
cs:0086 0000	add	[bx+sil,al	di	0000	a=0
cs:0088 0000	add	[bx+sil,al	bp	0000	i=1
cs:008A 45	inc	bp	sp	0000	d=0
cs:008B 6E	[]=Dump 2=[]				
cs:008C 7465	ds:0000 06 05 06 01 24 24 00 00 \$\$\$				
cs:008E 7220	ds:0008 00 00 45 6E 74 65 72 20 Enter				
cs:0090 7468	ds:0010 74 68 65 20 6F 70 74 69 the opti				
cs:0092 65206F	ds:0018 6F 6E 73 24 31 2E 20 41 ons\$1. A				

es:0000 CD 20 FF 9F 00 EA FF FF = f	ss:0002 6474
es:0008 AD DE E0 01 C5 15 AA 01 i	ss:0000 0000
es:0010 C5 15 89 02 20 10 92 01 +Se	
es:0018 01 03 01 00 02 FF FF FF	

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

4. Division :

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options Window Help READY

CPU 80486

cs:007D CC	int	03	ax	4758	c=0
cs:007E CD 21	int	21	bx	0106	z=1
cs:0080 06	push	es	cx	0000	s=0
cs:0081 050601	add	ax,0106	dx	00F6	o=0
cs:0084 0000	add	[bx+sil,al	si	0000	p=1
cs:0086 F60058	test	byte ptr [bx+	di	0000	a=0
cs:0089 47	inc	di	bp	0000	i=1
cs:008A 45	inc	bp	sp	0000	d=0
cs:008B 6E	[]=Dump 2=[]				
cs:008C 7465	ds:0000 06 05 06 01 00 00 F6 00 \$\$\$				
cs:008E 7220	ds:0008 58 47 45 6E 74 65 72 20 XGEnter				
cs:0090 7468	ds:0010 74 68 65 20 6F 70 74 69 the opti				
cs:0092 65206F	ds:0018 6F 6E 73 24 31 2E 20 41 ons\$1. A				

es:0000 CD 20 FF 9F 00 EA FF FF = f	ss:0002 6474
es:0008 AD DE E0 01 C5 15 AA 01 i	ss:0000 0000
es:0010 C5 15 89 02 20 10 92 01 +Se	
es:0018 01 03 01 00 02 FF FF FF	

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

POSTLABS

1. Explain registers of 8086.

ANS: In 8086 Microprocessor, the registers are categorized into mainly four types:

- 1) General Purpose Registers
- 2) Segment Registers
- 3) Pointers and Index Registers
- 4) Flag or Status Register

1) General Purpose Registers - The use of general-purpose register is to store temporary data. While the instructions are executed in the control unit, they may work on some numeric value or some operands. These need to be stored somewhere so that the processor can operate on them easily. So, these registers are used in these cases. There are 4 general purpose registers of 16-bit length each. Each of them is further divided into 2 subparts of 8 bits length each: one high, which stores the higher order bits and another low which stores the lower order bits.

- | | |
|-------------------|-------------------|
| 1) $AX = [AH:AL]$ | 3) $CX = [CH:CL]$ |
| 2) $BX = [BH:BL]$ | 4) $DX = [DH:DL]$ |

2) Segment Registers - There are 4 segment registers in 8086 microprocessor and each of them is of 16 bit. The code and instructions are stored inside these different segments.

- 1) Code Segment (CS) Register: The user cannot modify the content of these registers.
- 2) Data Segment (DS) Register: The user can modify the content of the data segment.
- 3) Stack Segment (SS) Register: The SS is used to store the information about the memory segment.

4) Extra Segment (ES) Register: If there is less space in that segment, then ES is used. ES is also used for copying purpose.

3) Pointers and Index Registers - The pointers will always store some address or memory location

1) Instruction Pointer (IP): The instruction pointer usually stores the address of the next instruction that is to be executed.

2) Base Pointer (BP): The base pointer stores the base address of the memory.

3) Stack Pointer (SP): The stack pointer points at the current top value of the stack.

4) Source Index (SI): It stores the offset address of the source.

5) Destination Index (DI): It stores the offset address of the destination.

4) Flag or Status Register - The flag or status register is a 16-bit register which contains 9 flags, and the remaining 7 bits are idle in this register. These flags tell about the status of the processor after any arithmetic or logical operation. If flag value is 1, the flag is set, and if it is 0, it is said to be reset.

2. Explain logical and physical address for 8086 with example.

ANS:

- Logical address is contained in the 16-bit IP, BP, SP, BX, SI or DI. It is also known as the offset address or the effective address.
- The physical address or the real address is formed by combining the offset and base segment addresses. This address is 20-bit and is primarily used for the accessing of the memory.
- The logical address is a virtual address and can be viewed by the user. The user can't view the physical address directly. The logical address is used like a reference, to access the physical address.
- The fundamental difference between logical and physical address is that logical address is generated by CPU during a program execution whereas, the physical address refers to a location in the memory unit.
- The logical address is generated by the CPU while physical address is computed by MMU.
- Identical logical address and physical address are generated by compile-time and load time address binding methods. The logical and physical address generated while run-time address binding method differs from each other.
- The set of all logical addresses generated by CPU for a program is called Logical Address Space. However, the set of all physical address mapped to corresponding logical addresses is referred as Physical Address Space.