Lecture 4: alisoomro666@gmail.com

Addressing Modes, Mov Instruction, Service Routine, ASCII Code and Interrupt

Opcode
(Operational Code)

$\underbrace{2 + 2}_{\text{Operands}}$

Addressing Modes

↳ Ways/Models to access data

Registers Addressing: Both operands are registers ← **Opcode** Reg1, Reg2
Add DL, Al

Immediate Addressing: One Operand is constant term ← **Opcode** Reg, Value
Add DI, 2

Memory Addressing: Access static data directly ← **Opcode** Reg, [Address]
Add DI, [address]

Data Transfer Instruction

Mov DL, 2 DL, 'A'

Mov Ah, 2 ← **Service Routine**

1 = Input a character with echo
2 = Output/Print a single character 'a'
8 = Input a character without echo
9 = Print collection of characters 'abcd' ← String

4ch = Exit

Interrupt

Stop the current program and allow microprocessor to access hardware to take input or give output

INT 21H = Interrupt for Text Handling
INT 20H = Interrupt for Video/Graphics Handling

Example 1: Output

Mov ah, 2
INT 21H

Example 2: Input

Mov ah, 1
INT 21H

ASCII Code

(American Standard Code for Information Interchange)

↳ Character Encoding Scheme

↳ By: American Standards Association (ASA)
Published in; 1963

A = 65 B = 66 → Z=90
a = 91 b = 92 → z=122
0 = 48 1 = 49 → 9=57

Next Line Feed = 10
Carriage Return = 13

Lecture 5:
Program Structure, Syntax and Program to print a single character on screen

alisoomro666@gmail.com

;Program to print a single character on screen

dosseg ← DOS Segment ← Manages the arrangement of segments in a program

.model small ← Model Directive ← Specifies the total amount of memory the program would take.

.stack 100h ← Stack Segment Directive ← Specifies the storage for stack

.data ← Data Segment Directive

;variables are defined here

.code ← Code Segment Directive

```
Main proc

Mov dl, 'A'

Mov ah, 2
INT 21h      ;here we write our program, executable instructions


Mov ah, 4CH
INT 21h

Main endp
End Main
```

Mov 'B', 'A' ✗

Mov 2, 3 ✗

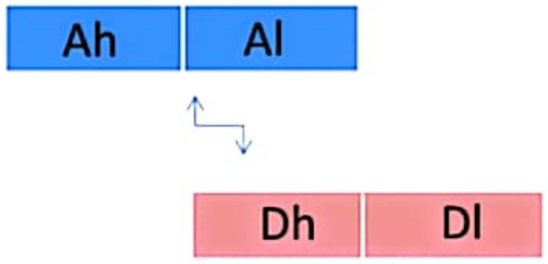
Mov dl, AX ✗

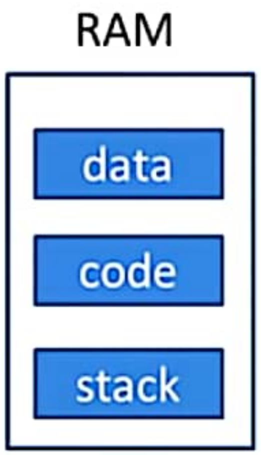
Mov dl, 'A'

Mov dx, AX

Mov dl, 2

Mov dh, al





Tiny	Code + Data <=64KB
Small	Code<=64KB, Data <=64KB
Medium	Code = Any size, Data <=64KB
Compact	Code <=64KB, data = any size
Large	Code = any size, Data = any size
Huge	Code = any size, Data = any size

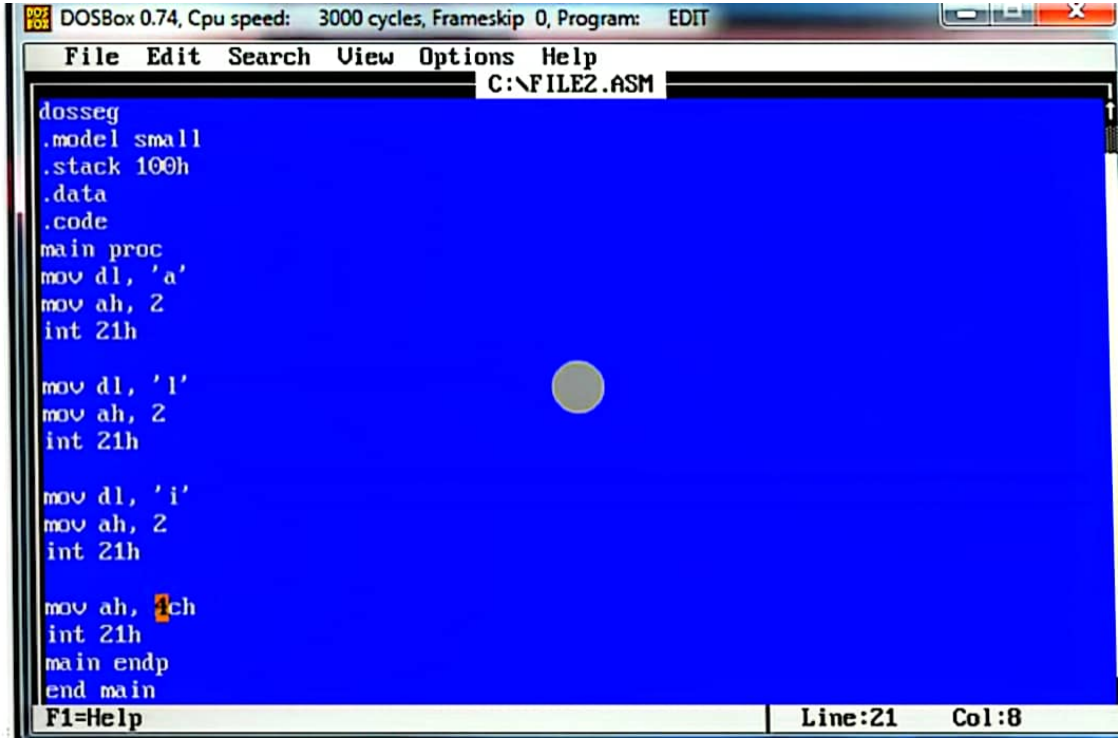
Syntax Rules

→ Space after Opcode
→ One operand must be general purpose register
→ Operands must be of same size
→ Comma , between operands
→ Comment must start with a semi colon ;

Lecture 7 alisoomro666@gmail.com

Program to print a name with characters

a l i

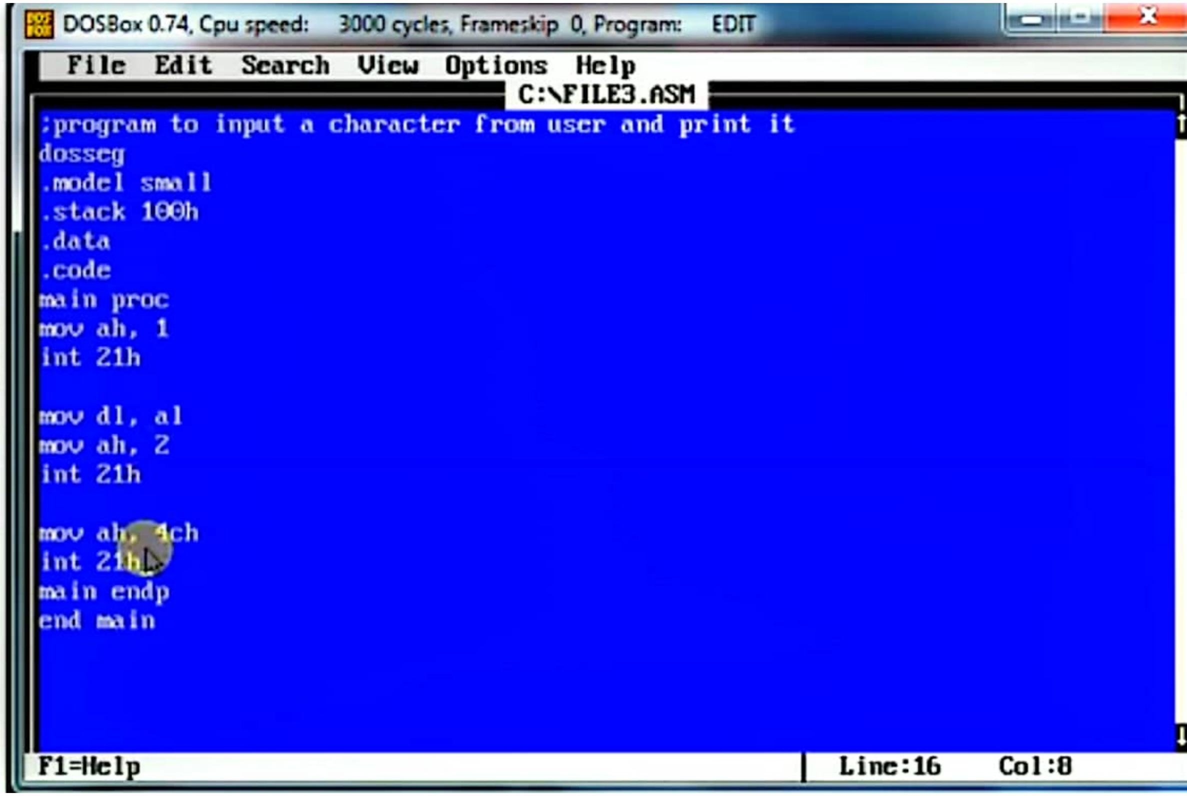

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE2.ASM

```
dosseg
.model small
.stack 100h
.data
.code
main proc
mov dl, 'a'
mov ah, 2
int 21h

mov dl, 'l'
mov ah, 2
int 21h

mov dl, 'i'
mov ah, 2
int 21h

mov ah, 4ch
int 21h
main endp
end main
```

Lecture 8

program to input a character from user and print it

```
mov ah, 1
int 21h

mov dl, al
mov ah, 2
int 21h
```


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File
Edit
Search
View
Options
Help
C:\FILE3.ASM
F1=Help
Line:16
Col:8

```
;program to input a character from user and print it
dosseg
.model small
.stack 100h
.data
.code
main proc
mov ah, 1
int 21h

mov dl, al
mov ah, 2
int 21h

mov ah, 4ch
int 21h
main endp
end main
```

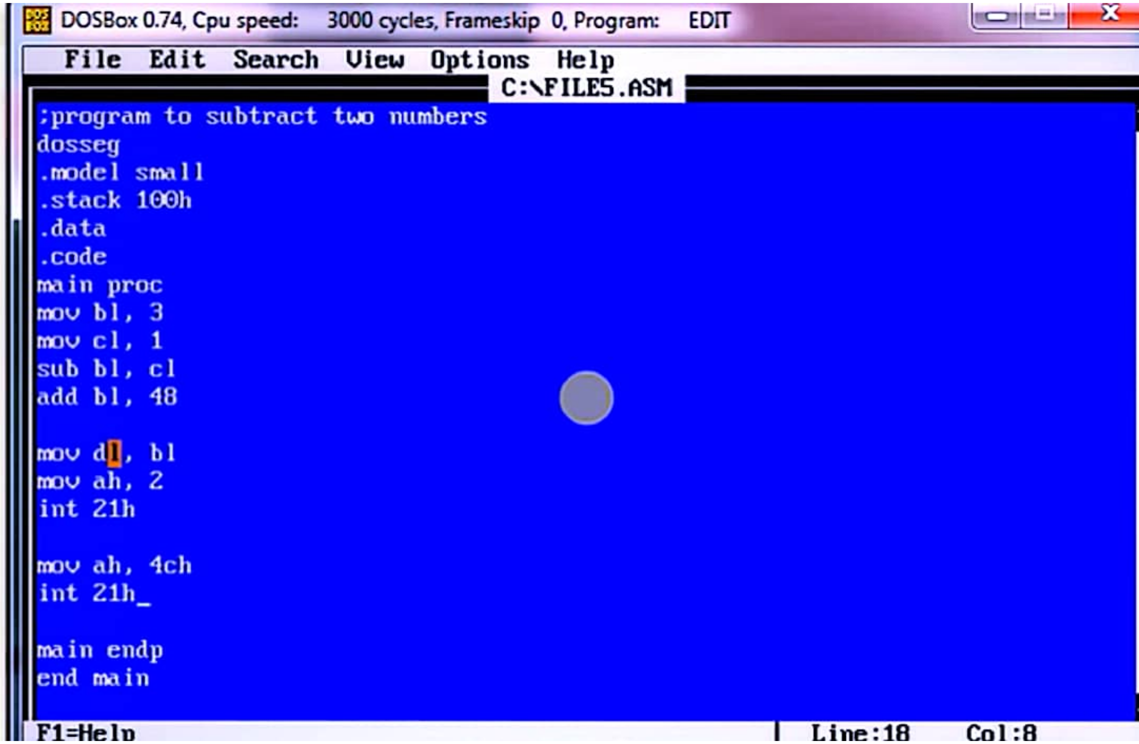
ecture 10

alisoomro666@gmail.com

Program to subtract two numbers

3 - 1 = 2

```
mov bl, 3
mov cl, 1
sub bl, cl

add bl, 48

mov dl, bl
```


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE5.ASM
F1=Help
Line:18
Col:8

```
;program to subtract two numbers
dosseg
.model small
.stack 100h
.data
.code
main proc
mov bl, 3
mov cl, 1
sub bl, cl
add bl, 48

mov dl, bl
mov ah, 2
int 21h

mov ah, 4ch
int 21h_

main endp
end main
```

ecture 9 alisoomro666@gmail.com

Program to add two numbers

1 +2 = 3

mov bl, 1

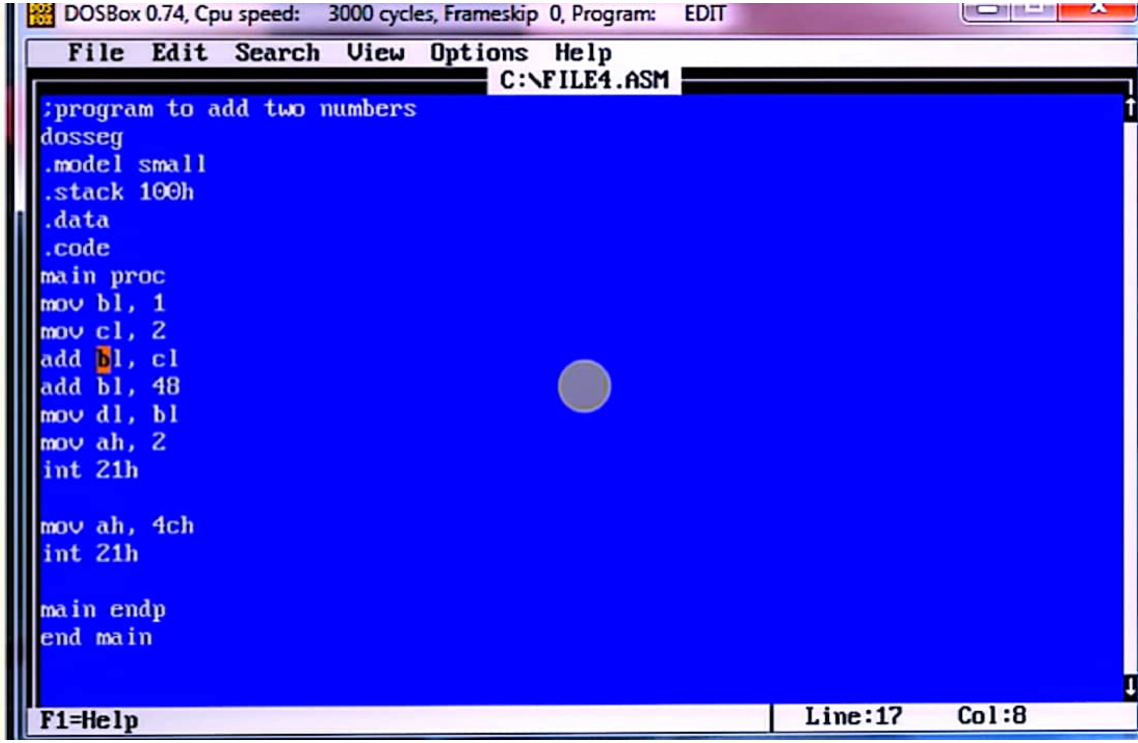
mov cl, 2

ADD bl, cl

add bl, 2

add bl, 48

3 + 48 = 51


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE4.ASM
F1=Help
Line:17 Col:8

```
;program to add two numbers
dosseg
.model small
.stack 100h
.data
.code
main proc
mov bl, 1
mov cl, 2
add bl, cl
add bl, 48
mov dl, bl
mov ah, 2
int 21h

mov ah, 4ch
int 21h

main endp
end main
```

Lecture 11 alisoomro666@gmail.com

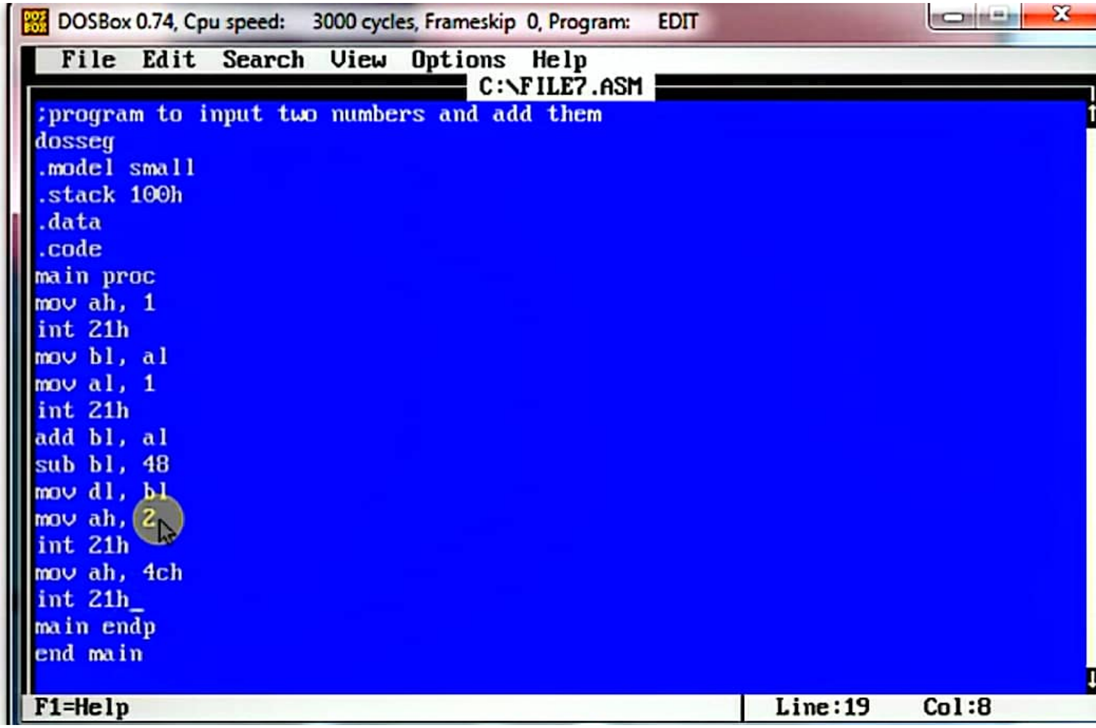
Program to input two numbers and add them

```
mov ah, 1
nt 21h
mov bl, al
mov ah, 1
nt 21h
add bl, al
sub bl, 48
```

1 = 49

2= 50

3 = 99 - 48 = 51


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE7.ASM

```
;program to input two numbers and add them
dosseg
.model small
.stack 100h
.data
.code
main proc
mov ah, 1
int 21h
mov bl, al
mov al, 1
int 21h
add bl, al
sub bl, 48
mov dl, bl
mov ah, 2
int 21h
mov ah, 4ch
int 21h
main endp
end main
```

Lecture 12 alisoomro666@gmail.com

Program to convert capital letter to small letter

mov ah, 1

int 21

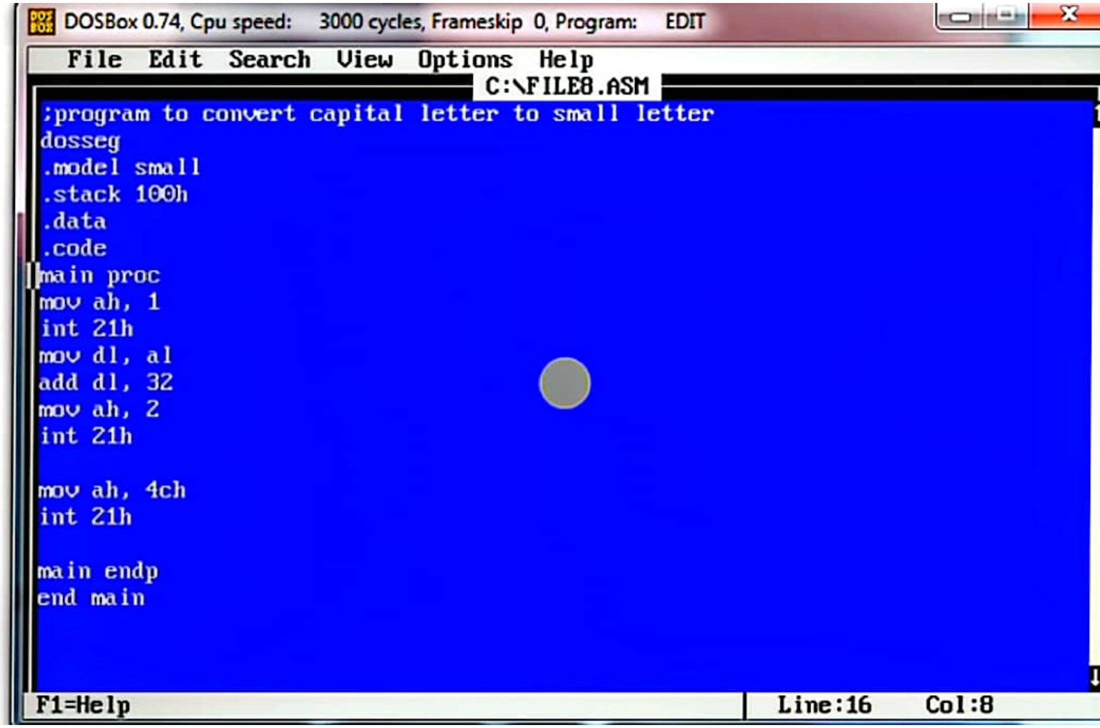
A = 65

B = 66

a = 97

b= 98

97-65 = 32


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE8.ASM
F1=Help
Line:16 Col:8

```
;program to convert capital letter to small letter
dosseg
.model small
.stack 100h
.data
.code
main proc
mov ah, 1
int 21h
mov dl, al
add dl, 32
mov ah, 2
int 21h

mov ah, 4ch
int 21h

main endp
end main
```

Lecture 13:

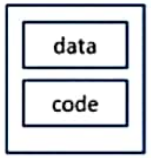
alisoomro666@gmail.com

Variables, Data Types, Offset and LEA

Where to initialize variables in program?

Variables are defined in .data directive of program structure

```
dosseg
.model small
.stack 100h
.data

.code
main proc

main endp
end main
```



How to initialize variables?

Initializer Directive

Initializer

VariableName	DataSize	Value
Var1	db	49
Var1	db	?
Var1	db	'1'
Var1	db	'A'
Var1	db	'1235$'
Var1	db	'hello world$'

AL, BL, CL, DL...
Sub, Add, DIV, MUL
Mov
POP, PUSH

Don't use reserved keywords as VariableName

$ must be used in end of string

$: Terminator, end point of string

Initializer Directive

DB	Define Byte	1 byte, 8 bits
DW	Define Word	2 bytes, 16 bits
DD	Define Double word	4 bytes, 32 bits
DQ	Define QuadWord	8 bytes, 64 bits
DT	Define TenBytes	10 bytes, 80 bits

It moves the memory location of @DATA into the AX register (16 bit register).

Moves data address to ds so that data segment gets initialized as heap memory to access variables fast

Offset

Holds the beginning address of Variable as 16 bits

Type Mismatch

16 bits Db : 8 bits
Mov dx, var1

LEA

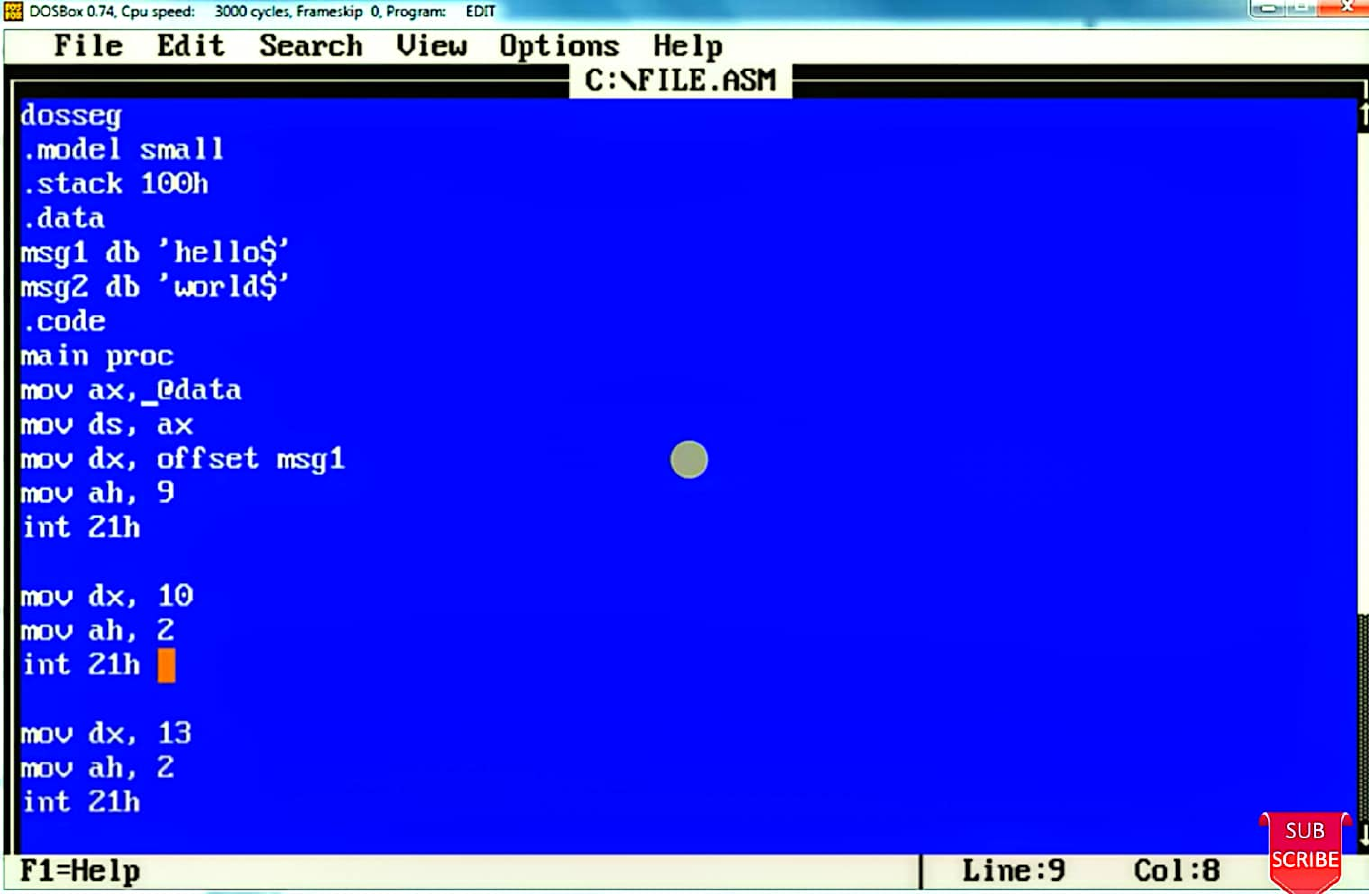
Load Effective Address

It is an indirect instruction used as a pointer in which first variable Points the address of second variable

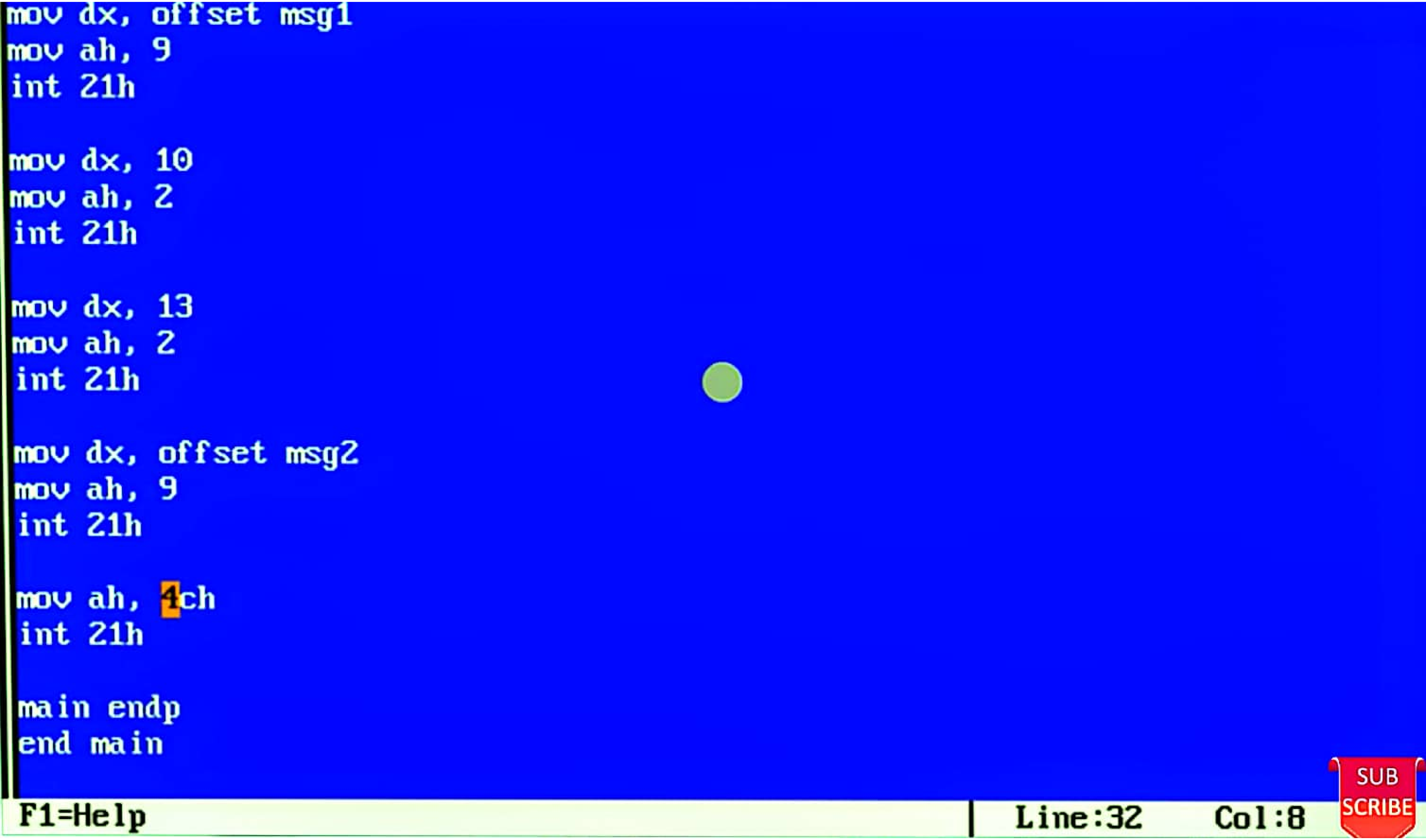
```
Dosseg
.model small
.stack 100h
.data
Var1 db '1'
Var2 db ?
Var3 db '1235$'
.code
Main proc
Mov ax, @data
Mov ds, ax
Mov dl, Var1
Mov ah, 2
Int 21h
Mov Var2, bl
Mov dl, Var3
Mov dx, offset Var3
lea dx, Var3
Mov ah, 9
Int 21h
Main endp
End main
```

Not for string

First Character From string


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE.ASM
F1=Help
Line:9 Col:8
SUB SCRIBE

```
dosseg
.model small
.stack 100h
.data
msg1 db 'hello$'
msg2 db 'world$'
.code
main proc
mov ax,_@data
mov ds, ax
mov dx, offset msg1
mov ah, 9
int 21h

mov dx, 10
mov ah, 2
int 21h

mov dx, 13
mov ah, 2
int 21h
```



```
mov dx, offset msg1
mov ah, 9
int 21h

mov dx, 10
mov ah, 2
int 21h

mov dx, 13
mov ah, 2
int 21h

mov dx, offset msg2
mov ah, 9
int 21h

mov ah, 4ch
int 21h

main endp
end main
```

Lecture 15:

alisoomro666@gmail.com

Loop, Label, Counter Register, Inc and Program to print 0 to 9

```
dosseg
.model small
.stack 100h
.data
.code
main proc

Mov cx,10
mov dx, 48

L1:
~~Mov dx, 48~~

Mov ah, 2
Int 21h

Add dx, 1

Loop L1

Mov ah,4ch
Int 21h

main endp
end main
```

Series of instructions that is repeated until a terminating condition is reached.

```
Mov dx, 'a'
Mov ah, 2
Int 21h
```

Increment by 1

Inc dx

~~Mov:~~

Label Syntax

~~1Test:~~

Test:
Test1:
T1:

LabelName:

```
Mov dx, 'a'
Mov ah, 2
Int 21h
```

Loop **LabelName**

General purpose registers,
Main purpose is to be used for a loop

Counter Register

Mov CX, 10

Works on
Decrement
By 1

Cx = 10
Cx = 9
Cx = 8

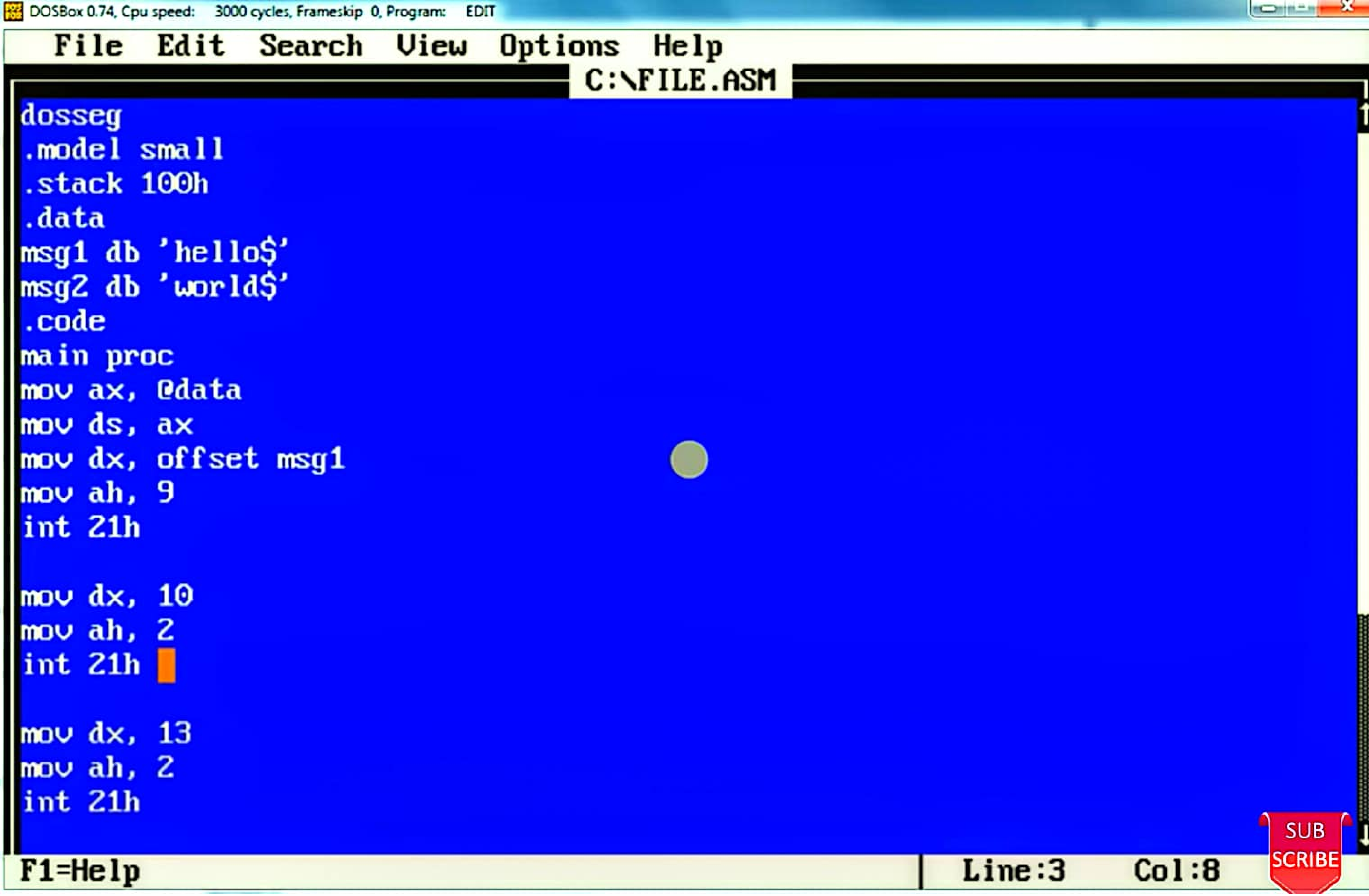
Cx = 0

Label Rules:

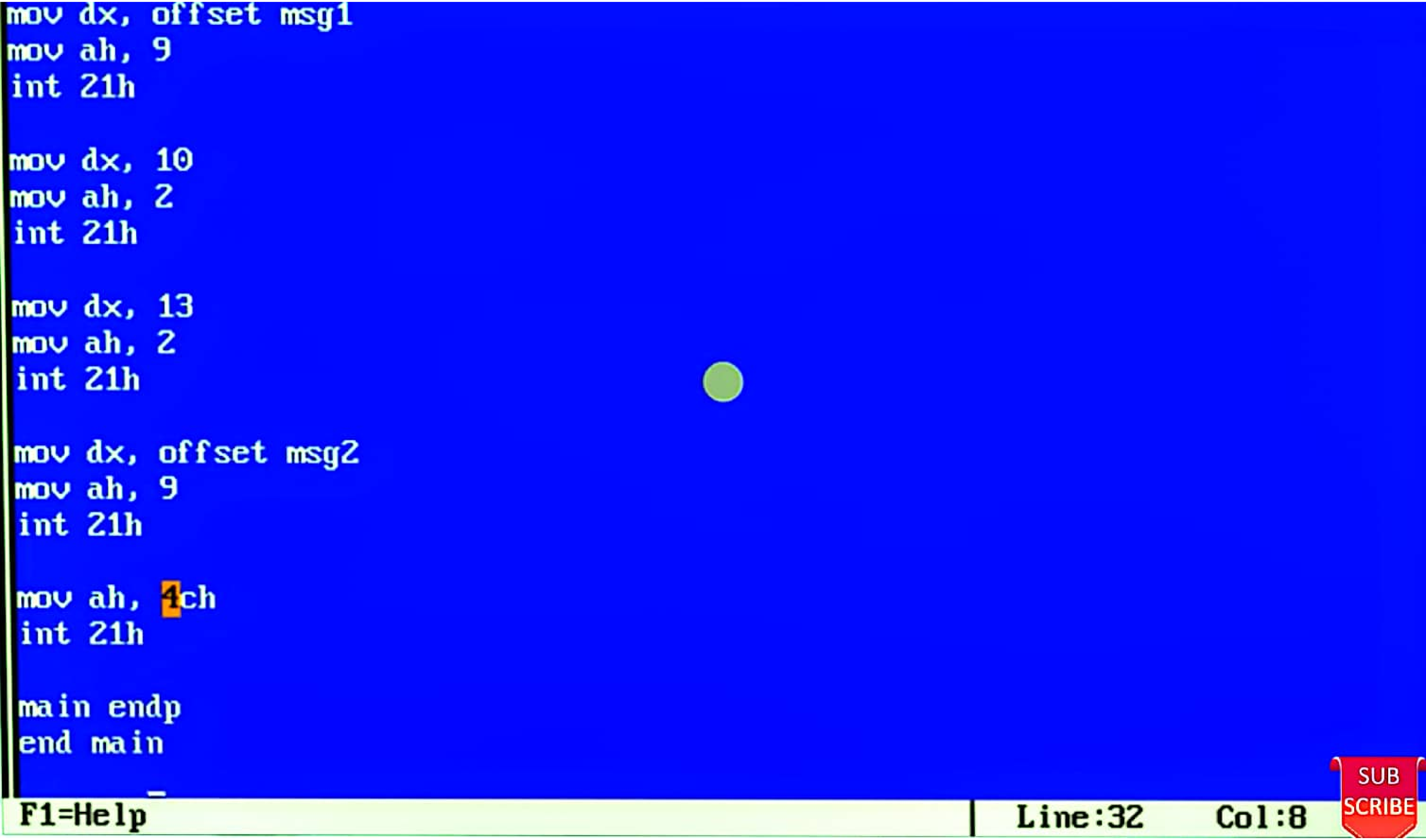
1. A **label** can be placed at the beginning of a statement, because the label is assigned the current value of line

2. Label name must not be a reserved word e.g. Mov, Add, DB and DW

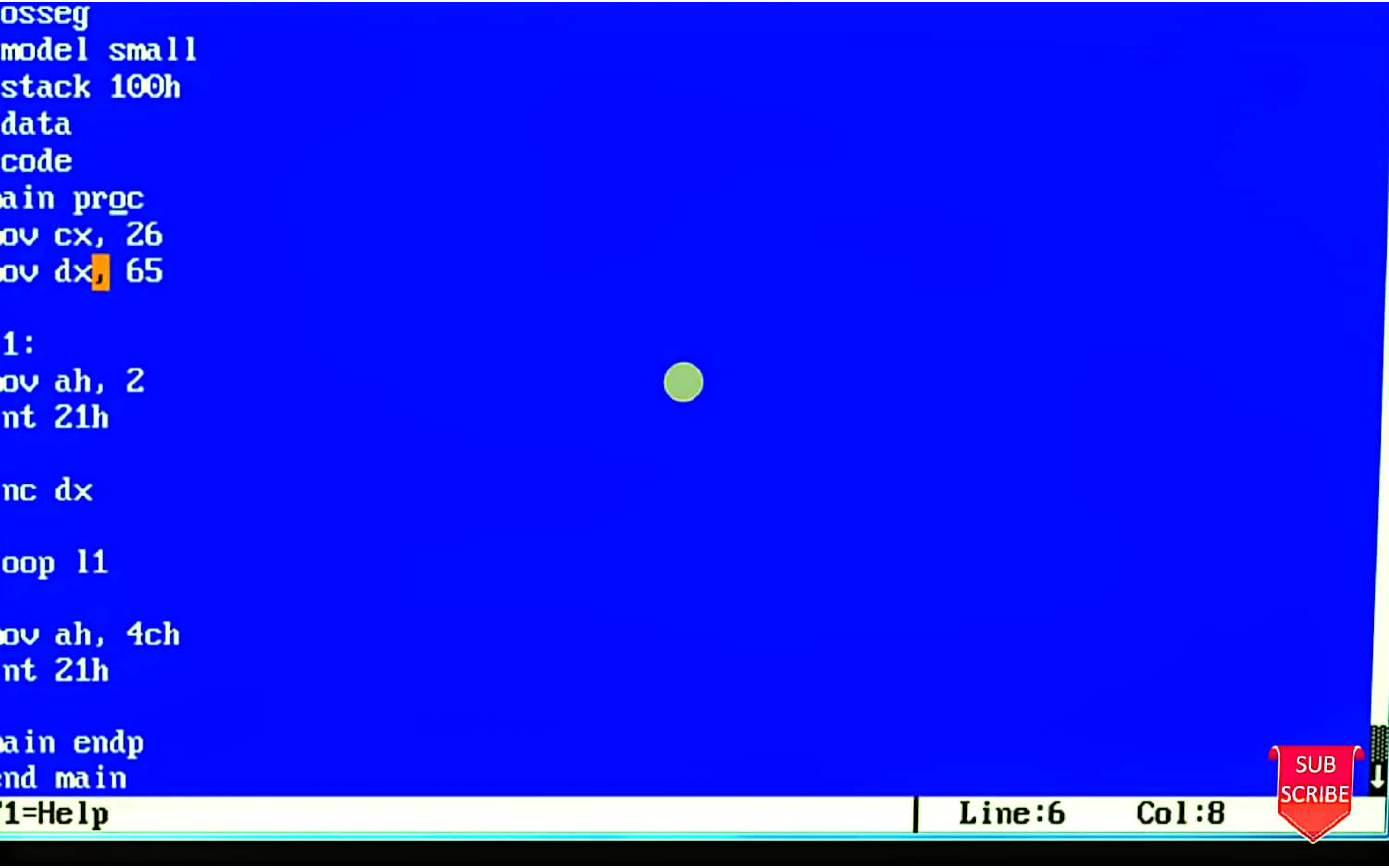
3. Colon : must be used with Label While initializing, but not while calling


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EDIT
File Edit Search View Options Help
C:\FILE.ASM
F1=Help
Line:3 Col:8
SUB SCRIBE

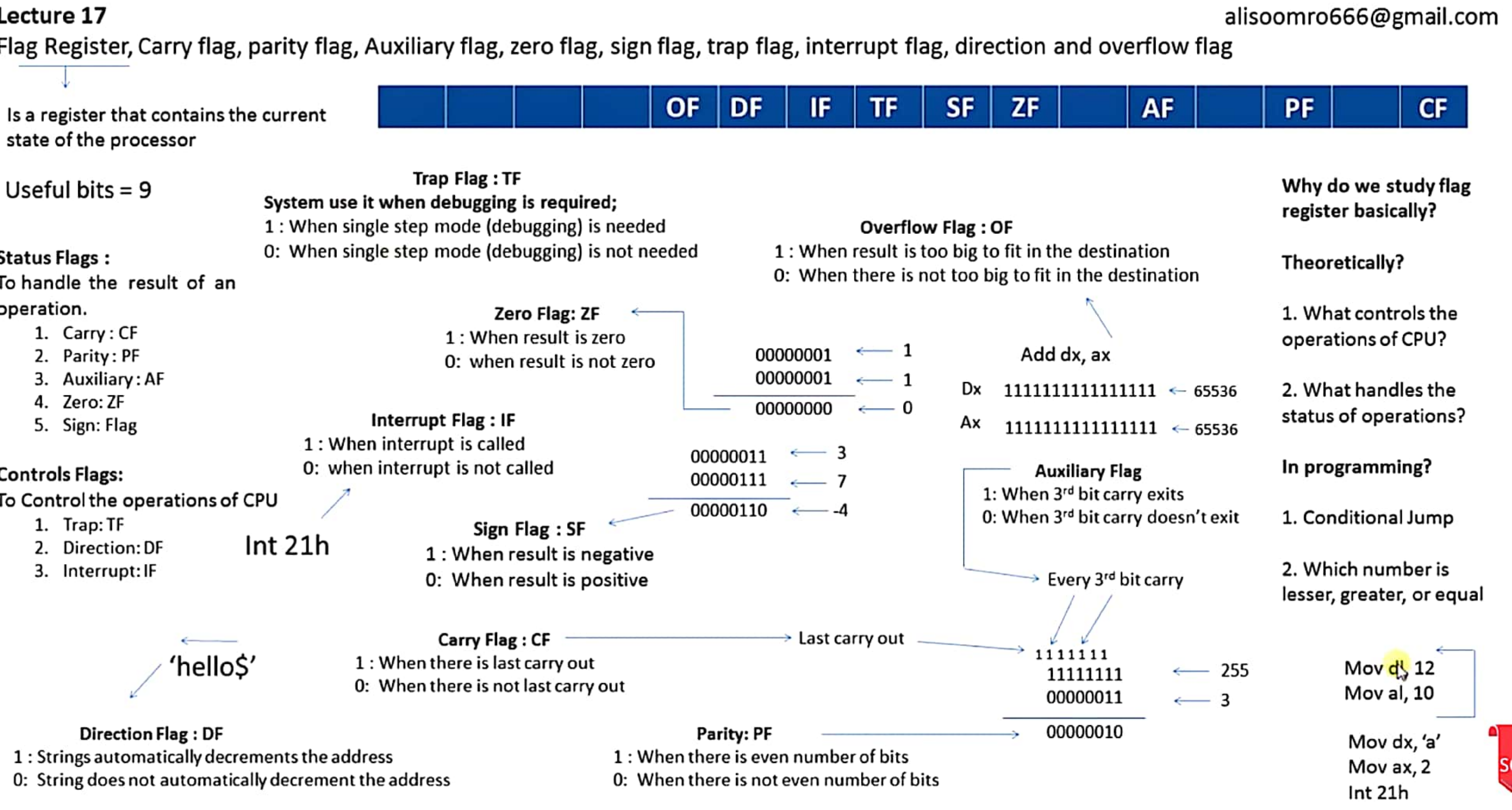
```
dosseg
.model small
.stack 100h
.data
msg1 db 'hello$'
msg2 db 'world$'
.code
main proc
mov ax, @data
mov ds, ax
mov dx, offset msg1
mov ah, 9
int 21h

mov dx, 10
mov ah, 2
int 21h

mov dx, 13
mov ah, 2
int 21h
```



```
mov dx, offset msg1
mov ah, 9
int 21h

mov dx, 10
mov ah, 2
int 21h

mov dx, 13
mov ah, 2
int 21h

mov dx, offset msg2
mov ah, 9
int 21h

mov ah, 4ch
int 21h

main endp
end main
```



```
osseg
model small
stack 100h
data
code
ain proc
ov cx, 26
ov dx, 65

1:
ov ah, 2
nt 21h

nc dx

oop l1

ov ah, 4ch
nt 21h

ain endp
nd main
```



Lecture 17

alisoomro666@gmail.com

Flag Register, Carry flag, parity flag, Auxiliary flag, zero flag, sign flag, trap flag, interrupt flag, direction and overflow flag

Is a register that contains the current state of the processor

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Useful bits = 9

Status Flags :
To handle the result of an operation.

1. Carry : CF
2. Parity : PF
3. Auxiliary : AF
4. Zero: ZF
5. Sign: Flag

Controls Flags:
To Control the operations of CPU

1. Trap: TF
2. Direction: DF
3. Interrupt: IF

Trap Flag : TF
System use it when debugging is required;
1 : When single step mode (debugging) is needed
0: When single step mode (debugging) is not needed

Overflow Flag : OF
1 : When result is too big to fit in the destination
0: When there is not too big to fit in the destination

Add dx, ax

Dx 1111111111111111 ← 65536
Ax 1111111111111111 ← 65536

Zero Flag: ZF
1 : When result is zero
0: when result is not zero

```
00000001  ← 1
00000001  ← 1
--------
00000000  ← 0
```

Interrupt Flag : IF
1 : When interrupt is called
0: when interrupt is not called

Int 21h

```
00000011  ← 3
00000111  ← 7
--------
00000110  ← -4
```

Sign Flag : SF
1 : When result is negative
0: When result is positive

Auxiliary Flag
1: When 3rd bit carry exits
0: When 3rd bit carry doesn't exit

Every 3rd bit carry

Carry Flag : CF
1 : When there is last carry out
0: When there is not last carry out

Last carry out

```
 111111
 11111111  ← 255
 00000011  ← 3
---------
 00000010
```

'hello$'

Direction Flag : DF
1 : Strings automatically decrements the address
0: String does not automatically decrement the address

Parity: PF
1 : When there is even number of bits
0: When there is not even number of bits

Why do we study flag register basically?

Theoretically?

1. What controls the operations of CPU?

2. What handles the status of operations?

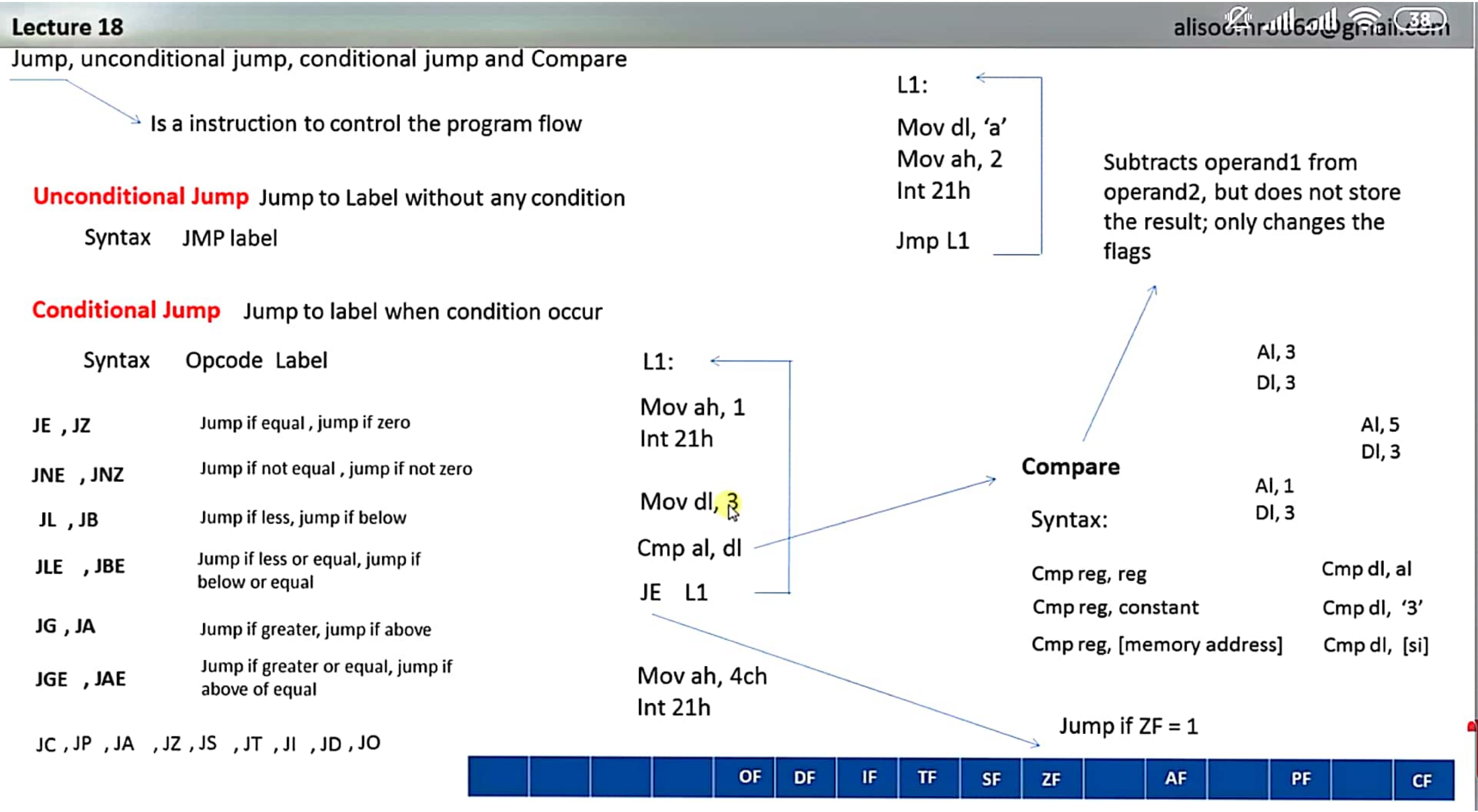
In programming?

1. Conditional Jump

2. Which number is lesser, greater, or equal

Mov dl, 12
Mov al, 10

Mov dx, 'a'
Mov ax, 2
Int 21h



Lecture 18 alisoomro060@gmail.com

Jump, unconditional jump, conditional jump and Compare

Is a instruction to control the program flow

Unconditional Jump Jump to Label without any condition

Syntax JMP label

Conditional Jump Jump to label when condition occur

Syntax Opcode Label

JE , JZ	Jump if equal , jump if zero
JNE , JNZ	Jump if not equal , jump if not zero
JL , JB	Jump if less, jump if below
JLE , JBE	Jump if less or equal, jump if below or equal
JG , JA	Jump if greater, jump if above
JGE , JAE	Jump if greater or equal, jump if above of equal

JC , JP , JA , JZ , JS , JT , JI , JD , JO

```
L1:

Mov dl, 'a'
Mov ah, 2
Int 21h

Jmp L1
```

```
L1:

Mov ah, 1
Int 21h

Mov dl, 3

Cmp al, dl

JE    L1

Mov ah, 4ch
Int 21h
```

Subtracts operand1 from operand2, but does not store the result; only changes the flags

Al, 3
Dl, 3

Al, 5
Dl, 3

Al, 1
Dl, 3

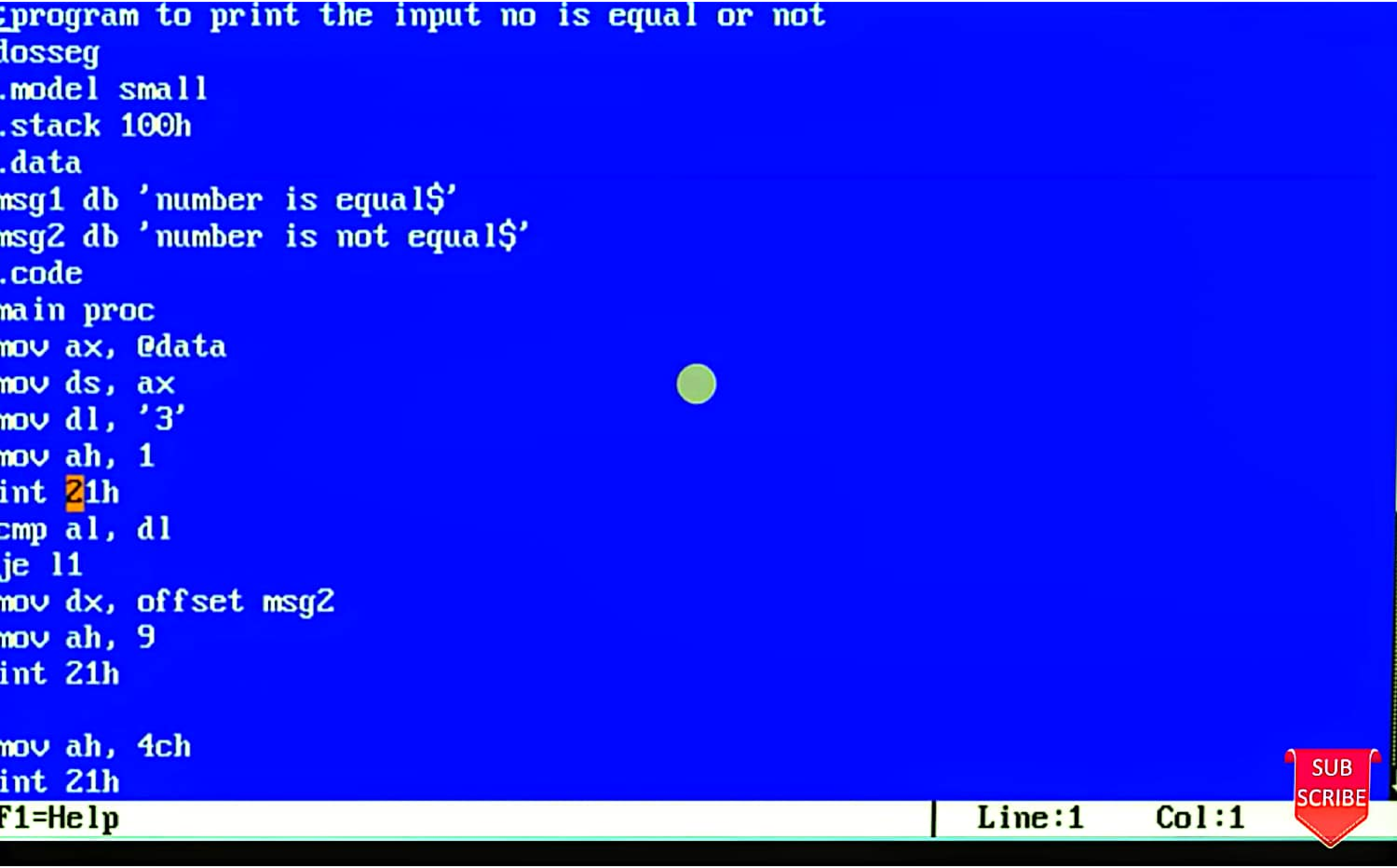
Compare

Syntax:

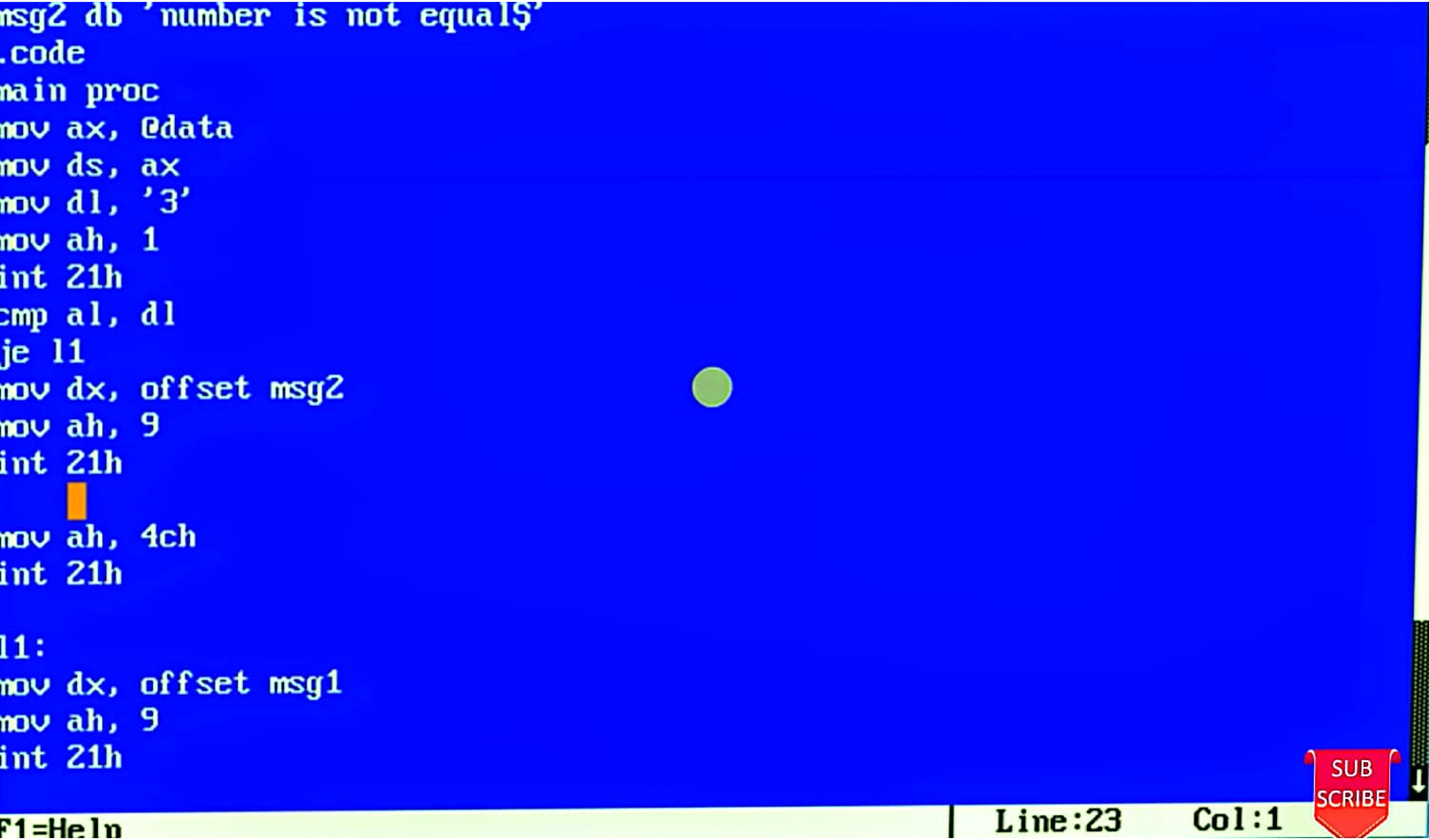
Cmp reg, reg	Cmp dl, al
Cmp reg, constant	Cmp dl, '3'
Cmp reg, [memory address]	Cmp dl, [si]

Jump if ZF = 1

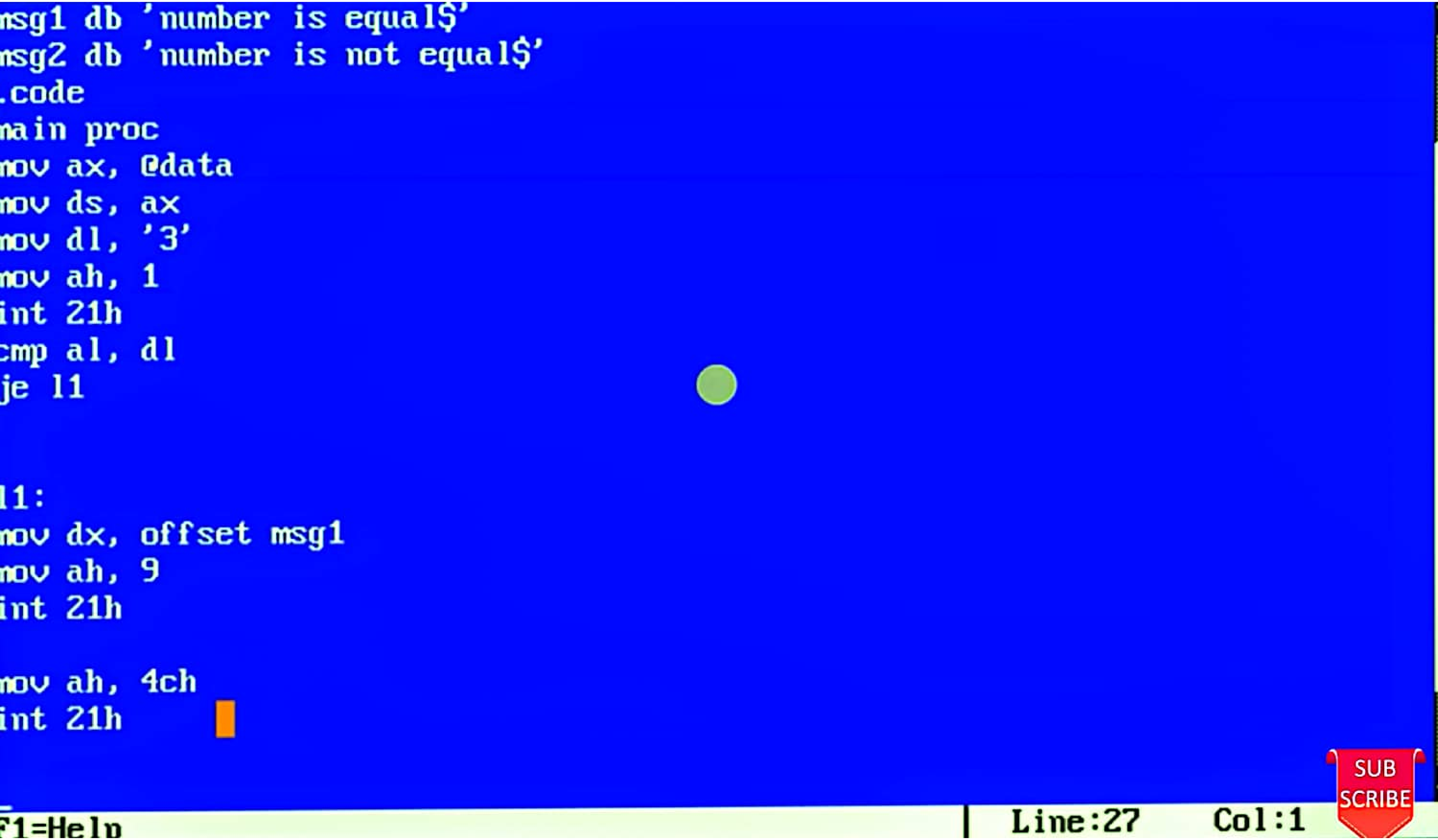
				OF	DF	IF	TF	SF	ZF		AF		PF		CF



```
;program to print the input no is equal or not
dosseg
.model small
.stack 100h
.data
msg1 db 'number is equal$'
msg2 db 'number is not equal$'
.code
main proc
mov ax, @data
mov ds, ax
mov dl, '3'
mov ah, 1
int 21h
cmp al, dl
je l1
mov dx, offset msg2
mov ah, 9
int 21h

mov ah, 4ch
int 21h
```

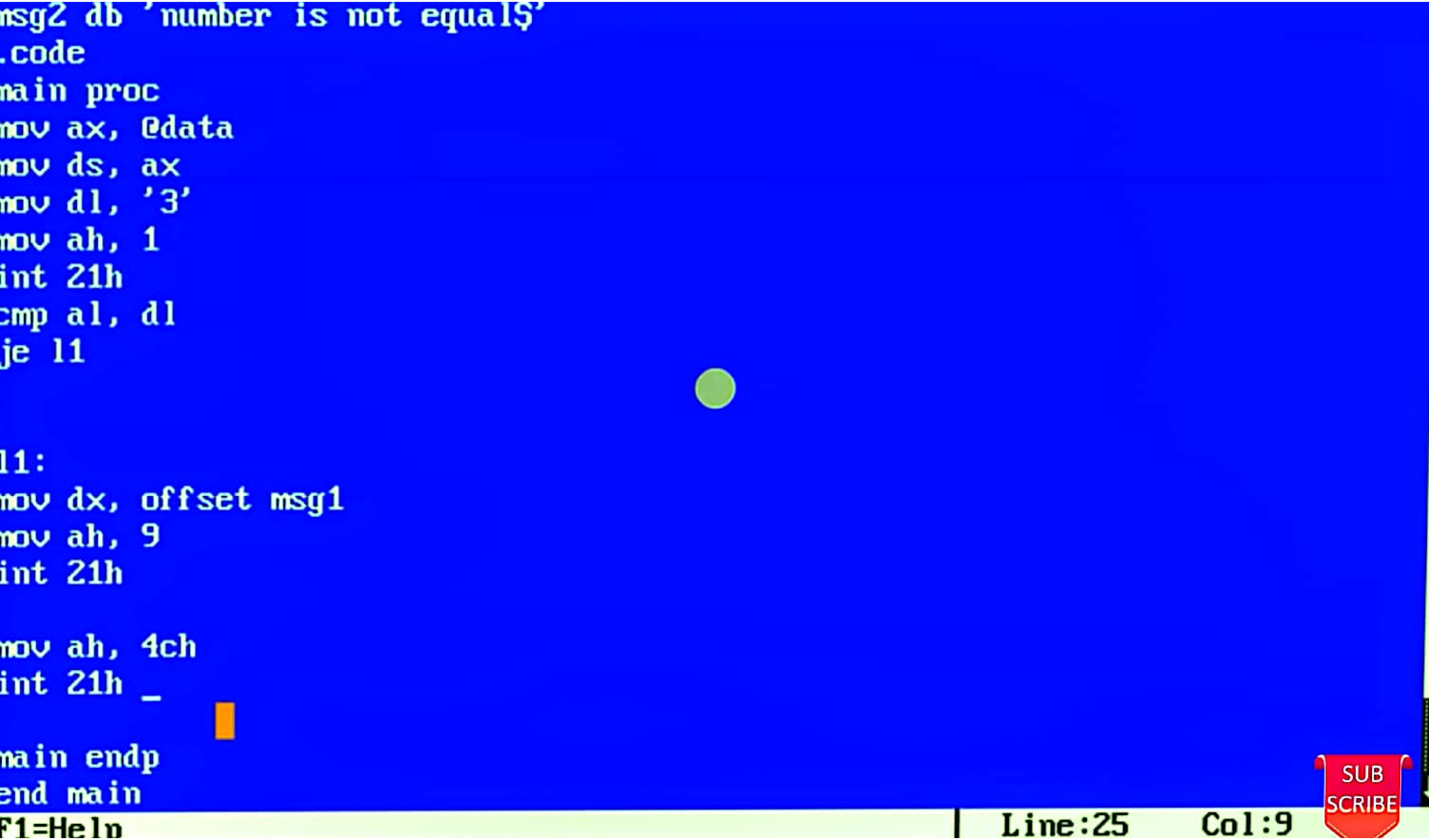


```
msg2 db 'number is not equal$'
.code
main proc
mov ax, @data
mov ds, ax
mov dl, '3'
mov ah, 1
int 21h
cmp al, dl
je l1
mov dx, offset msg2
mov ah, 9
int 21h

mov ah, 4ch
int 21h

l1:
mov dx, offset msg1
mov ah, 9
int 21h
```



```
msg1 db 'number is equal$'
msg2 db 'number is not equal$'
.code
main proc
mov ax, @data
mov ds, ax
mov dl, '3'
mov ah, 1
int 21h
cmp al, dl
je l1


l1:
mov dx, offset msg1
mov ah, 9
int 21h

mov ah, 4ch
int 21h
```

```
F1=Help                                   Line:27     Col:1
```



```
msg2 db 'number is not equal$'
.code
main proc
mov ax, @data
mov ds, ax
mov dl, '3'
mov ah, 1
int 21h
cmp al, dl
je l1


l1:
mov dx, offset msg1
mov ah, 9
int 21h

mov ah, 4ch
int 21h _

main endp
end main
```

```
F1=Help                                                  | Line:25     Col:9
```

Lecture 20

Array, dup and Source Index

alisoomro666@gmail.com

Collection of characters in sequence

Where to initialize?

Array is defined in .data directive of program as variable

How to initialize?

In same way as variable but with multiple values

Arr1	db	1,2,3,4	
Arr1	db	'a', 'b', 'c'	
Arr1	db	'abc'	
Arr1	db	'a', 'a', 'a'	
Arr1	db	?, ?, ?, ?	
Arr1	db	3	Dup('a')
Arr1	db	4	Dup(?)

Number of characters

Duplicates the value

SI

Source index register, used as pointer to access array

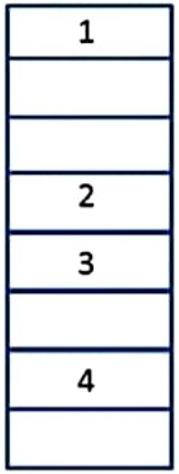
1, 2, 3, 4

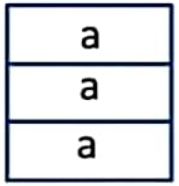
Var1 db 1

Var2 db 2

Var3 db 3

Var4 db 4





```
.model small
.stack 100h
.data

arr1 db 1, 2, 3, 4

.code
Main proc
Mov ax, @data
Mov ds, ax

Mov si, offset arr1

~~Mov dx, si~~

Mov dx, [si]
Mov ah, 2
Int 21h

; mov dx, [si + 1]
Inc si
Mov dx, [si]
Mov ah, 2
Int 21h

Main endp
End main
```

Why do we need to learn Array?

To store many characters with single variable name in sequence in memory

Var1 db 1, 2, 3, 4

How to access array?

Starting address

/

Address of first character

Bracket form to access value at address

Ah000	1
Ah001	2
Ah002	3
Ah003	4