Lab 9

Hooking Software Interrupts & Exceptions

After completing this lab

- Students will be able to hook interrupts.
- Students will be able to handle exceptions.

Interrupts

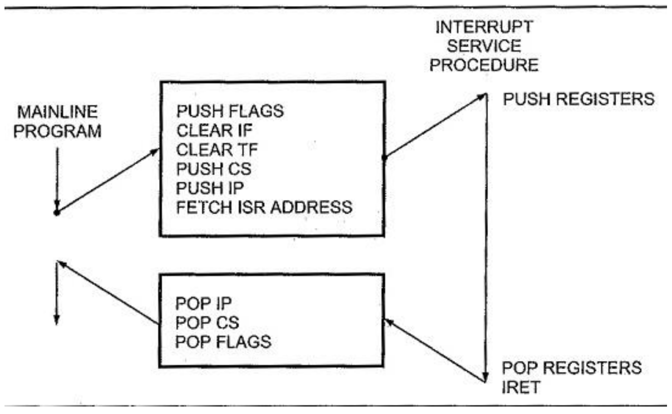
The processor uses the interrupt number to index the Interrupt Vector Table (IVT) to get the address of a special type of procedure called an Interrupt Service Routine (ISR). The ISR is executed whenever the interrupt is generated. There are 256 entries in IVT, and each entry stores the logical address of a respective ISR. IVT resides in memory at **physical address 0x00000**. Each entry stores the **logical address**, which is 4 bytes; the total size of the IVT is **256 x 4B = 1 KB**. To index the IVT, the processor multiplies the interrupt number by 4 to get the physical address of the entry against that interrupt number.

To hook an interrupt in this context means to change the ISR address of an interrupt in the Interrupt Vector Table. Before doing so, one needs to write ISR first and then add its logical address, consisting of Segment: Offset, to IVT.

ISR is a special procedure that is called whenever an interrupt is generated. Whenever an ISR is called as a result of an interrupt, the flag register is pushed onto the stack along with IP and CS. Therefore, unlike procedures, the ISRs return control by issuing "IRET" instructions instead of "RET" instructions. The IRET instruction also pops the flag register from the stack, whereas the RET instruction does not.

8086 Interrupt maps

When an interrupt is generated, the following steps are performed.



1. It decrements stack pointer by 2 and pushes the flag register on the stack.
2. It disables the INTR interrupt input by clearing the interrupt flag in the flag.
3. It resets the trap flag in the flag register.
4. It decrements stack pointer by 2 and pushes the current code segment register contents on the stack.
5. It decrements stack pointer by 2 and pushes the current instruction pointer contents on the stack.
6. It does an indirect jump at the start of the procedure by loading the CS and IP values for the start of the interrupt service routine (ISR).
7. An IRET instruction at the end of the interrupt service procedure returns execution to the main program.
8. The ISR should push registers to stack before using them and pop them before IRET.

INT#0
This interrupt is generated when a number is divided by zero or the result of division overflows. The following code handle this exception.

Example#1:
Program to write ISR for "**DIV BY Zero**" exception i.e., **INT#0** and hook it.

```
.model small

.data


.code

mov ax,@data
mov ds,ax

mov ax,0
mov es,ax

mov word ptr es:[0 * 4 + 0],isr0      ; offset address of ISR0
mov word ptr es:[ 0 * 4 + 2],cs        ; Segment address of ISR0


mov ax,100
mov bl,0

div bl                                  ;Dividing a number by zero to check if the hooking


.exit


ISR0 proc

jmp l@1
msg: db "Divide by Zero Exception!$"
l@1:
mov dx,offset msg
mov ah,9
int 0x21

IRET

ISR0 endp
```

INT#4
This interrupt is generated in response to an INTO instruction that checks the overflow flag to generate the interrupt. If the overflow flag is not set, **INT#4** will not be generated.

Example#2:
Program to hook "**OVERFLOW**" exception, i.e., **INT#4**.

```
.model small

.data


.code

mov ax,@data
mov ds,ax

mov ax,0
mov es,ax

mov word ptr es:[ 4 * 4 + 0],isr4
mov word ptr es:[ 4 * 4 + 2],cs


mov al,127
mov bl,1
add al,bl

into                    ;if OF is set, generate int#4



.exit


isr4 proc

jmp l@1
msg: db "Overflow flag is set$"
l@1:
mov dx,offset msg
mov ah,9
int 0x21

iret
isr4 endp
```

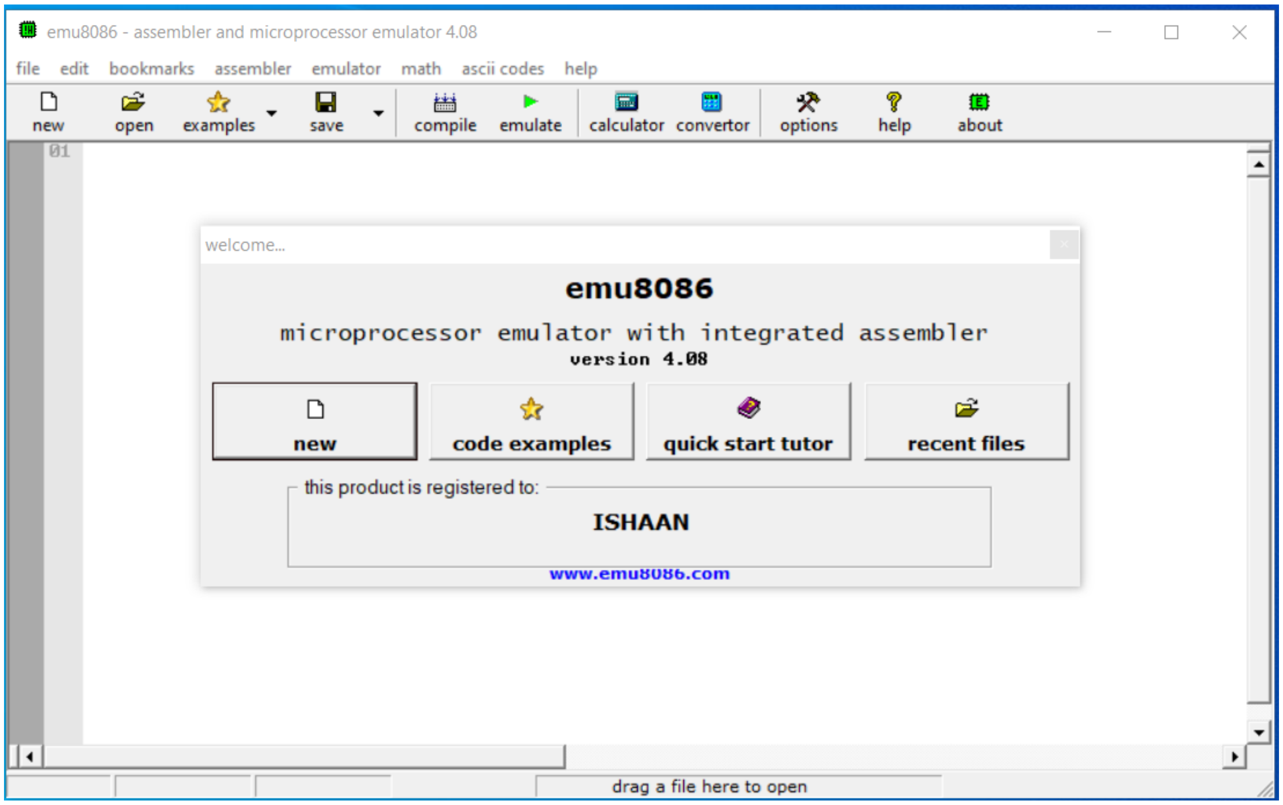
Computer Organization and Assembly Language

Emu8086 Tutorial Step by Step

Step-1

Double click on the icon on the desktop

Step-2
The following window will appear. Click on new.


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
welcome...
new
code examples
quick start tutor
recent files
drag a file here to open

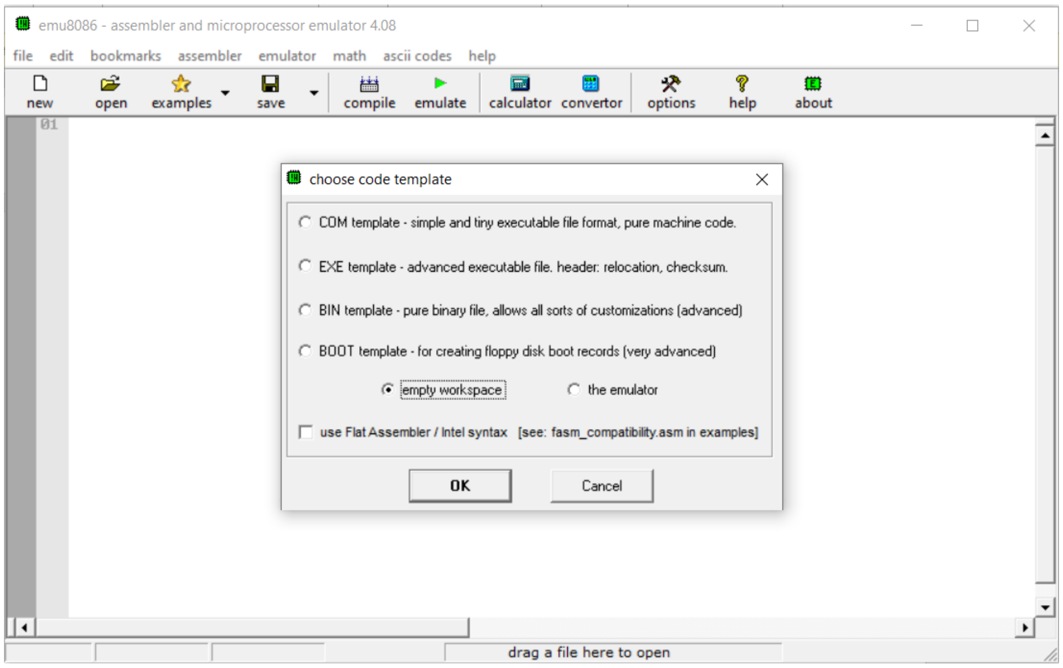
emu8086

microprocessor emulator with integrated assembler
version 4.08

this product is registered to:
ISHAAN

www.emu8086.com

Step-3:
Click on empty workspace and press OK.


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
drag a file here to open

choose code template

( ) COM template - simple and tiny executable file format, pure machine code.

( ) EXE template - advanced executable file. header: relocation, checksum.

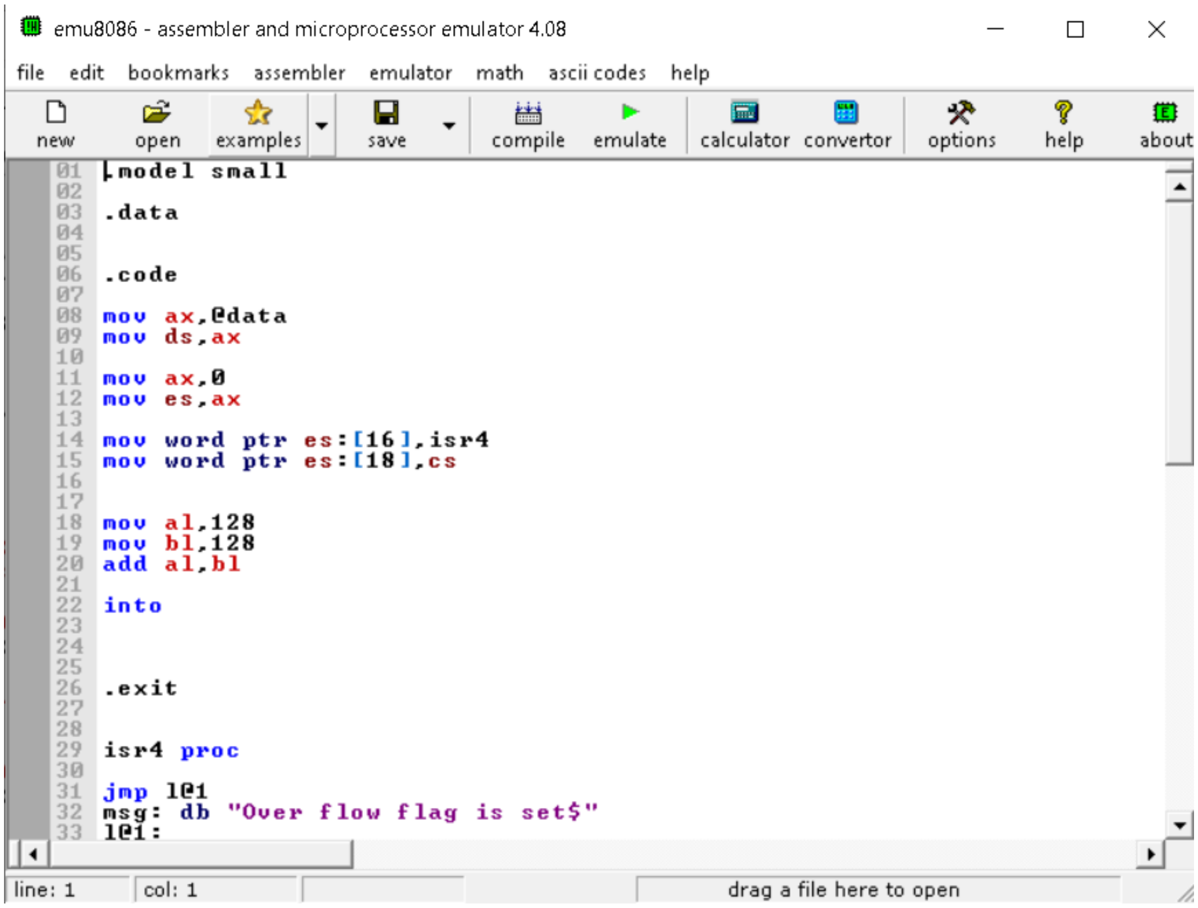
( ) BIN template - pure binary file, allows all sorts of customizations (advanced)

( ) BOOT template - for creating floppy disk boot records (very advanced)

[✓ empty workspace]   ( ) the emulator

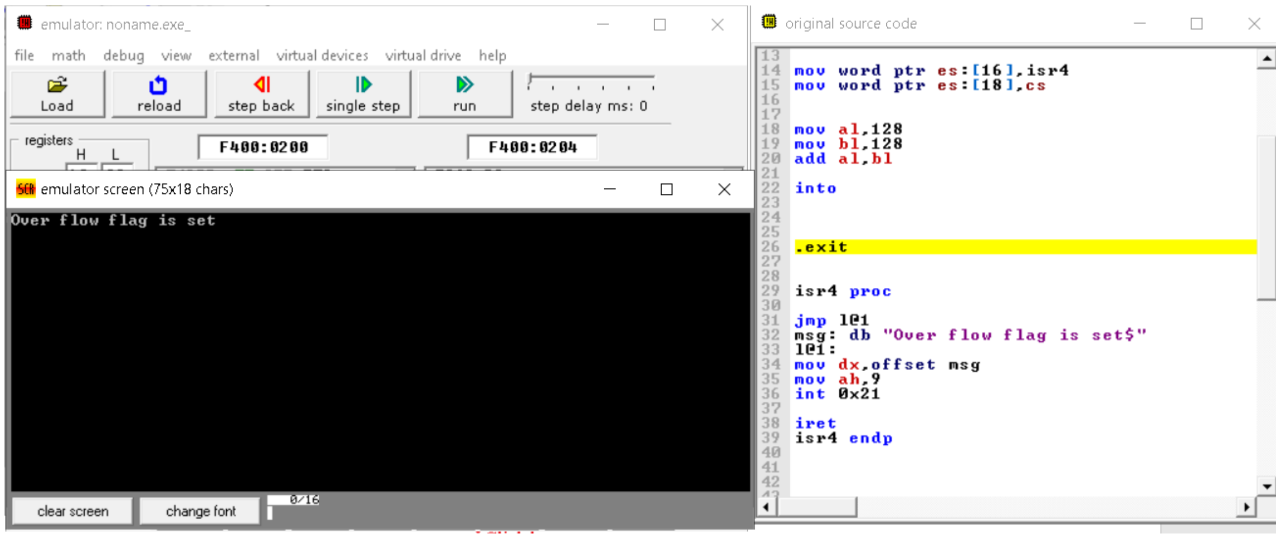
[ ] use Flat Assembler / Intel syntax [see: fasm_compatibility.asm in examples]

Step-4:
Type the code given in example#2 and click on emulate.


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
line: 1 col: 1
drag a file here to open

```
.model small

.data


.code

mov ax,@data
mov ds,ax

mov ax,0
mov es,ax

mov word ptr es:[16],isr4
mov word ptr es:[18],cs


mov al,128
mov bl,128
add al,bl

into



.exit


isr4 proc

jmp l01
msg: db "Over flow flag is set$"
l01:
```

Step-5:
Keep clicking on "Single step" till the last instruction and observe the behavior of the program.



```
Over flow flag is set
```

```
mov word ptr es:[16],isr4
mov word ptr es:[18],cs


mov al,128
mov bl,128
add al,bl

into



.exit


isr4 proc

jmp l@1
msg: db "Over flow flag is set$"
l@1:
mov dx,offset msg
mov ah,9
int 0x21

iret
isr4 endp
```

Practice Exercise

Task

Write a program that hooks **0x65** interrupt with the following services. **The ISR should preserve all the registers except AX**.

Service#	**Description**
0x1	Add two words pointed by SI and DI and store the result in the memory pointed by BX. **[BX] 🡐 [SI] + [DI]**
0x2	Multiply two words pointed by SI and DI and store result in the memory pointed by BX. **[BX] 🡐 [SI] x [DI]**
0x3	Divide the word pointed by SI by the byte pointed by DI and store quotient and remainder in the AL and AH registers, respectively. **AL 🡐 [SI]/[DI]** **AH 🡐 [SI] % [DI]**