Finally we are ready to display the character:

```
MOV DL,BL          ;get character
INT 21h            ;and display it
```

Here is the complete program:

Program Listing PGM4_1.ASM

```
TITLE PGM4_1: ECHO PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
;display prompt
      MOV AH,2          ;display character function
```

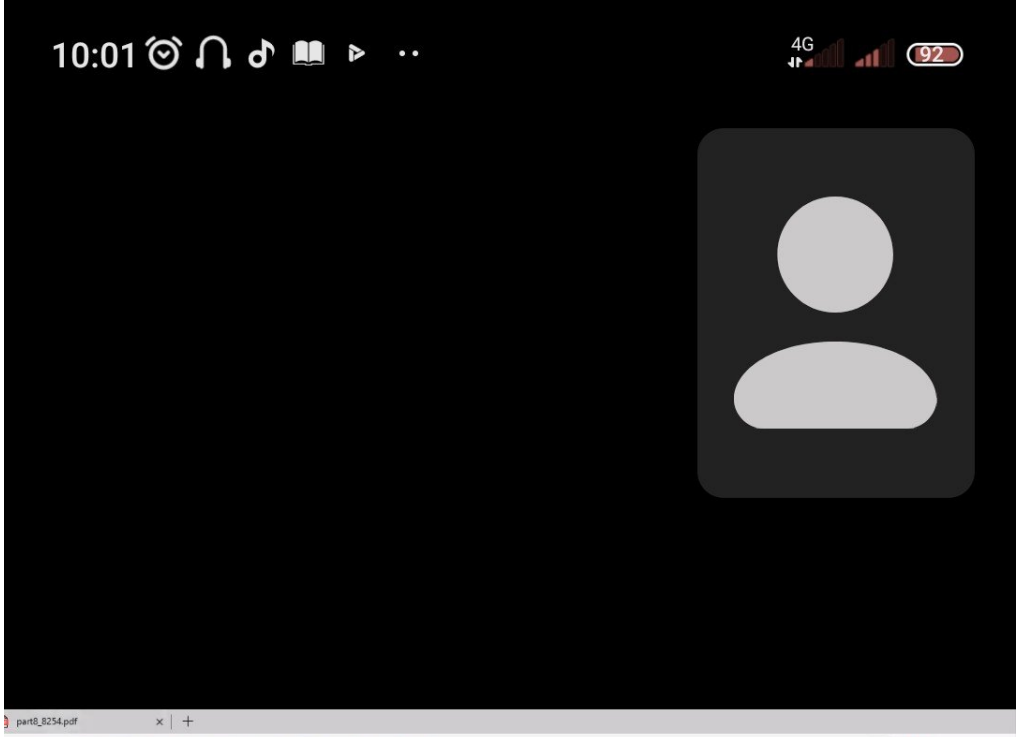


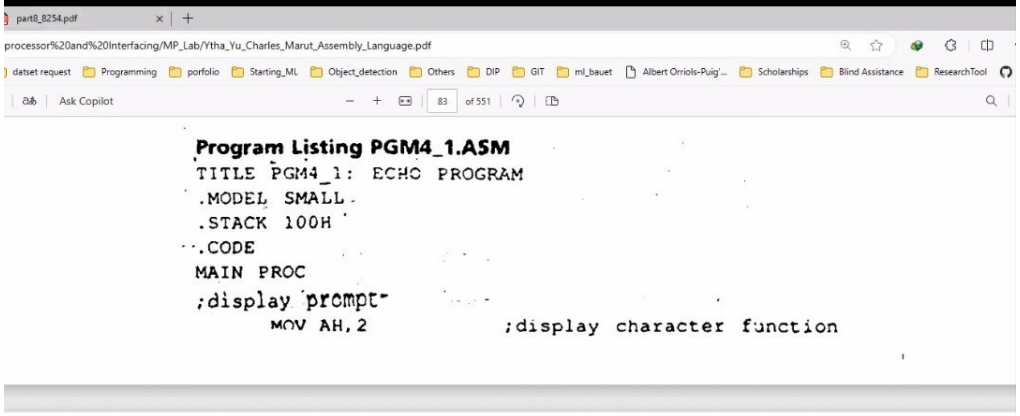
```
      MOV DL,'?'        ;character is '?'
      INT 21H           ;display it
;input a character
      MOV AH,1          ;read character function
      INT 21H           ;character in AL
      MOV BL,AL         ;save it in BL
;go to a new line
      MOV AH,2          ;display character function
      MOV DL,0DH        ;carriage return
      INT 21H           ;execute carriage return
      MOV DL,0AH        ;line feed
      INT 21H           ;execute line feed
;display character
      MOV DL,BL         ;retrieve character
      INT 21H           ;and display it
;return to DOS
      MOV AH,4CH        ;DOS exit function
      INT 21H           ;exit to DOS
MAIN  ENDP
      END  MAIN
```

Because no variables were used, the data segment was omitted.

Terminating a Program

The last two lines in the MAIN procedure require some explanation. When a program terminates, it should return control to DOS. This can be accomplished by executing INT 21h, function 4Ch.





part8_8254.pdf

processor%20and%20Interfacing/MP_Lab/Ytha_Yu_Charles_Marut_Assembly_Language.pdf

datset request | Programming | porfolio | Starting_ML | Object_detection | Others | DIP | GIT | ml_bauet | Albert Orriols-Puig'... | Scholarships | Blind Assistance | ResearchTool

Ask Copilot | 83 of 551

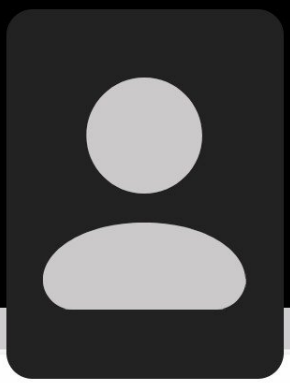
Program Listing PGM4_1.ASM

```
TITLE PGM4_1: ECHO PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
;display prompt
      MOV AH,2          ;display character function
```



```
      MOV                ;character is '?'
      INT                ;display it
;input a
      MOV                ;read character function
      INT                ;character in AL
      MOV                ;save it in BL
;go to a new line
```

Copy
Search
Highlight
Add comment
Ask Copilot
More actions



Program Listing PGM4_2.ASM

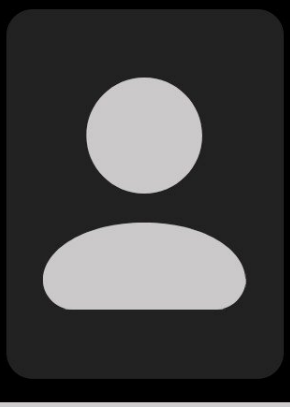
```
TITLE PGM4_2: PRINT STRING PROGRAM
.MODEL  SMALL
.STACK  100H
.DATA
MSG     DB      'HELLO!$'
.CODE
MAIN    PROC
;initialize DS
        MOV AX,@DATA
        MOV DS,AX               ;initialize DS
;display message
        LEA DX,MSG              ;get message
        MOV AH,9                ;display string function
        INT 21h                 ;display message
;return to DOS
        MOV AH,4CH
```



```
        INT 21h                 ;DOS exit
MAIN    ENDP
        END  MAIN
```

And here is a sample execution:

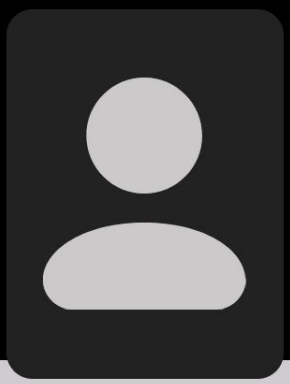
```
A> PGM4_2
```




```
if(a>b)
true statement
else
statement
```

Control
uctions

For assembly language programs to carry out useful tasks, there must be a way to make decisions and repeat sections of code. In this chapter we show how these things can be accomplished with the jump and loop instructions.

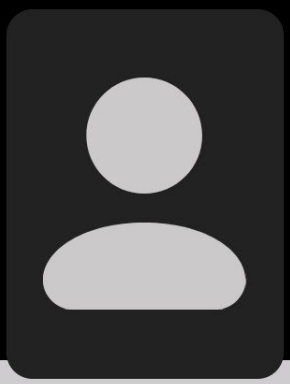


if(a>b)
true statement
else
statement

CMP
jump
Cj
uj

Control
uctions

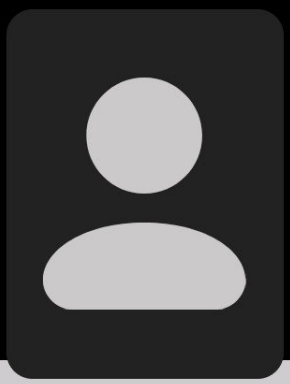
For assembly language programs to carry out useful tasks, there
be a way to make decisions and repeat sections of code. In this chapte
show how these things can be accomplished with the jump and loo
structions.



Control
ctions

```
if(a>b)
true statement
else
statement

CMP
jump
Cj
uj

DES

STATEMENT SYNTAX
LEVEL: OPERATION OPERAND
:COMMENT
```

For assembly language programs to carry out useful tasks, there mu
be a way to make decisions and repeat sections of code. In this chapter w
show how these things can be accomplished with the jump and loop in
structions.



if(a>b)
true statement
else
statement

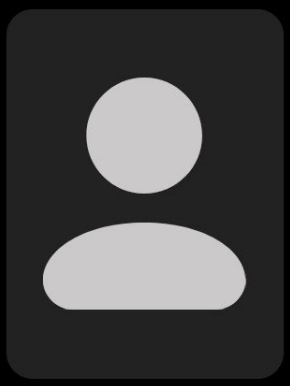
CMP
jump
Cj
uj

DES

STATEMENT SYNTAX
LEVEL: OPERATION OPERAND
;COMMENT
MOV AL,BL

Control Instructions

For assembly language programs to carry out useful tasks, there must be a way to make decisions and repeat sections of code. In this chapter we show how these things can be accomplished with the jump and loop instructions.



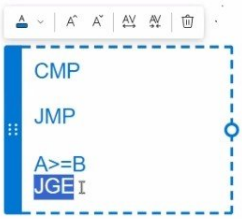


Table 6.1 Conditional Jumps

Signed Jumps

Symbol	*Description*	*Condition for Jumps*
JG/JNLE	jump if greater than jump if not less than or equal to	ZF = 0 and SF = OF
JGE/JNL	jump if greater than or equal to jump if not less than or equal to	SF = OF
JL/JNGE	jump if less than jump if not greater than or equal	SF <> OF
JLE/JNG	jump if less than or equal jump if not greater than	ZF = 1 or SF <> OF

Unsigned Conditional Jumps

Symbol	*Description*	*Condition for Jumps*
JA/JNBE	jump if above jump if not below or equal	CF = 0 and ZF = 0
JAE/JNB	jump if above or equal jump if not below	CF = 0
JB/JNAE	jump if below jump if not above or equal	CF = 1
JBE/JNA	jump if equal jump if not above	CF = 1 or ZF = 1

and (3) the **single-flag jumps**, which operate on settings of individual flags. *Note:* the jump instructions themselves do not affect the flags.

The first column of Table 6.1 gives the opcodes for the jumps. Many of the jumps have two opcodes; for example, JG and JNLE. Both opcodes produce the same machine code. Use of one opcode or its alternate is usually determined by the context in which the jump appears.

The CMP Instruction

The jump condition is often provided by the **CMP** *(compare)* instruction. It has the form

```
CMP  destination, source
```

This instruction compares destination and source by computing destination contents minus source contents. The result is not stored, but the flags are affected. The operands of CMP may not both be memory locations. Destination may not be a constant. *Note:* CMP is just like SUB, except that destination is not changed.

For example, suppose a program contains these lines:

```
CMP  AX,BX
JG   BELOW
```

where AX = 7FFFh, and BX = 0001. The result of CMP AX,BX is 7FFFh − 0001h = 7FFEh. Table 6.1 shows that the jump condition for JG is satisfied, because ZF = SF = OF = 0, so control transfers to label BELOW.

The CMP Instruction

The jump condition is often provided by the **CMP** *(compare)* instruction. It has the form

```
CMP  destination, source
```

This instruction compares destination and source by computing destination contents minus source contents. The result is not stored, but the flags are affected. The operands of CMP may not both be memory locations. Destination may not be a constant. *Note:* CMP is just like SUB, except that destination is not changed.

For example, suppose a program contains these lines:

```
CMP   AX, BX
JG    BELOW
```

where AX = 7FFFh, and BX = 0001. The result of CMP AX,BX is 7FFFh − 0001h = 7FFEh. Table 6.1 shows that the jump condition for JG is satisfied, because ZF = SF = OF = 0, so control transfers to label BELOW.

The CMP Instruction

The jump condition is often provided by the **CMP** *(compare)* instruction. It has the form

```
CMP  destination, source
```

This instruction compares destination and source by computing destination contents minus source contents. The result is not stored, but the flags are affected. The operands of CMP may not both be memory locations. Destination may not be a constant. *Note:* CMP is just like SUB, except that destination is not changed.

For example, suppose a program contains these lines:

```
CMP   AX, BX
JG    BELOW
```

where AX = 7FFFh, and BX = 0001. The result of CMP AX,BX is 7FFFh – 0001h = 7FFEh. Table 6.1 shows that the jump condition for JG is satisfied, because ZF = SF = OF = 0, so control transfers to label BELOW.

MOV AL,BL

CMP AL,BL

Example 6.2 Replace the number in AX by its absolute value.

Solution: A pseudocode algorithm is

```
IF AX < 0
 THEN
  replace AX    -AX
END_IF
```

It can be coded as follows:

```
;if AX < 0
                CMP   AX,0        ;AX < 0 ?
                JNL   END_IF      ;no, exit
;then
                NEG   AX          ;yes, change sign
END_IF:
```

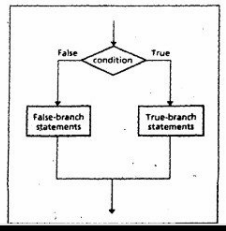
The condition AX < 0 is expressed by CMP AX,0. If AX is not less than 0, there is nothing to do, so we use a JNL (jump if not less) to jump around the NEG AX. If condition AX < 0 is true, the program goes on to execute NEG AX.

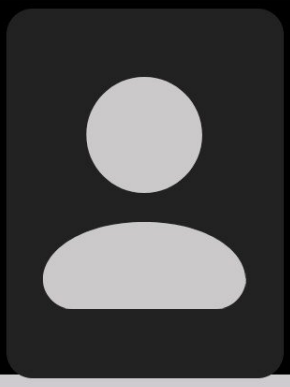
```
 THEN
  display the character in AL
 ELSE
  display the character in BL
END_IF
```

It can be coded like this:

```
                MOV  AH,2       ;prepare to display
;if AL <= BL
                CMP  AL,BL      ;AL <= BL?
                JNBE ELSE_      ;no, display char in BL
;then                           ;AL <= BL
                MOV  DL,AL      ;move char to be displayed
                JMP  DISPLAY    ;go to display
ELSE_:                          ;BL < AL
                MOV  DL,BL
```

Figure 6.3 IF-THEN-ELSE





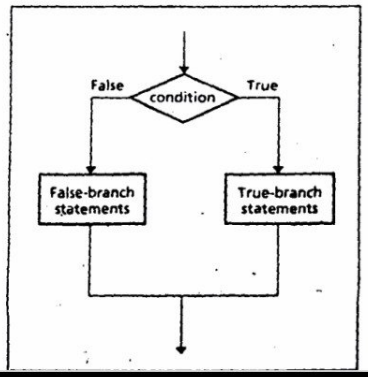

```
 THEN
   display the character in AL
 ELSE
   display the character in BL
END_IF
```

It can be coded like this:

```
                        MOV  AH,2       ;prepare to display

;if AL <= BL

                        CMP  AL,BL      ;AL <= BL?
                        JNBE ELSE_      ;no, display char in BL
                                        ;AL <= BL
;then                   MOV DL,AL       ;move char to be displayed
                        JMP DISPLAY     ;go to display
ELSE_:                                  ;BL < AL
                        MOV DL,BL
```

JLE ELSE_

Figure 6.3 IF-THEN-ELSE



```
 THEN
  display the character in AL
 ELSE
  display the character in BL
END_IF
```

It can be coded like this:

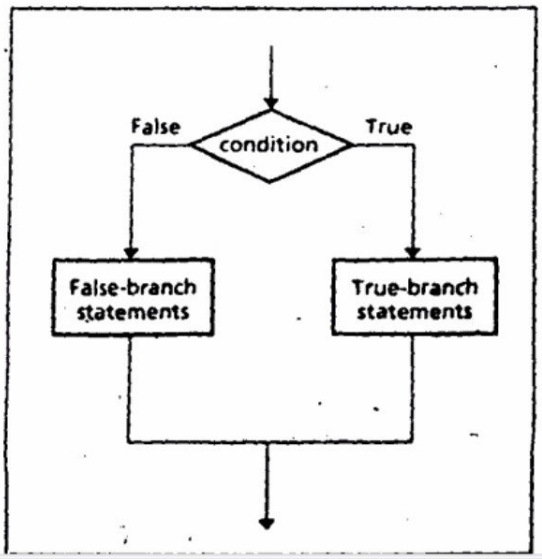
```
                    MOV  AH,2       ;prepare
;if AL <= BL
                    CMP  AL,BL      ;AL <=
                    JNBE ELSE_      ;no, dis
                                    ;AL <=
                    MOV DL,AL       ;move ch
                    JMP DISPLAY     ;go to
                                    ;BL < AL
ELSE_:              MOV DL,BL
```

CMP DES,SOU
JUMP INS(TRUE)

STATEMENTS(FALSE-ELSE)

;then

gure 6.3 IF-THEN-ELSE



Solution:

```
CASE AX
   <0: put -1 in BX
   =0: put 0 in BX
   >0: put 1 in BX
END_CASE
```

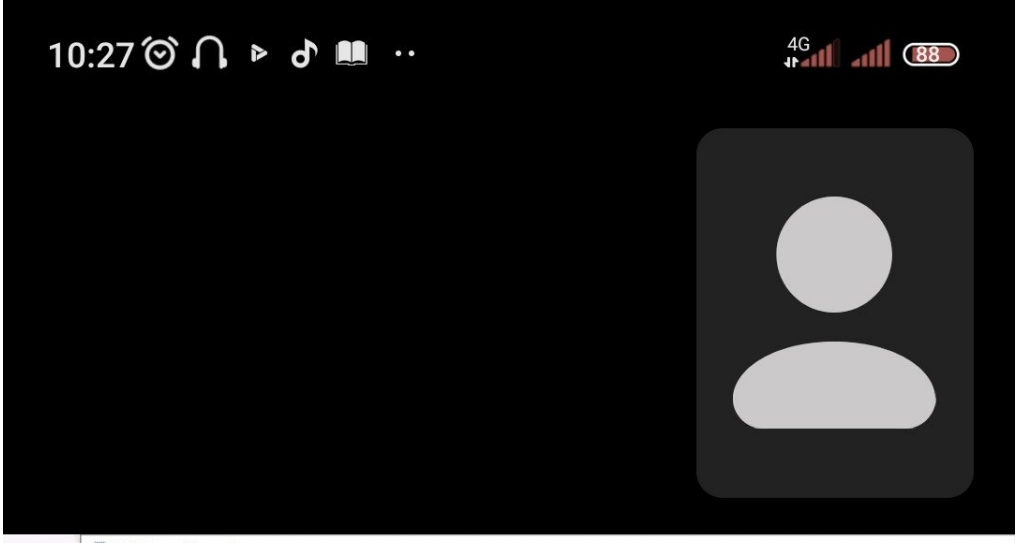
It can be coded as follows:

```
;case AX
            CMP AX,0        ;test ax
            JL  NEGATIVE    ;AX < 0
            JE  ZERO        ;AX = 0
            JG  POSITIVE    ;AX > 0
NEGATIVE:
            MOV  BX,-1      ;put -1 in BX
            JMP  END_CASE   ;and exit
ZERO:
            MOV  BX,0       ;put 0 in BX
            JMP  END_CASE   ;and exit
POSITIVE:
            MOV  BX,1       ;put 1 in BX
END_CASE:
```

Note: only one CMP is needed, because jump instructions do not affect the flags.

Example 6.5 If AL contains 1 or 3, display "o"; if AL contains 2 or 4, display "e".

Solution:

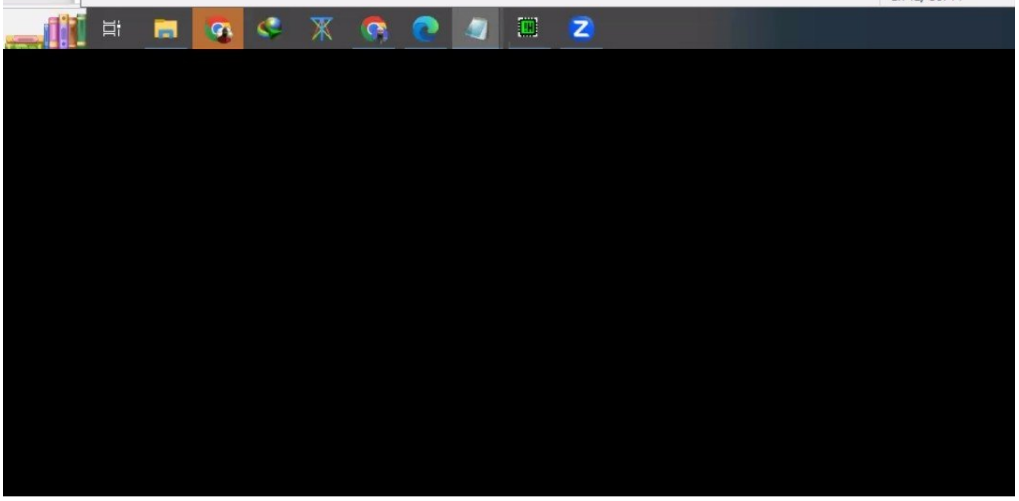


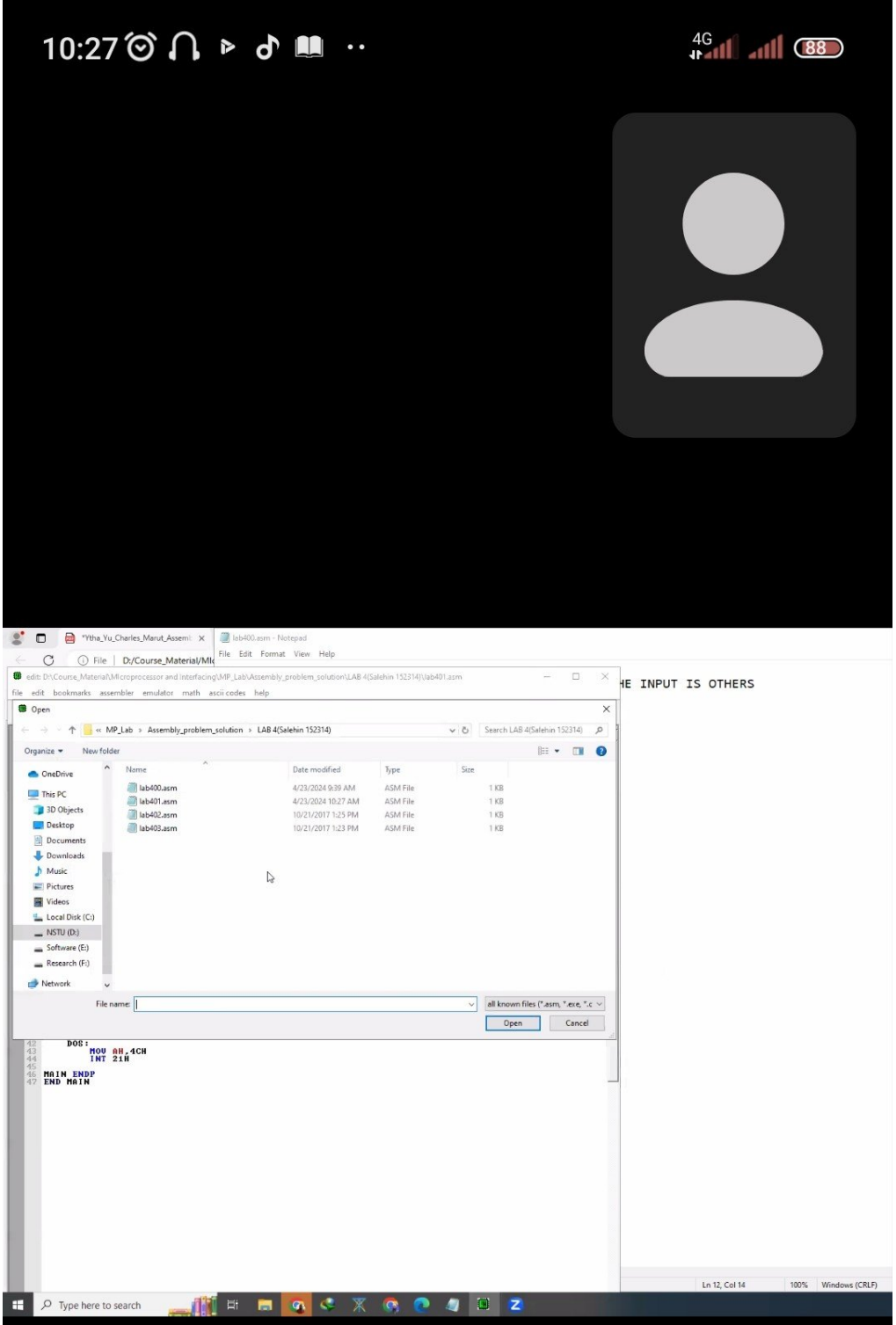
lab400.asm - Notepad

File Edit Format View Help

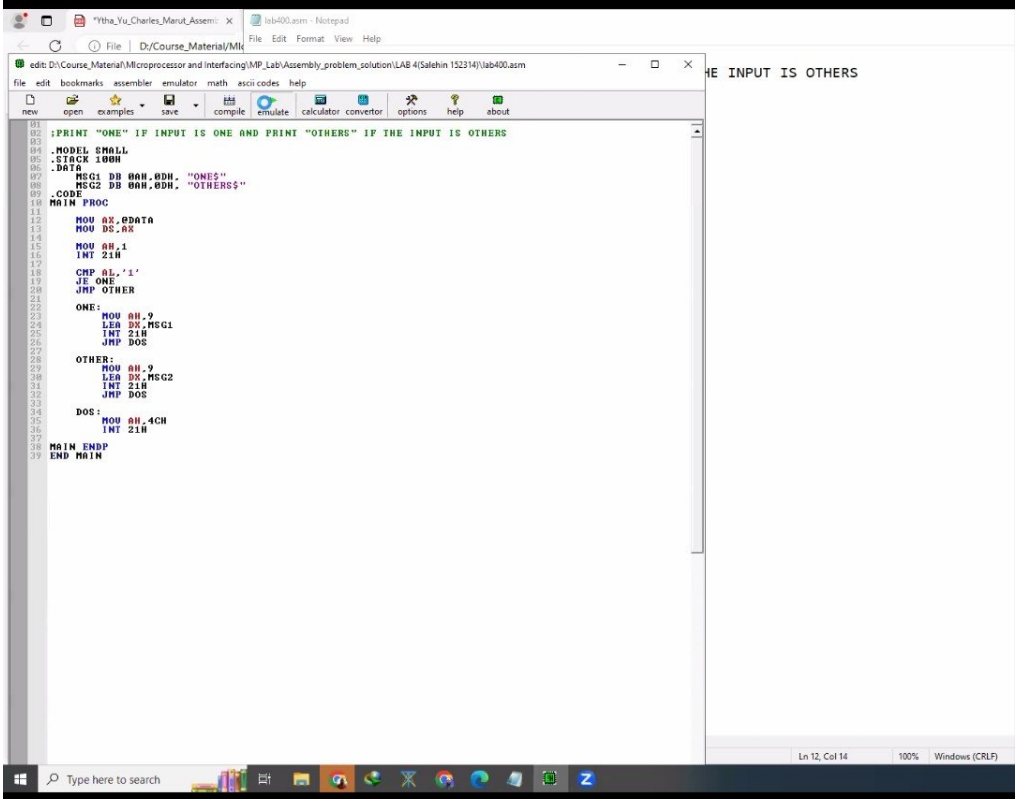
```
;PRINT "ONE" IF INPUT IS ONE AND PRINT "OTHERS" IF THE INPUT IS OTHERS

.MODEL SMALL
.STACK 100H
.DATA
        MSG1 DB 0AH,0DH, "ONE$"
        MSG2 DB 0AH,0DH, "OTHERS$"
.CODE
MAIN PROC

        MOV AX,@DATA
        MOV DS,AX

        MOV AH,1
        INT 21H

        CMP AL,'1'
        JE ONE
        JMP OTHER

        ONE:
                MOV AH,9
                LEA DX,MSG1
                INT 21H
                JMP DOS

        OTHER:
                MOV AH,9
                LEA DX,MSG2
                INT 21H
                JMP DOS

        DOS:
                MOV AH,4CH
                INT 21H

MAIN ENDP
END MAIN
```

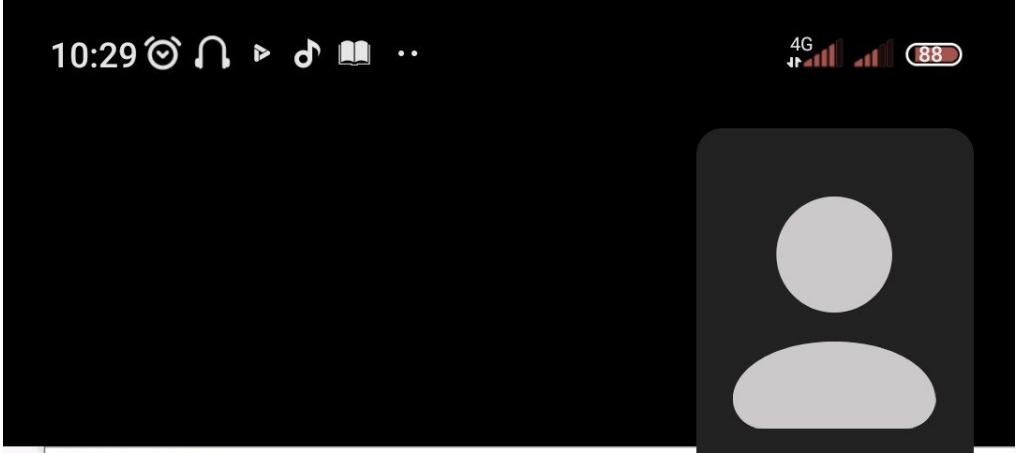
Ln 12, Col 14




10:27
4G
88

lab400.asm - Notepad
File Edit Format View Help
HE INPUT IS OTHERS
Open
MP_Lab › Assembly_problem_solution › LAB 4(Salehin 152314)
Search LAB 4(Salehin 152314)
Organize
New folder
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
NSTU (D:)
Software (E:)
Research (F:)
Network
Name
Date modified
Type
Size
lab400.asm 4/23/2024 9:39 AM ASM File 1 KB
lab401.asm 4/23/2024 10:27 AM ASM File 1 KB
lab402.asm 10/21/2017 1:25 PM ASM File 1 KB
lab403.asm 10/21/2017 1:23 PM ASM File 1 KB
File name:
all known files (*.asm, *.exe, *.c
Open
Cancel
DOS:
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
Ln 12, Col 14
100%
Windows (CRLF)
Type here to search


lab400.asm - Notepad
File Edit Format View Help
edit: D:\Course_Material\Microprocessor and Interfacing\MP_Lab\Assembly_problem_solution\LAB 4(Salehin 152314)\lab400.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
;PRINT "ONE" IF INPUT IS ONE AND PRINT "OTHERS" IF THE INPUT IS OTHERS
.MODEL SMALL
.STACK 100H
.DATA
MSG1 DB 0AH,0DH, "ONE$"
MSG2 DB 0AH,0DH, "OTHERS$"
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
MOV AH,1
INT 21H
CMP AL,'1'
JE ONE
JMP OTHER
ONE:
MOV AH,9
LEA DX,MSG1
INT 21H
JMP DOS
OTHER:
MOV AH,9
LEA DX,MSG2
INT 21H
JMP DOS
DOS:
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
HE INPUT IS OTHERS
Ln 12, Col 14
100%
Windows (CRLF)
Type here to search

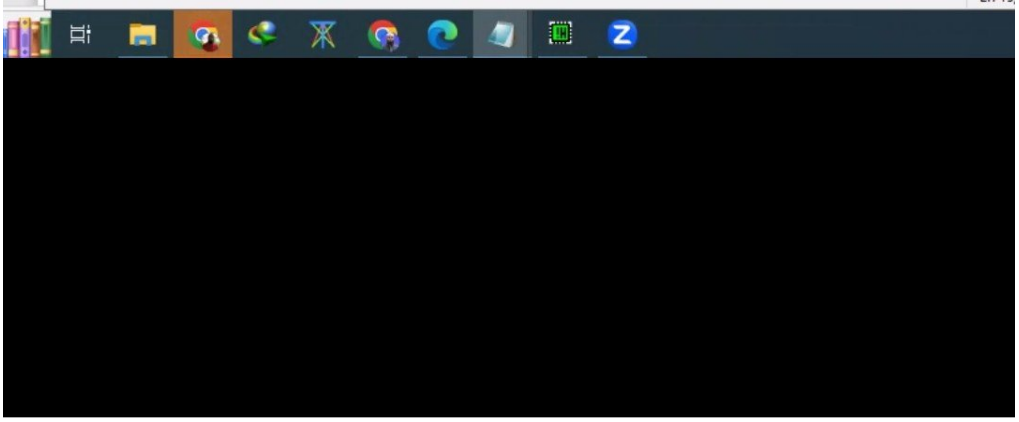


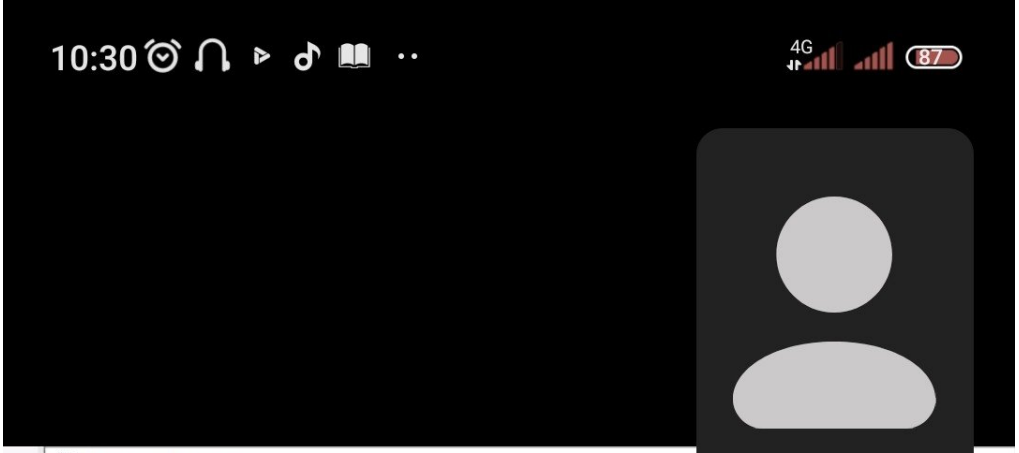
lab400.asm - Notepad

File Edit Format View Help

```
;PRINT "ONE" IF INPUT IS ONE AND PRINT "OTHERS" IF THE INPUT IS OT

.MODEL SMALL
.STACK 100H
.DATA
        MSG1 DB 0AH,0DH, "ONE$"
        MSG2 DB 0AH,0DH, "OTHERS$"
.CODE
MAIN PROC

        MOV AX,@DATA
        MOV DS,AX

        MOV AH,1
        INT 21H

        CMP AL,'1'
        JE ONE
        JMP OTHER

        ONE:
                MOV AH,9
                LEA DX,MSG1
                INT 21H
                JMP DOS

        OTHER:
                MOV AH,9
                LEA DX,MSG2
                INT 21H
                JMP DOS

        DOS:
                MOV AH,4CH
                INT 21H

MAIN ENDP
END MAIN
```

Ln 15



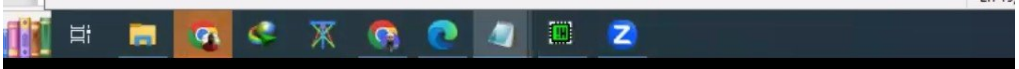


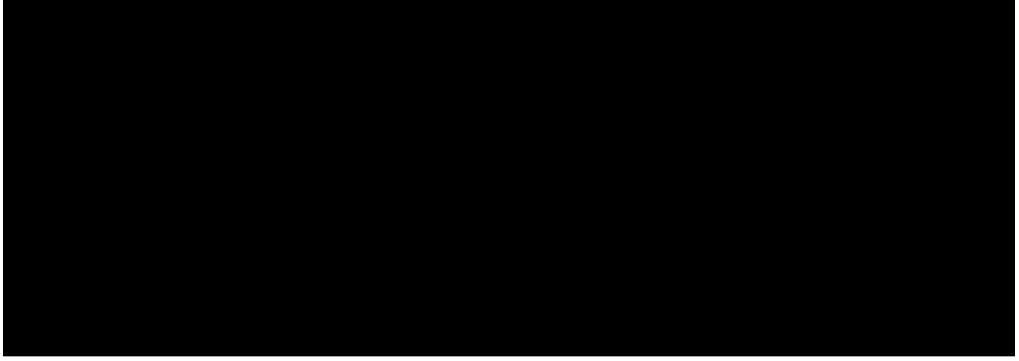
*lab400.asm - Notepad

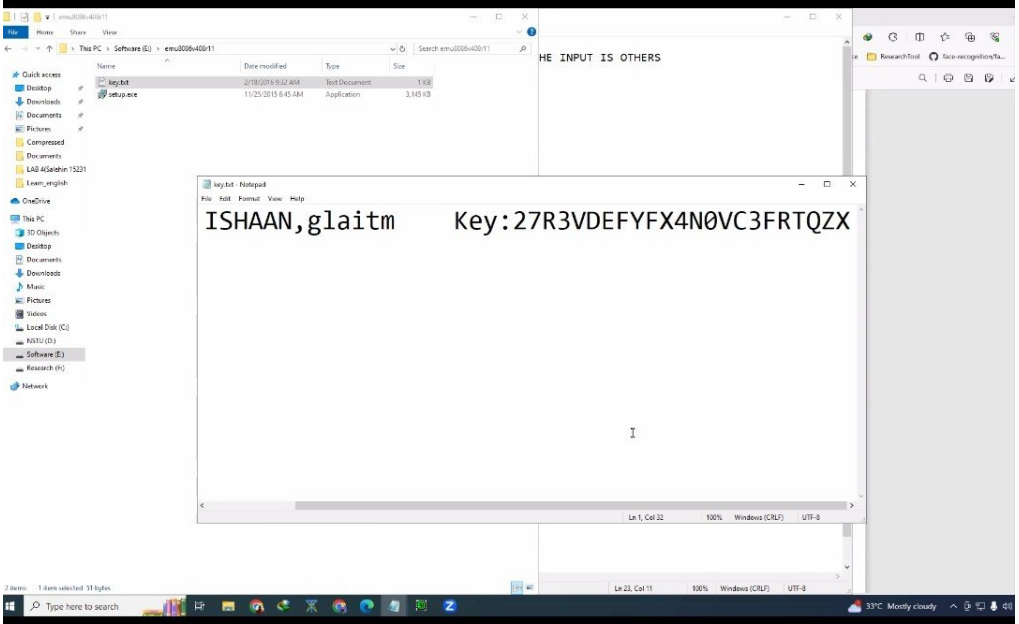
File Edit Format View Help

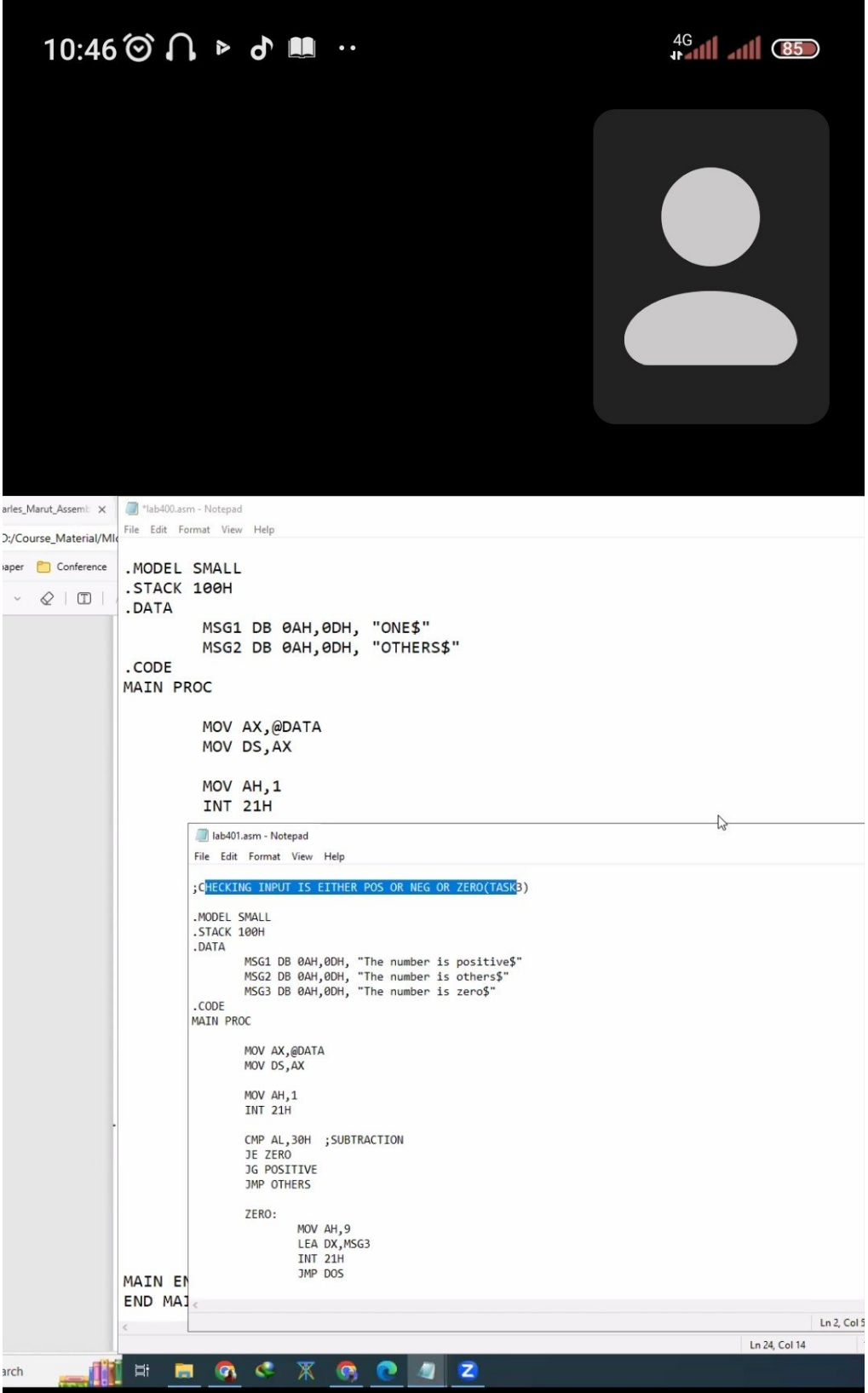
```
;PRINT "ONE" IF INPUT IS ONE AND PRINT "OTHERS" IF THE INPUT IS OT

.MODEL SMALL
.STACK 100H
.DATA
        MSG1 DB 0AH,0DH, "ONE$"
        MSG2 DB 0AH,0DH, "OTHERS$"
.CODE
MAIN PROC

        MOV AX,@DATA
        MOV DS,AX

        MOV AH,1
        INT 21H

        CMP AL,'1'
        JE ONE
        JMP OTHER

        ONE:
                MOV AH,9
                LEA DX,MSG1
                INT 21H
                JMP DOS

        OTHER:
                MOV AH,9
                LEA DX,MSG2
                INT 21H
                JMP DOS

        DOS:
                MOV AH,4CH
                INT 21H

MAIN ENDP
END MAIN
```

Ln 19

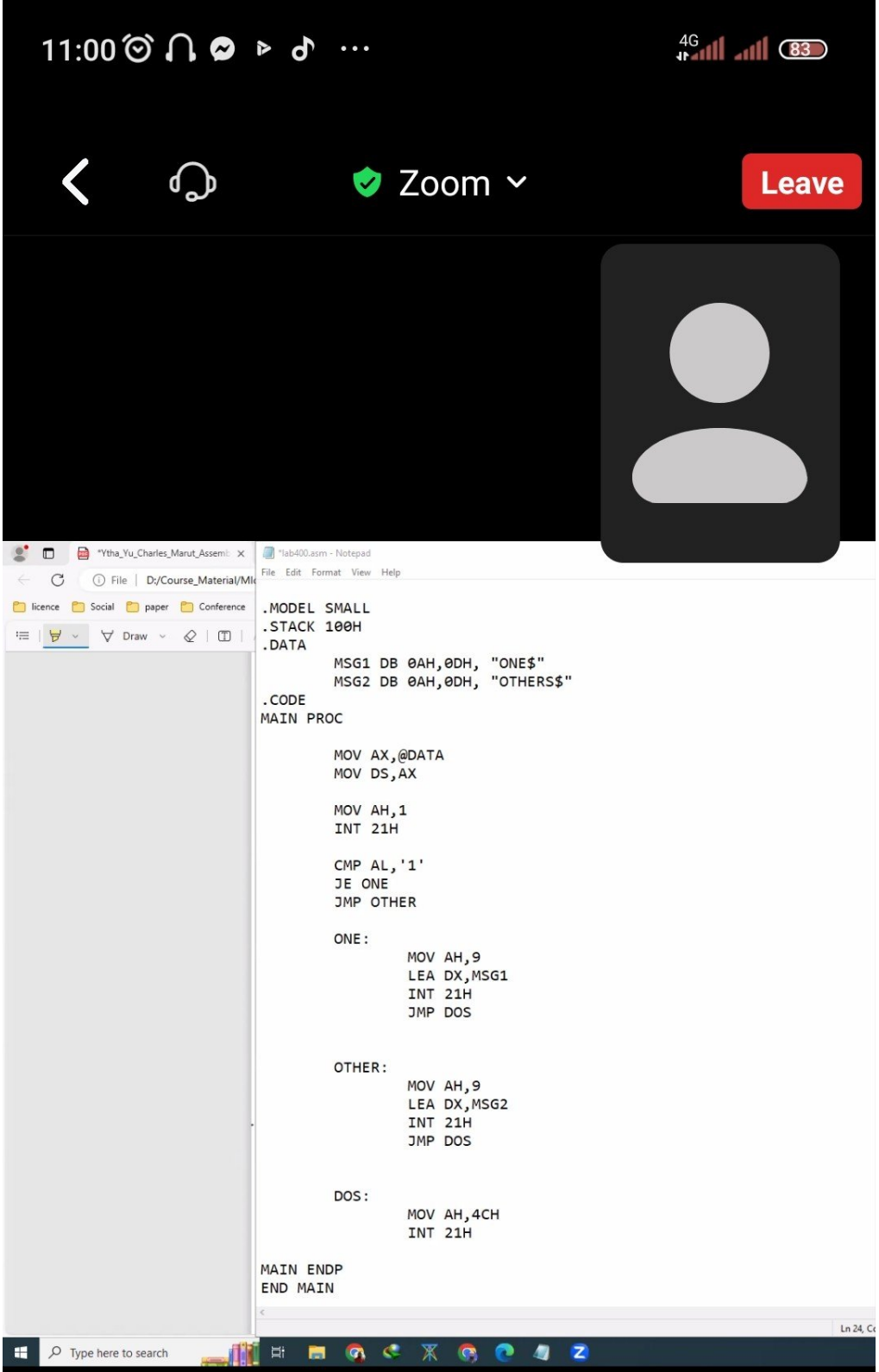


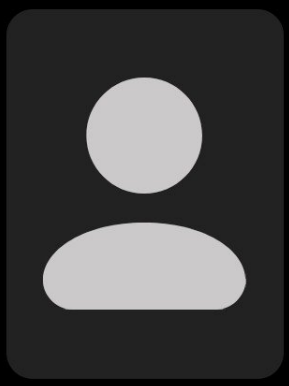


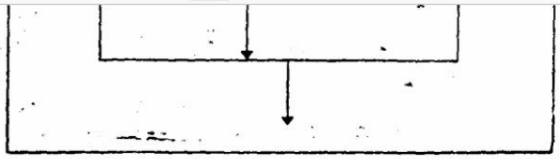

HE INPUT IS OTHERS
key.txt - Notepad
File Edit Format View Help
ISHAAN,glaitm Key:27R3VDEFYFX4N0VC3FRTQZX



Abrar Mahmud to Everyone
Shared a file in the meeting.


11:00
Zoom
Leave
*Ytha_Yu_Charles_Marut_Assemb
*lab400.asm - Notepad
File Edit Format View Help
licence Social paper Conference
Draw
.MODEL SMALL
.STACK 100H
.DATA
MSG1 DB 0AH,0DH, "ONE$"
MSG2 DB 0AH,0DH, "OTHERS$"
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
MOV AH,1
INT 21H
CMP AL,'1'
JE ONE
JMP OTHER
ONE:
MOV AH,9
LEA DX,MSG1
INT 21H
JMP DOS
OTHER:
MOV AH,9
LEA DX,MSG2
INT 21H
JMP DOS
DOS:
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
Type here to search
Unmute
Start Video
Participants
31
Chat
3
Reactions



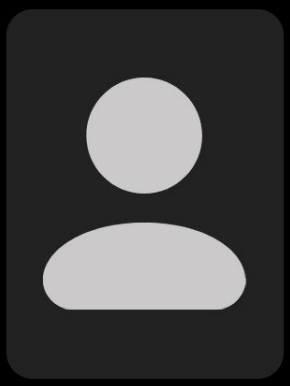


Solution:

```
CASE AX
   <0: put -1 in BX
   =0: put 0 in BX
   >0: put 1 in BX
END_CASE
```

It can be coded as follows:

```
;case AX
                CMP AX,0        ;test ax
                JL  NEGATIVE    ;AX < 0
                JE  ZERO        ;AX = 0
                JG  POSITIVE    ;AX > 0
NEGATIVE:
                MOV  BX,-1      ;put -1 in BX
                JMP  END_CASE   ;and exit
ZERO:
                MOV  BX,0       ;put 0 in BX
                JMP  END_CASE   ;and exit
POSITIVE:
```



2<A<8

IF(2<A && A<8)

CMP AL,2
JG L1

CMP AL,8
JL L1

```
DISPLAY:
                INT 21h         ;display it
END_IF
```

Note: the label ELSE_ is used because ELSE is a reserved word.

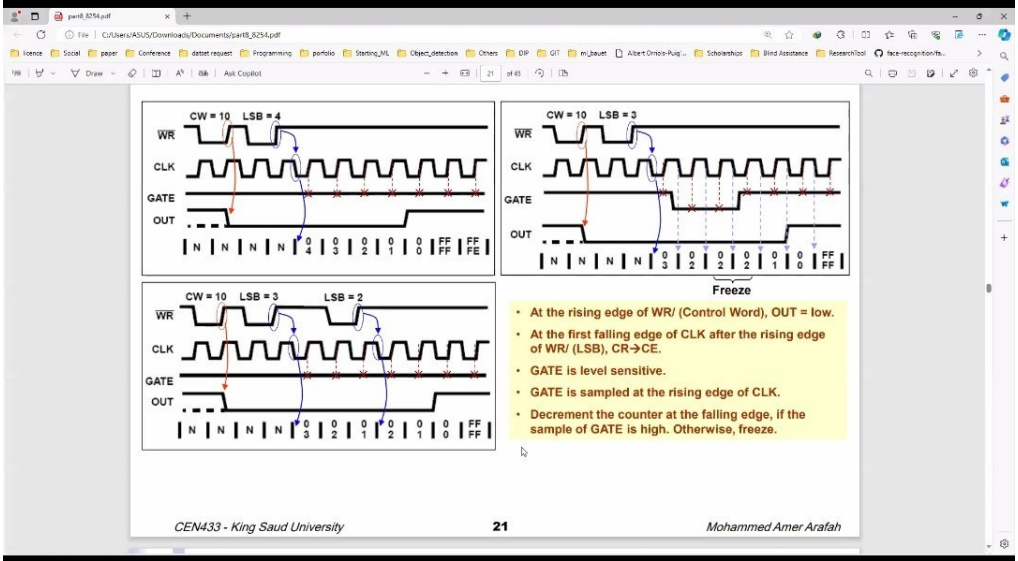
The condition AL <= BL is expressed by CMP AL,BL. If it's false, the program jumps around the true-branch statements to ELSE_. We use the unsigned jump JNBE (jump if not below or equal), because we're comparing extended characters.

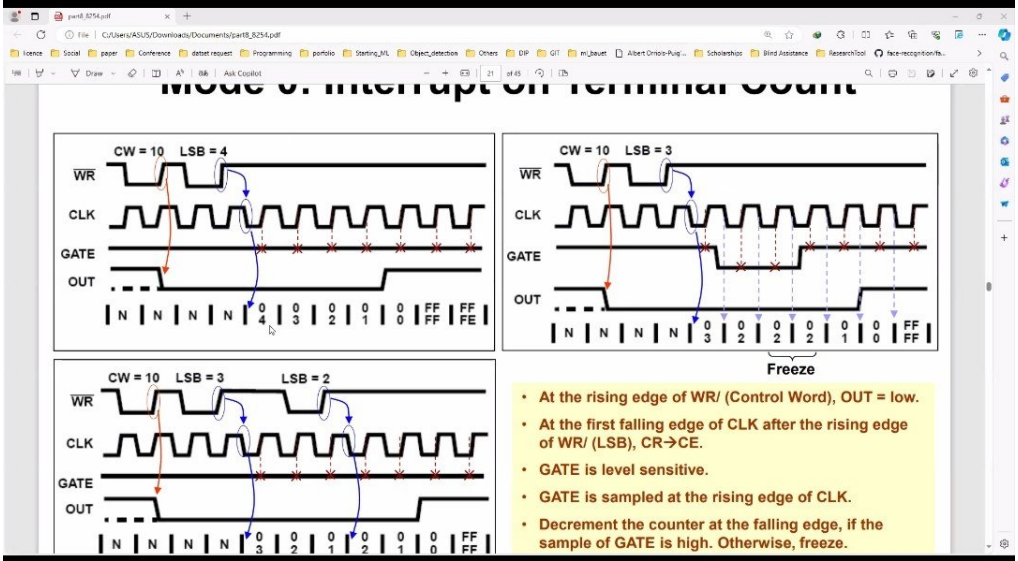
If AL <= BL is true, the true-branch statements are done. Note that JMP DISPLAY is needed to skip the false branch. This is different from the high-level language IF-THEN-ELSE, in which the false-branch statements are automatically skipped if the true-branch statements are done.

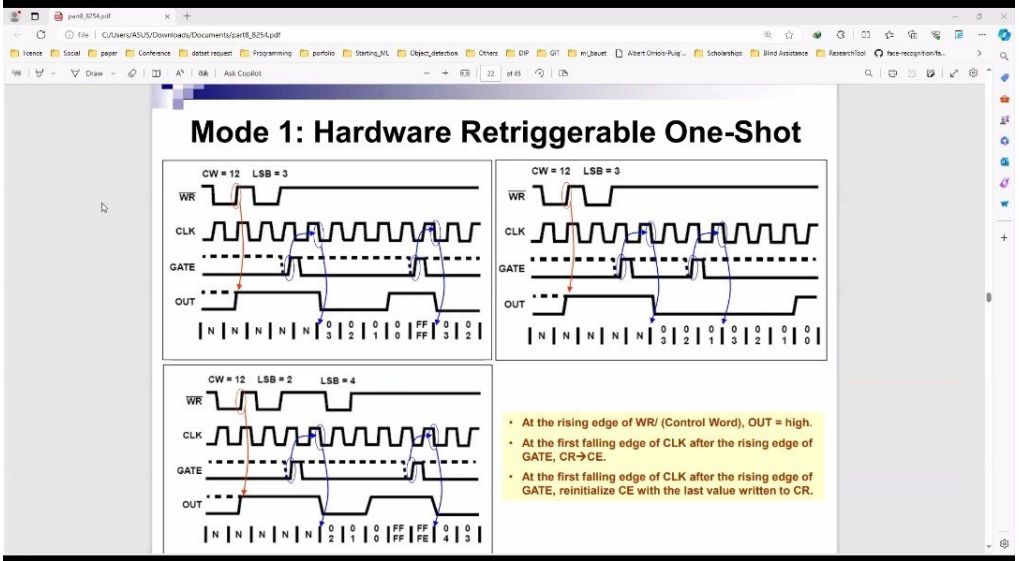
CASE

A CASE is a multiway branch structure that tests a register, variable, or expression for particular values or a range of values. The general form is as follows:

```
CASE expression
   values_1: statements_1
   values_2: statements_2
```

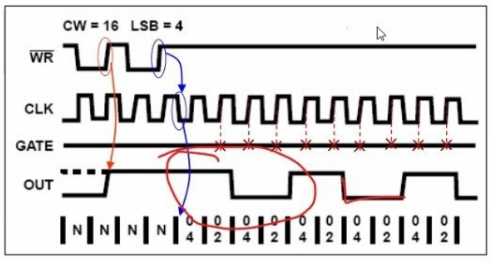


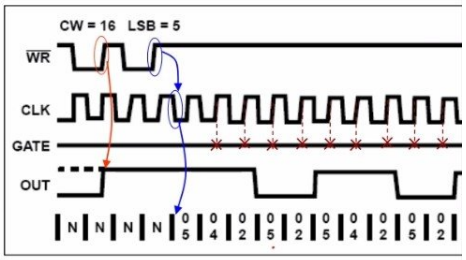


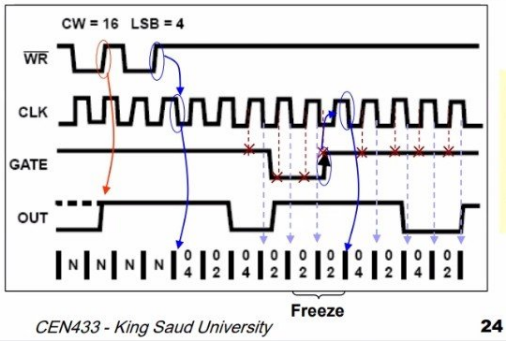

Mode 1: Hardware Retriggerable One-Shot
CW = 12 LSB = 3
WR
CLK
GATE
OUT
CW = 12 LSB = 3
CW = 12 LSB = 2 LSB = 4
At the rising edge of WR/ (Control Word), OUT = high.
At the first falling edge of CLK after the rising edge of GATE, CR→CE.
At the first falling edge of CLK after the rising edge of GATE, reinitialize CE with the last value written to CR.

4=2 4 6 8
4=2 4 6 8

Mode 3: Square Wave Mode



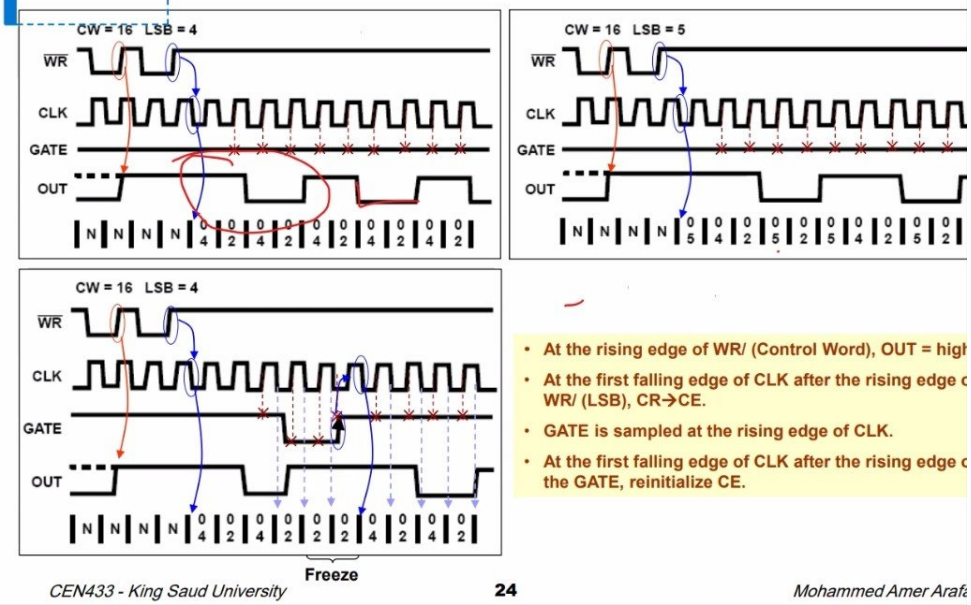




- At the rising edge of WR/ (Control Word), OUT = high
- At the first falling edge of CLK after the rising edge of WR/ (LSB), CR→CE.
- GATE is sampled at the rising edge of CLK.
- At the first falling edge of CLK after the rising edge of the GATE, reinitialize CE.

Mode 3: Square Wave Mode

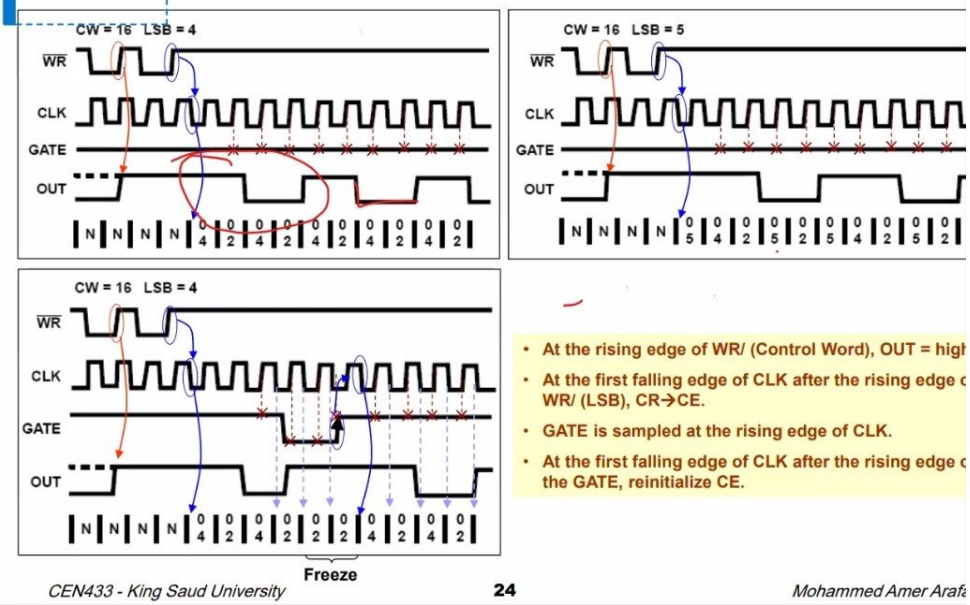
3=2 4 5
2=2 5



- At the rising edge of WR/ (Control Word), OUT = high
- At the first falling edge of CLK after the rising edge of WR/ (LSB), CR→CE.
- GATE is sampled at the rising edge of CLK.
- At the first falling edge of CLK after the rising edge of the GATE, reinitialize CE.

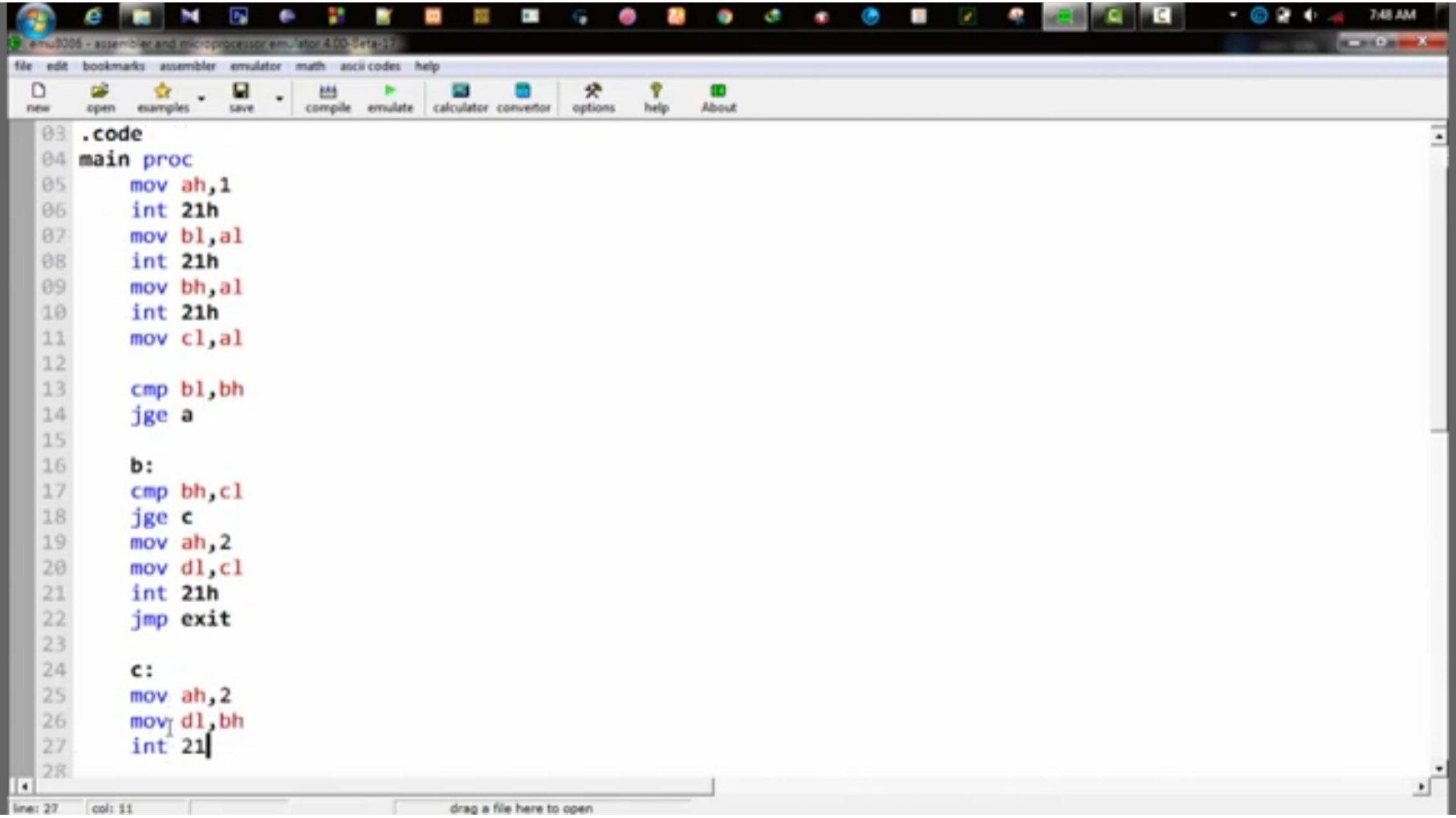
Mode 3: Square Wave Mode

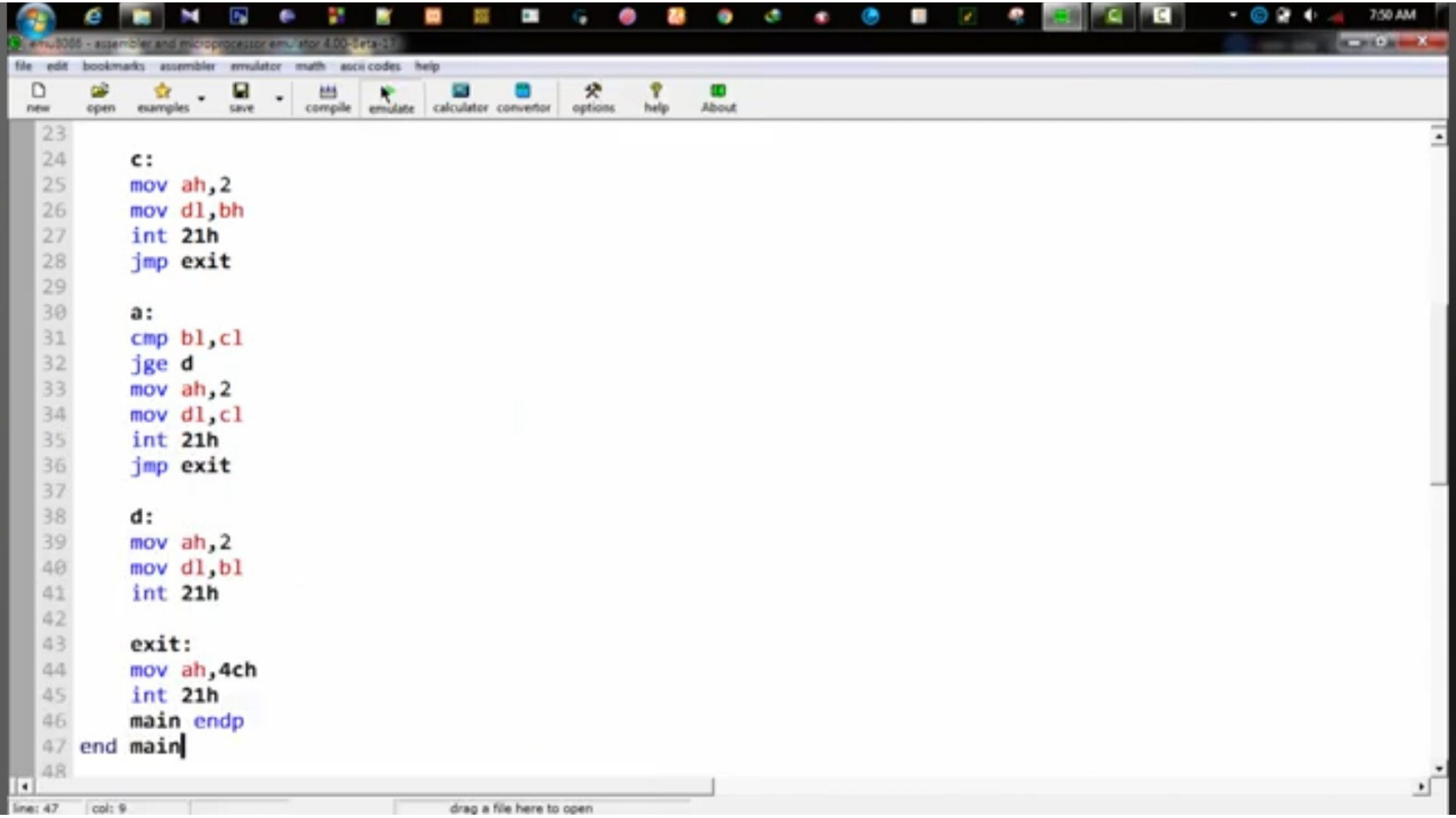
4=2467
3=247

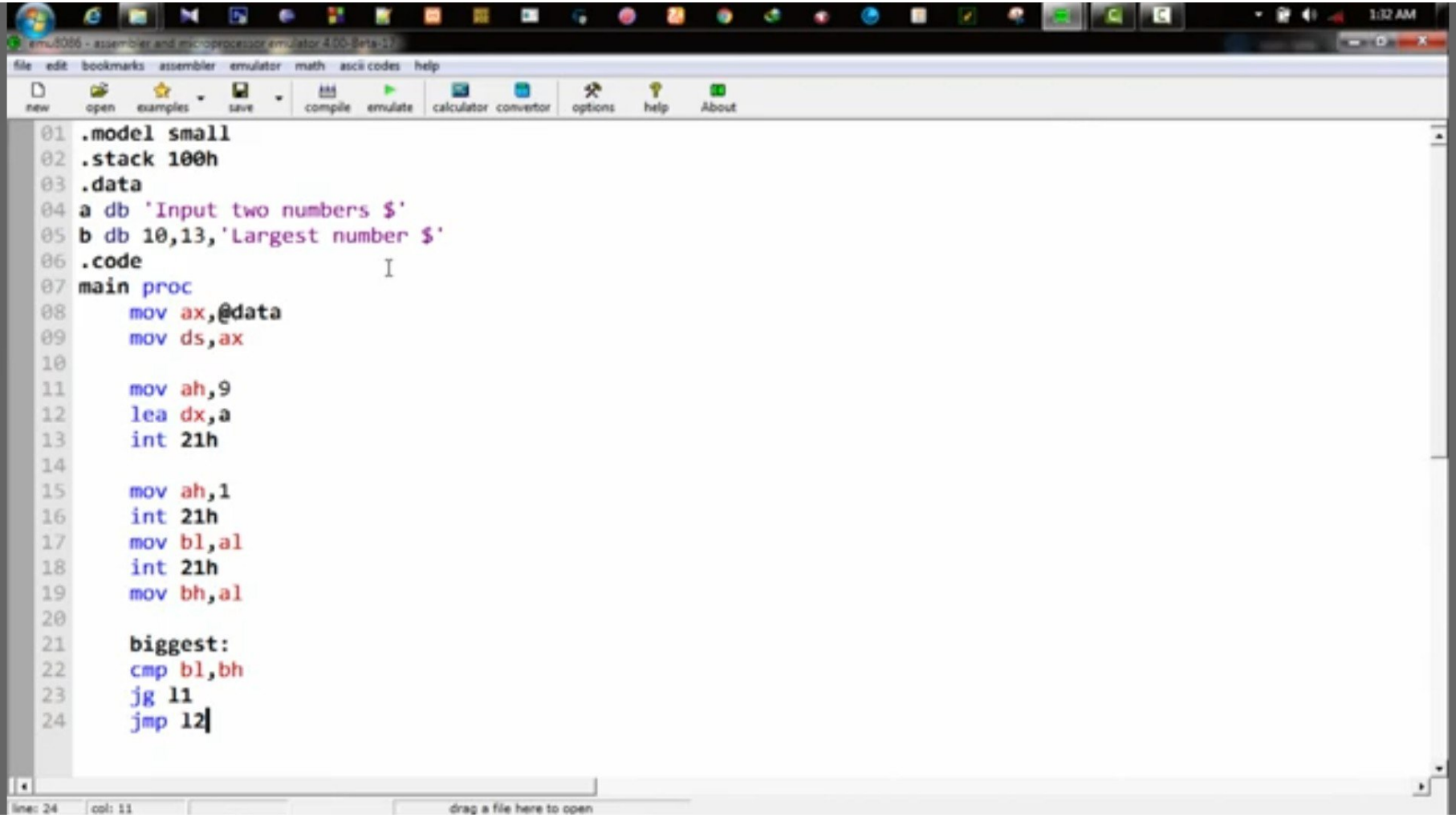


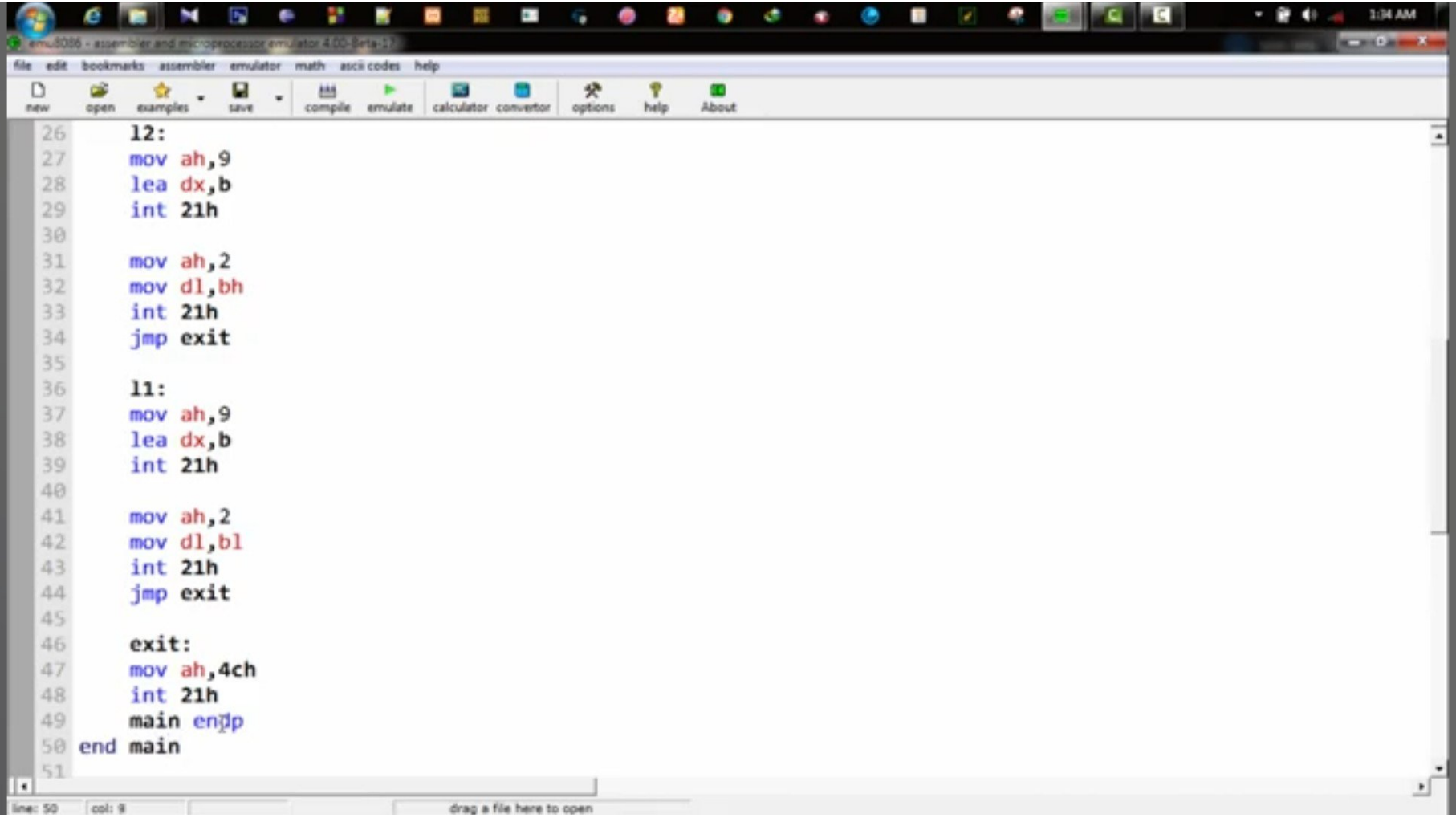
- At the rising edge of WR/ (Control Word), OUT = high
- At the first falling edge of CLK after the rising edge o WR/ (LSB), CR→CE.
- GATE is sampled at the rising edge of CLK.
- At the first falling edge of CLK after the rising edge o the GATE, reinitialize CE.

```
.MODEL SMALL
.STACK 100H
.DATA
  IN1 DB 'ENTER A VALUE $'
  NEW_LINE DB 0DH,0AH,'$'
  RES DB 'VALUE IS $'
  NUM DB 'NUMBER $'
  CP DB 'CAPITAL $'
  SM DB 'SMALL $'
.CODE

MAIN PROC
  MOV AX,@DATA
  MOV DS,AX

  MOV AH,9
  LEA DX,IN1
  INT 21H

  MOV AH ,1
  INT 21H
  MOV BL,AL

  MOV AH,9
 LEA DX,NEW_LINE
 INT 21H

  CMP BL,'0'
  JNGE END
  CMP BL,'9'
  JNLE CHECK_CAPITAL
  LEA BX,NUM
  JMP PRINT

  CHECK_CAPITAL:
      CMP BL,'A'
      JNGE END
      CMP BL,'Z'
      JNLE CHECK_SMALL
       LEA BX,CP
      JMP PRINT

  CHECK_SMALL:
       CMP BL,'a'
      JNGE END
      CMP BL,'z'
      JNLE END
      LEA BX,SM
      JMP PRINT

  PRINT:
        MOV AH,9
        LEA DX,RES
        INT 21H
        MOV AH,9
        LEA DX,BX
        INT 21H
        JMP END


  END:
     MOV AH,4CH
     INT 21H

 MAIN ENDP
END MAIN
```


emu8086 - assembler and microprocessor emulator 4.08 beta (*)
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help About
.code
main proc
mov ah,1
int 21h
mov bl,al
int 21h
mov bh,al
int 21h
mov cl,al
cmp bl,bh
jge a
b:
cmp bh,cl
jge c
mov ah,2
mov dl,cl
int 21h
jmp exit
c:
mov ah,2
mov dl,bh
int 21
line: 27 col: 11
drag a file here to open


file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help About
c:
mov ah,2
mov dl,bh
int 21h
jmp exit
a:
cmp bl,cl
jge d
mov ah,2
mov dl,cl
int 21h
jmp exit
d:
mov ah,2
mov dl,bl
int 21h
exit:
mov ah,4ch
int 21h
main endp
end main
line: 47
col: 9
drag a file here to open


file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help About
01 .model small
02 .stack 100h
03 .data
04 a db 'Input two numbers $'
05 b db 10,13,'Largest number $'
06 .code
07 main proc
08 mov ax,@data
09 mov ds,ax
10
11 mov ah,9
12 lea dx,a
13 int 21h
14
15 mov ah,1
16 int 21h
17 mov bl,al
18 int 21h
19 mov bh,al
20
21 biggest:
22 cmp bl,bh
23 jg l1
24 jmp l2
line: 24
col: 11
drag a file here to open


1:34 AM
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help About
l2:
mov ah,9
lea dx,b
int 21h
mov ah,2
mov dl,bh
int 21h
jmp exit
l1:
mov ah,9
lea dx,b
int 21h
mov ah,2
mov dl,bl
int 21h
jmp exit
exit:
mov ah,4ch
int 21h
main endp
end main
line: 50 col: 9
drag a file here to open

8259 Programmable Interrupt Controller (PIC) microprocessor:

There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086, respectively. But by connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.

Features of 8259 PIC:

- Intel 8259 is designed for Intel 8085 and Intel 8086 microprocessor.
- It can be programmed either in level triggered or in edge triggered interrupt level.
- We can mask individual bits of interrupt request register.
- **We can increase interrupt handling capability up to 64 interrupt level by cascading further 8259 PIC.**
- Clock cycle is not required.