ECE2002 - COMPUTER ORGANIZATION AND ARCHITECTURE

ASSEMBLY LANGUAGE PROGRAMS 8086 AND KEYBOARD INTERFACING

- JOB FERNANDEZ 19BCD7154

SECTION 1

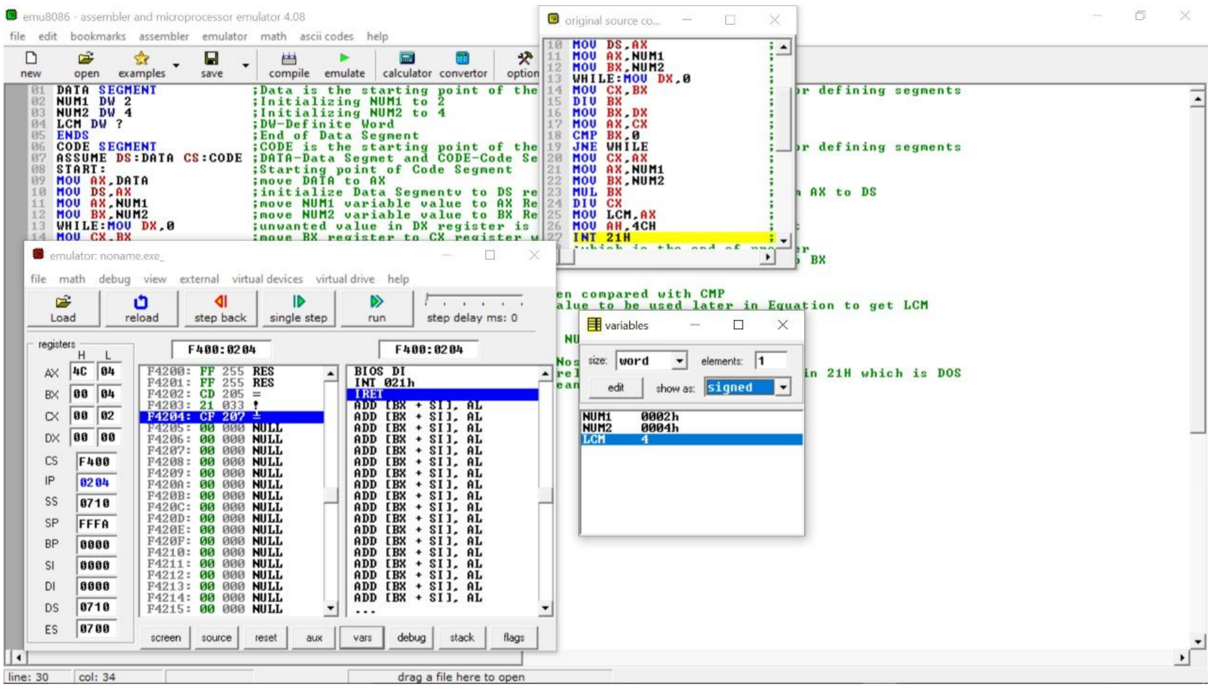
2. Write a program to compute LCM of two numbers using 8086

ALP Code:

```
DATA SEGMENT    ; Data is the starting point of the Data segment and SEGMENT is for
defining segments

NUM1 DW 2            ;Initializing NUM1 to 2

NUM2 DW 4            ; Initializing NUM2 to 4

LCM DW ?             ; DW-Definite Word

ENDS                 ; End of Data Segment

CODE SEGMENT   ; CODE is the starting point of the Code segment and SEGMENT is for
defining segments

ASSUME DS:DATA CS:CODE ; DATA-Data Segment and CODE-Code Segment

START:               ; Starting point of Code Segment

MOV AX,DATA          ; move DATA to AX

MOV DS,AX    ; initialize Data SegmentV to DS register  and then move data from AX to DS

MOV AX,NUM1         ; move NUM1 variable value to AX Register

MOV BX,NUM2         ; move NUM2 variable value to BX Register

WHILE:MOV DX,0    ; unwanted value in DX register is removed by assigning ZERO to it

MOV CX,BX            ; move BX register to CX register where CX is temporary register

DIV BX               ; divides AX:DX register with NUM2 which is passed into BX register
```

```
MOV BX,DX          ; Remainder of division is present in DX register so move value to BX

MOV AX,CX          ; move CX register to AX register

CMP BX,0           ; Compare BX register to ZERO

JNE WHILE          ; jump to label WHILE if Not Equal when compared with CMP

MOV CX,AX          ; move AX to CX to preserve the LCM value to be used later in
Equation to get LCM

MOV AX,NUM1        ; move NUM1 variable to AX Register

MOV BX,NUM2        ; move NUM2 variable to BX Register

MUL BX          ; BX register will be multiplied with NUM1

DIV CX         ; CX divides BX

MOV LCM,AX   ; move AX register(LCM of Two 16-bit Nos.) to LCM variable

MOV AH,4CH   ; Standard Input and Standard Output related Interrupts are found in 21H
which is DOS

INT 21H        ; interrupt,it works with 4CH  that means return to DOS
               ; which is the end of program
ENDS         ; end point of Code Segment

END START          ; end point
```

OUTPUT:

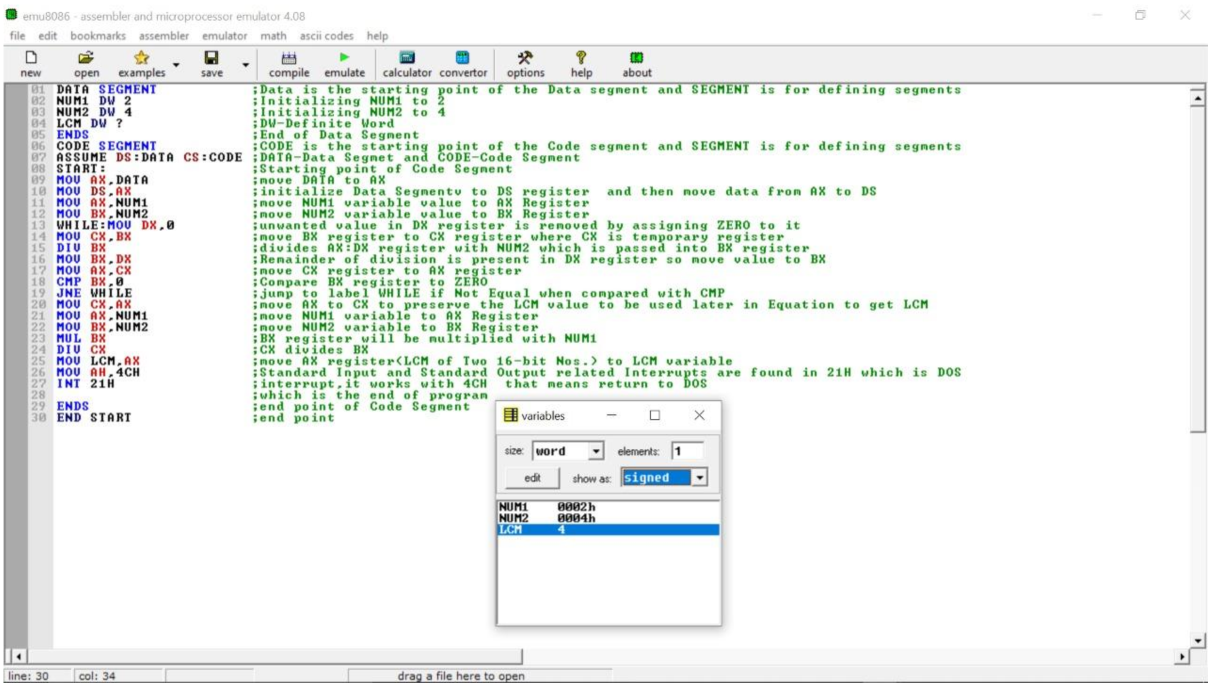


```
registers
      H    L
AX   4C   04
BX   00   04
CX   00   02
DX   00   00
CS   F400
IP   0204
SS   0710
SP   FFFA
BP   0000
SI   0000
DI   0000
DS   0710
ES   0700
```

```
F400:0204
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ±
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
F4211: 00 000 NULL
F4212: 00 000 NULL
F4213: 00 000 NULL
F4214: 00 000 NULL
F4215: 00 000 NULL
```

```
F400:0204
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

```
variables
size: word   elements: 1
show as: signed
NUM1    0002h
NUM2    0004h
LCM     4
```



```
DATA SEGMENT                ;Data is the starting point of the Data segment and SEGMENT is for defining segments
NUM1  DW 2                  ;Initializing NUM1 to 2
NUM2  DW 4                  ;Initializing NUM2 to 4
LCM DW ?                    ;DW-Definite Word
ENDS                        ;End of Data Segment
CODE SEGMENT                ;CODE is the starting point of the Code segment and SEGMENT is for defining segments
ASSUME DS:DATA CS:CODE      ;DATA-Data Segmet and CODE-Code Segment
START:                      ;Starting point of Code Segment
MOV AX,DATA                 ;move DATA to AX
MOV DS,AX                   ;initialize Data Segmentv to DS register  and then move data from AX to DS
MOV AX,NUM1                 ;move NUM1 variable value to AX Register
MOV BX,NUM2                 ;move NUM2 variable value to BX Register
WHILE:MOV DX,0              ;unwanted value in DX register is removed by assigning ZERO to it
MOV CX,BX                   ;move BX register to CX register where CX is temporary register
DIV BX                      ;divides AX:DX register with NUM2 which is passed into BX register
MOV BX,DX                   ;Remainder of division is present in DX register so move value to BX
MOV AX,CX                   ;move CX register to AX register
CMP BX,0                    ;Compare BX register to ZERO
JNE WHILE                   ;jump to label WHILE if Not Equal when compared with CMP
MOV CX,AX                   ;move AX to CX to preserve the LCM value to be used later in Equation to get LCM
MOV AX,NUM1                 ;move NUM1 variable to AX Register
MOV BX,NUM2                 ;move NUM2 variable to BX Register
MUL BX                      ;BX register will be multiplied with NUM1
DIV CX                      ;CX divides BX
MOV LCM,AX                  ;move AX register(LCM of Two 16-bit Nos.) to LCM variable
MOV AH,4CH                  ;Standard Input and Standard Output related Interrupts are found in 21H which is DOS
INT 21H                     ;interrupt,it works with 4CH  that means return to DOS
                            ;which is the end of program
ENDS                        ;end point of Code Segment
END START                   ;end point
```

```
variables
size: word   elements: 1
show as: signed
NUM1    0002h
NUM2    0004h
LCM     4
```

7. Write a program to find the GCD between two numbers using 8086

ALP Code:

DATA SEGMENT ; *DATA is the starting point of the Data segment and SEGMENT is for defining segments*

```
NUM1 DW 8               ; Initializing NUM1 to 8

NUM2 DW 10              ; Initializing NUM2 to 10

HCF DW ?                ; DW-Definite Word

ENDS                    ; End of Data Segment

CODE SEGMENT   ; CODE is the starting point of the Code segment and SEGMENT is for
defining segments

ASSUME DS:DATA CS:CODE ; Data Segment & Code Segment

START:                  ; Starting point of Code Segment

MOV AX,DATA             ; move DATA to AX

MOV DS,AX            ; initialize Data Segment to DS register and then move from AX to DS

MOV AX,NUM1             ; move NUM1 variable value to AX Register

MOV BX,NUM2             ; move NUM2 variable value to BX Register

WHILE:MOV DX,0         ; unwanted value in DX register is removed by assigning ZERO to it

MOV CX,BX               ; move BX register to CX register where CX is temporary register

DIV BX                  ; divides AX:DX register with NUM2 which is passed into BX register

MOV BX,DX               ; remainder of division is present in DX register so move value to BX

MOV AX,CX               ; move CX register to AX register

CMP BX,0                ; compare BX register to ZERO

JNE WHILE               ; jump to label WHILE if Not Equal when compared with CMP

MOV HCF,AX              ; move AX register(HCF of Two 16-bit Nos.) to HCF variable

MOV CX,AX              ; move AX to CX to preserve the HCF value to be used to get HCF

MOV AX,NUM1             ; move NUM1 variable to AX Register

MOV BX,NUM2             ; move NUM2 variable to BX Register

MUL BX                  ; BX register will be multiplied with NUM1
```

DIV CX ; *CX divides BX*

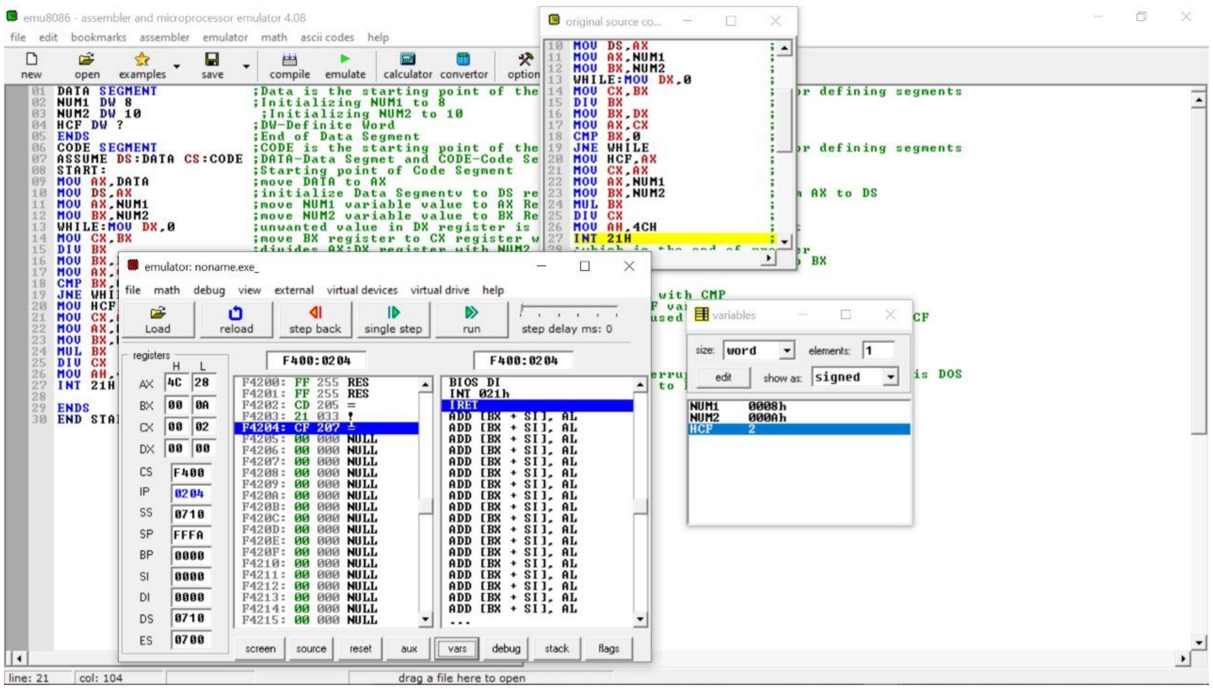
MOV AH,4CH ;*Standard Input and Standard Output related Interrupts are found in 21H which is DOS*

INT 21H ; *interrupt,it works with 4CH that means return to DOS*
; *which is the end of program*

ENDS ; *end point of Code Segment*

END START ; *end point*

OUTPUT:



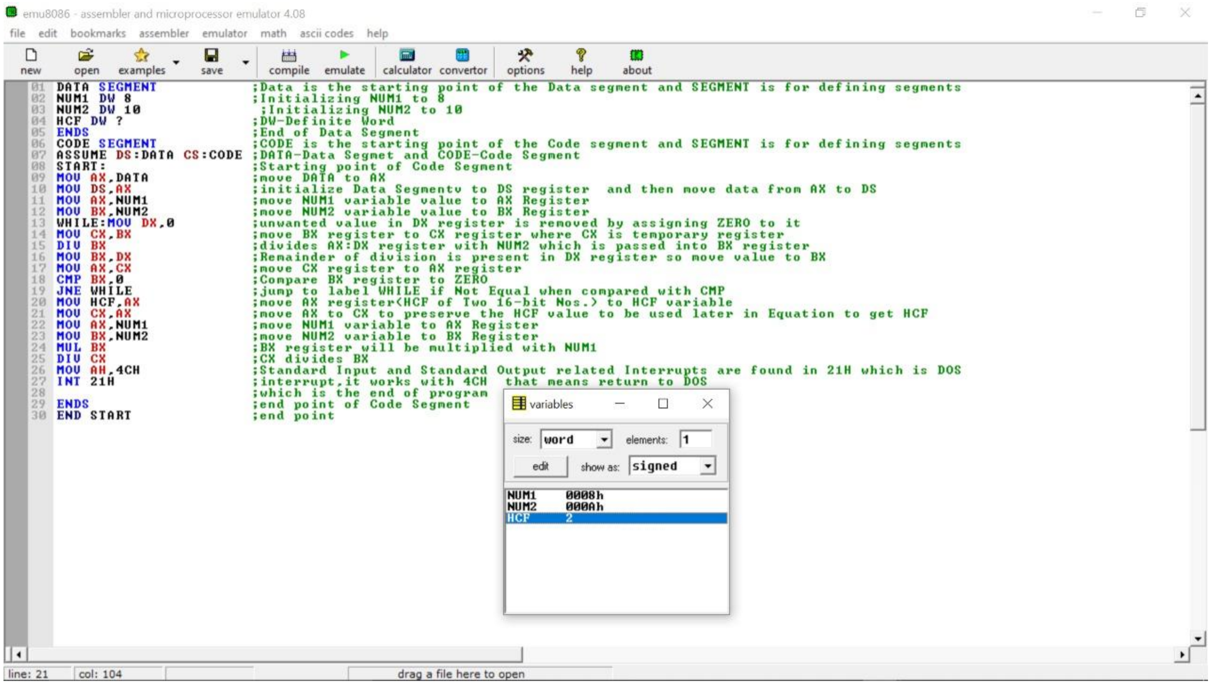
emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor option
emulator: noname.exe_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
original source co...
variables
line: 21 col: 104 drag a file here to open

```
DATA SEGMENT                ;Data is the starting point of the
NUM1 DW 8                   ;Initializing NUM1 to 8
NUM2 DW 10                   ;Initializing NUM2 to 10
HCF DW ?                    ;DW-Definite Word
ENDS                        ;End of Data Segment
CODE SEGMENT                ;CODE is the starting point of the
ASSUME DS:DATA CS:CODE      ;DATA-Data Segmet and CODE-Code Se
START:                      ;Starting point of Code Segment
MOV AX,DATA                 ;move DATA to AX
MOV DS,AX                   ;initialize Data Segmentv to DS re
MOV AX,NUM1                 ;move NUM1 variable value to AX Re
MOV BX,NUM2                 ;move NUM2 variable value to BX Re
WHILE:MOV DX,0              ;unwanted value in DX register is
MOV CX,BX                   ;move BX register to CX register w
DIV BX
MOV BX,DX
MOV AX,CX
CMP BX,0
JNE WHILE
MOV HCF,AX
MOV CX,AX
MOV AX,NUM1
MOV BX,NUM2
MUL BX
DIV CX
MOV AH,4CH
INT 21H

ENDS
END START
```

```
registers
     H   L
AX   4C  28
BX   00  0A
CX   00  02
DX   00  00
CS   F400
IP   0204
SS   0710
SP   FFFA
BP   0000
SI   0000
DI   0000
DS   0710
ES   0700
```

```
F400:0204                    F400:0204
F4200: FF 255 RES            BIOS DI
F4201: FF 255 RES            INT 021h
F4202: CD 205 =              IRET
F4203: 21 033 !              ADD [BX + SI], AL
F4204: CF 207 ±              ADD [BX + SI], AL
F4205: 00 000 NULL           ADD [BX + SI], AL
F4206: 00 000 NULL           ADD [BX + SI], AL
F4207: 00 000 NULL           ADD [BX + SI], AL
F4208: 00 000 NULL           ADD [BX + SI], AL
F4209: 00 000 NULL           ADD [BX + SI], AL
F420A: 00 000 NULL           ADD [BX + SI], AL
F420B: 00 000 NULL           ADD [BX + SI], AL
F420C: 00 000 NULL           ADD [BX + SI], AL
F420D: 00 000 NULL           ADD [BX + SI], AL
F420E: 00 000 NULL           ADD [BX + SI], AL
F420F: 00 000 NULL           ADD [BX + SI], AL
F4210: 00 000 NULL           ADD [BX + SI], AL
F4211: 00 000 NULL           ADD [BX + SI], AL
F4212: 00 000 NULL           ADD [BX + SI], AL
F4213: 00 000 NULL           ADD [BX + SI], AL
F4214: 00 000 NULL           ADD [BX + SI], AL
F4215: 00 000 NULL           ...
```

```
size: word    elements: 1
show as: signed

NUM1     0008h
NUM2     000Ah
HCF      2
```


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: 21 col: 104 drag a file here to open

```
DATA SEGMENT              ;Data is the starting point of the Data segment and SEGMENT is for defining segments
NUM1 DW 8                 ;Initializing NUM1 to 8
NUM2 DW 10                 ;Initializing NUM2 to 10
HCF DW ?                  ;DW-Definite Word
ENDS                      ;End of Data Segment
CODE SEGMENT              ;CODE is the starting point of the Code segment and SEGMENT is for defining segments
ASSUME DS:DATA CS:CODE    ;DATA-Data Segmet and CODE-Code Segment
START:                    ;Starting point of Code Segment
MOV AX,DATA               ;move DATA to AX
MOV DS,AX                 ;initialize Data Segmentv to DS register  and then move data from AX to DS
MOV AX,NUM1               ;move NUM1 variable value to AX Register
MOV BX,NUM2               ;move NUM2 variable value to BX Register
WHILE:MOV DX,0            ;unwanted value in DX register is removed by assigning ZERO to it
MOV CX,BX                 ;move BX register to CX register where CX is temporary register
DIV BX                    ;divides AX:DX register with NUM2 which is passed into BX register
MOV BX,DX                 ;Remainder of division is present in DX register so move value to BX
MOV AX,CX                 ;move CX register to AX register
CMP BX,0                  ;Compare BX register to ZERO
JNE WHILE                 ;jump to label WHILE if Not Equal when compared with CMP
MOV HCF,AX                ;move AX register(HCF of Two 16-bit Nos.) to HCF variable
MOV CX,AX                 ;move AX to CX to preserve the HCF value to be used later in Equation to get HCF
MOV AX,NUM1               ;move NUM1 variable to AX Register
MOV BX,NUM2               ;move NUM2 variable to BX Register
MUL BX                    ;BX register will be multiplied with NUM1
DIV CX                    ;CX divides BX
MOV AH,4CH                ;Standard Input and Standard Output related Interrupts are found in 21H which is DOS
INT 21H                   ;interrupt,it works with 4CH  that means return to DOS
                          ;which is the end of program
ENDS                      ;end point of Code Segment
END START                 ;end point
```

```
variables
size: word    elements: 1
edit    show as: signed

NUM1    0008h
NUM2    000Ah
HCF     2
```

9. Write a program for arranging a set of data in ascending order using 8086

ALP Code:

MOV SI,1200H ; *initialize memory location for array size 'n'.*

MOV CL, [SI] ; *initialize the count to array size (i.e., 'n').*

DEC CL ; *Decrement CL by 1 to represent number to repetitions (i.e., 'n-1')required.*

REPEAT : MOV SI,1200H ; *Initialize pointer*

MOV CH, [SI] ; *Get the array size in CH*

DEC CH ; *Set CH as count for n-1 comparisons.*

INC SI ; *Increment the pointer.*

REPCOM : MOV AL, [SI] ; *Get an element of array in AL register.*

INC SI ; *Increment the pointer.*

CMP AL, [SI] ; *Compare with the next element of the array in memory.*

JC AHEAD ; *if AL<next element of array ,then go to AHEAD*

XCHG AL, [SI] ; *Exchange data between memory locations pointed by SI and SI-1*

XCHG AL, [SI-1] ; *[SI]<->[SI-1]*

AHEAD : DEC CH ; *Decrement count for comparisons.*

JNZ REPCOM ; *Repeat comparisons until CH count is zero.*

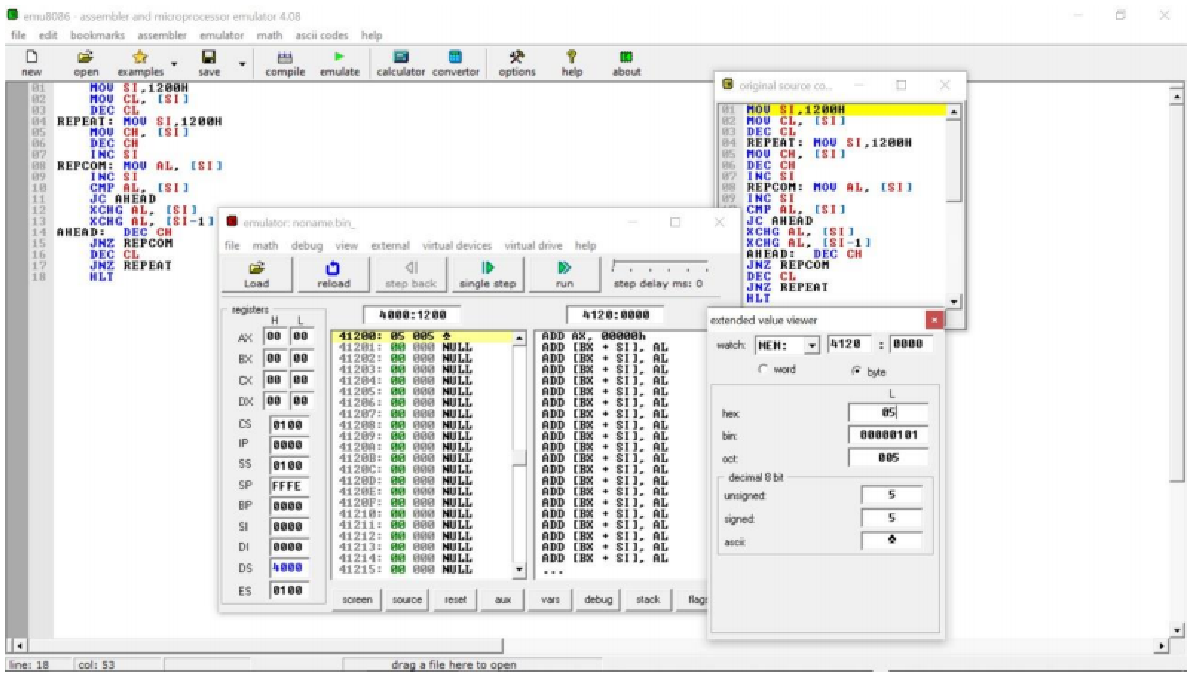
DEC CL ; *Decrement count for repetitions.*

JNZ REPEAT ; *Repeat n-1 comparisons until CL count is zero.*

HLT ; *Stop the program.*

INPUT:

Array size = 5


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
emulator: noname.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run
screen source reset aux vars debug stack flags
original source co...
line: 18 col: 53 drag a file here to open

```
      MOV SI,1200H
      MOV CL, [SI]
      DEC CL
REPEAT: MOV SI,1200H
      MOV CH, [SI]
      DEC CH
      INC SI
REPCOM: MOV AL, [SI]
      INC SI
      CMP AL, [SI]
      JC AHEAD
      XCHG AL, [SI]
      XCHG AL, [SI-1]
AHEAD:  DEC CH
      JNZ REPCOM
      DEC CL
      JNZ REPEAT
      HLT
```

step delay ms: 0

registers

```
    H  L
AX  00 00
BX  00 00
CX  00 00
DX  00 00
CS  0100
IP  0000
SS  0100
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  4000
ES  0100
```

```
4000:1200                   4120:0000
41200: 05 005 ♠             ADD AX, 00000h
41201: 00 000 NULL          ADD [BX + SI], AL
41202: 00 000 NULL          ADD [BX + SI], AL
41203: 00 000 NULL          ADD [BX + SI], AL
41204: 00 000 NULL          ADD [BX + SI], AL
41205: 00 000 NULL          ADD [BX + SI], AL
41206: 00 000 NULL          ADD [BX + SI], AL
41207: 00 000 NULL          ADD [BX + SI], AL
41208: 00 000 NULL          ADD [BX + SI], AL
41209: 00 000 NULL          ADD [BX + SI], AL
4120A: 00 000 NULL          ADD [BX + SI], AL
4120B: 00 000 NULL          ADD [BX + SI], AL
4120C: 00 000 NULL          ADD [BX + SI], AL
4120D: 00 000 NULL          ADD [BX + SI], AL
4120E: 00 000 NULL          ADD [BX + SI], AL
4120F: 00 000 NULL          ADD [BX + SI], AL
41210: 00 000 NULL          ADD [BX + SI], AL
41211: 00 000 NULL          ADD [BX + SI], AL
41212: 00 000 NULL          ADD [BX + SI], AL
41213: 00 000 NULL          ADD [BX + SI], AL
41214: 00 000 NULL          ADD [BX + SI], AL
41215: 00 000 NULL          ...
```

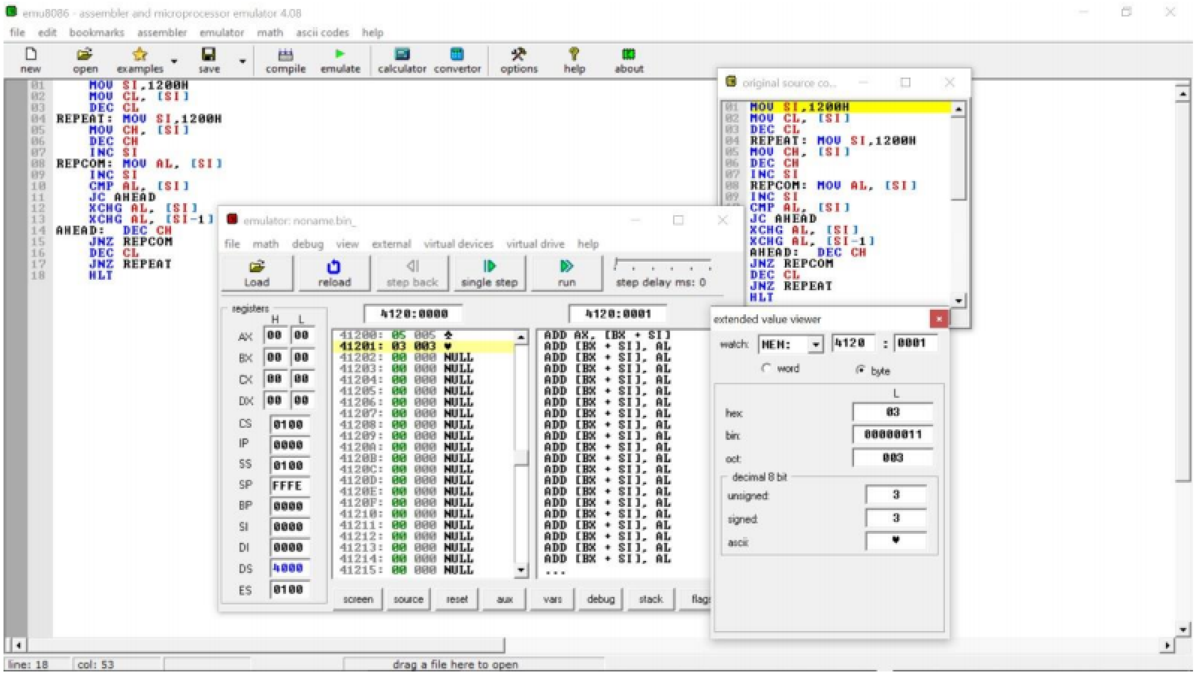
extended value viewer

watch: MEM: 4120 : 0000

word [✓ byte]

```
           L
hex:       05
bin:       00000101
oct:       005
decimal 8 bit
unsigned:  5
signed:    5
ascii:     ♠
```

Input: 3


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
emulator: noname.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
original source co...
line: 18 col: 53 drag a file here to open

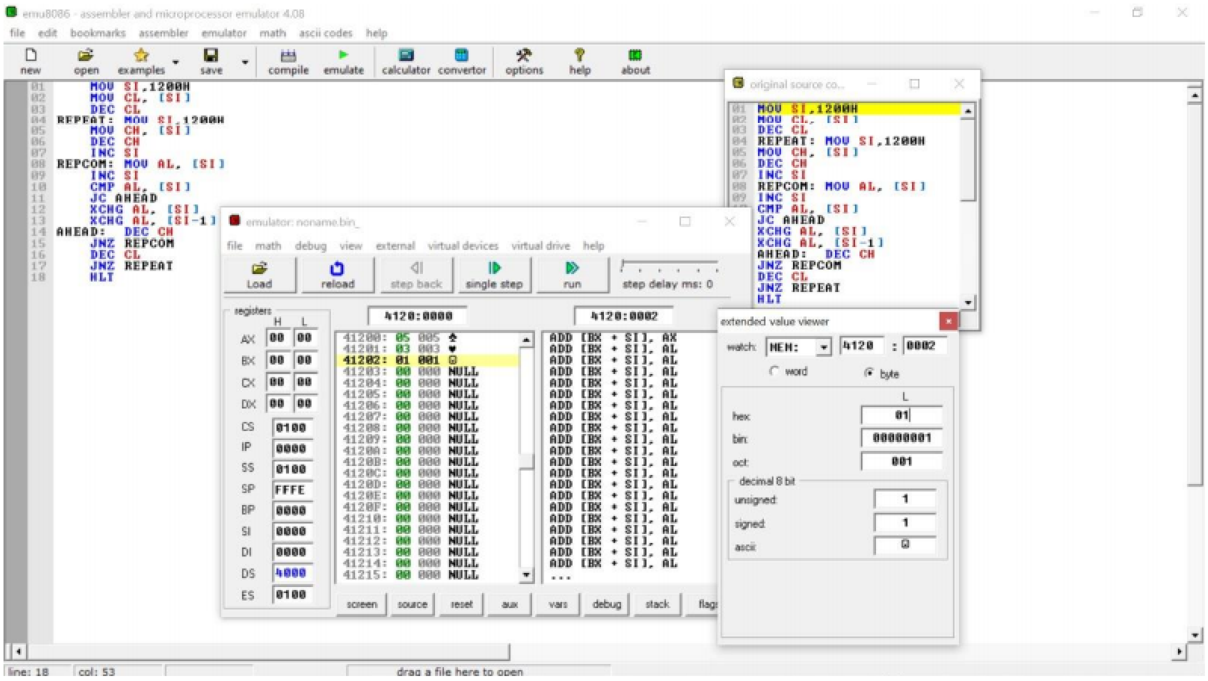
```
        MOV SI,1200H
        MOV CL, [SI]
        DEC CL
REPEAT: MOV SI,1200H
        MOV CH, [SI]
        DEC CH
        INC SI
REPCOM: MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC AHEAD
        XCHG AL, [SI]
        XCHG AL, [SI-1]
AHEAD:  DEC CH
        JNZ REPCOM
        DEC CL
        JNZ REPEAT
        HLT
```

```
registers  H   L
AX        00  00
BX        00  00
CX        00  00
DX        00  00
CS      0100
IP      0000
SS      0100
SP      FFFE
BP      0000
SI      0000
DI      0000
DS      4000
ES      0100
```

```
4120:0000                    4120:0001
41200: 05 005 ♣              ADD AX, [BX + SI]
41201: 03 003 ♥              ADD [BX + SI], AL
41202: 00 000 NULL           ADD [BX + SI], AL
41203: 00 000 NULL           ADD [BX + SI], AL
41204: 00 000 NULL           ADD [BX + SI], AL
41205: 00 000 NULL           ADD [BX + SI], AL
41206: 00 000 NULL           ADD [BX + SI], AL
41207: 00 000 NULL           ADD [BX + SI], AL
41208: 00 000 NULL           ADD [BX + SI], AL
41209: 00 000 NULL           ADD [BX + SI], AL
4120A: 00 000 NULL           ADD [BX + SI], AL
4120B: 00 000 NULL           ADD [BX + SI], AL
4120C: 00 000 NULL           ADD [BX + SI], AL
4120D: 00 000 NULL           ADD [BX + SI], AL
4120E: 00 000 NULL           ADD [BX + SI], AL
4120F: 00 000 NULL           ADD [BX + SI], AL
41210: 00 000 NULL           ADD [BX + SI], AL
41211: 00 000 NULL           ADD [BX + SI], AL
41212: 00 000 NULL           ADD [BX + SI], AL
41213: 00 000 NULL           ADD [BX + SI], AL
41214: 00 000 NULL           ADD [BX + SI], AL
41215: 00 000 NULL           ...
```

```
extended value viewer
watch: MEM: 4120 : 0001
( ) word  (•) byte
                 L
hex:            03
bin:      00000011
oct:           003
decimal 8 bit
unsigned:        3
signed:          3
ascii:           ♥
```

Input: 1


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
emulator: noname.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
original source co...
line: 18 col: 53 drag a file here to open

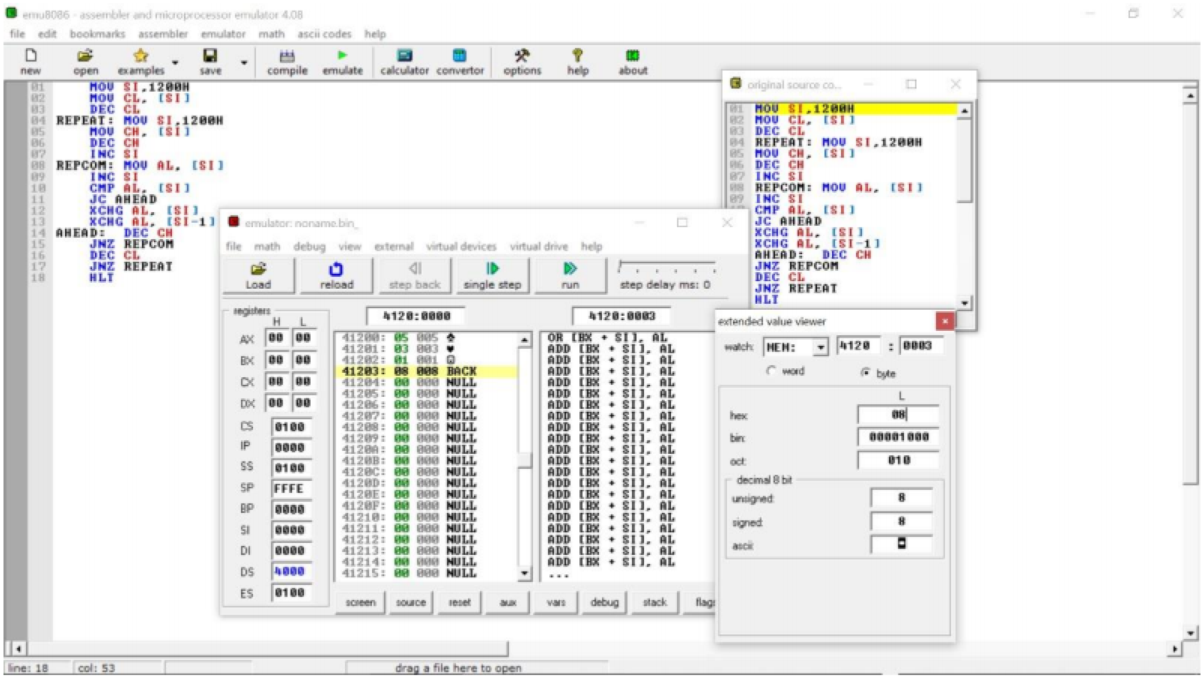
```
        MOV SI,1200H
        MOV CL, [SI]
        DEC CL
REPEAT: MOV SI,1200H
        MOV CH, [SI]
        DEC CH
        INC SI
REPCOM: MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC AHEAD
        XCHG AL, [SI]
        XCHG AL, [SI-1]
AHEAD:  DEC CH
        JNZ REPCOM
        DEC CL
        JNZ REPEAT
        HLT
```

```
registers  H   L
AX        00  00
BX        00  00
CX        00  00
DX        00  00
CS      0100
IP      0000
SS      0100
SP      FFFE
BP      0000
SI      0000
DI      0000
DS      4000
ES      0100
```

```
4120:0000                    4120:0002
41200: 05 005 ♣              ADD [BX + SI], AX
41201: 03 003 ♥              ADD [BX + SI], AL
41202: 01 001 ☺              ADD [BX + SI], AL
41203: 00 000 NULL           ADD [BX + SI], AL
41204: 00 000 NULL           ADD [BX + SI], AL
41205: 00 000 NULL           ADD [BX + SI], AL
41206: 00 000 NULL           ADD [BX + SI], AL
41207: 00 000 NULL           ADD [BX + SI], AL
41208: 00 000 NULL           ADD [BX + SI], AL
41209: 00 000 NULL           ADD [BX + SI], AL
4120A: 00 000 NULL           ADD [BX + SI], AL
4120B: 00 000 NULL           ADD [BX + SI], AL
4120C: 00 000 NULL           ADD [BX + SI], AL
4120D: 00 000 NULL           ADD [BX + SI], AL
4120E: 00 000 NULL           ADD [BX + SI], AL
4120F: 00 000 NULL           ADD [BX + SI], AL
41210: 00 000 NULL           ADD [BX + SI], AL
41211: 00 000 NULL           ADD [BX + SI], AL
41212: 00 000 NULL           ADD [BX + SI], AL
41213: 00 000 NULL           ADD [BX + SI], AL
41214: 00 000 NULL           ADD [BX + SI], AL
41215: 00 000 NULL           ...
```

```
extended value viewer
watch: MEM: 4120 : 0002
( ) word  (•) byte
                 L
hex:            01
bin:      00000001
oct:           001
decimal 8 bit
unsigned:        1
signed:          1
ascii:           ☺
```

Input: 8

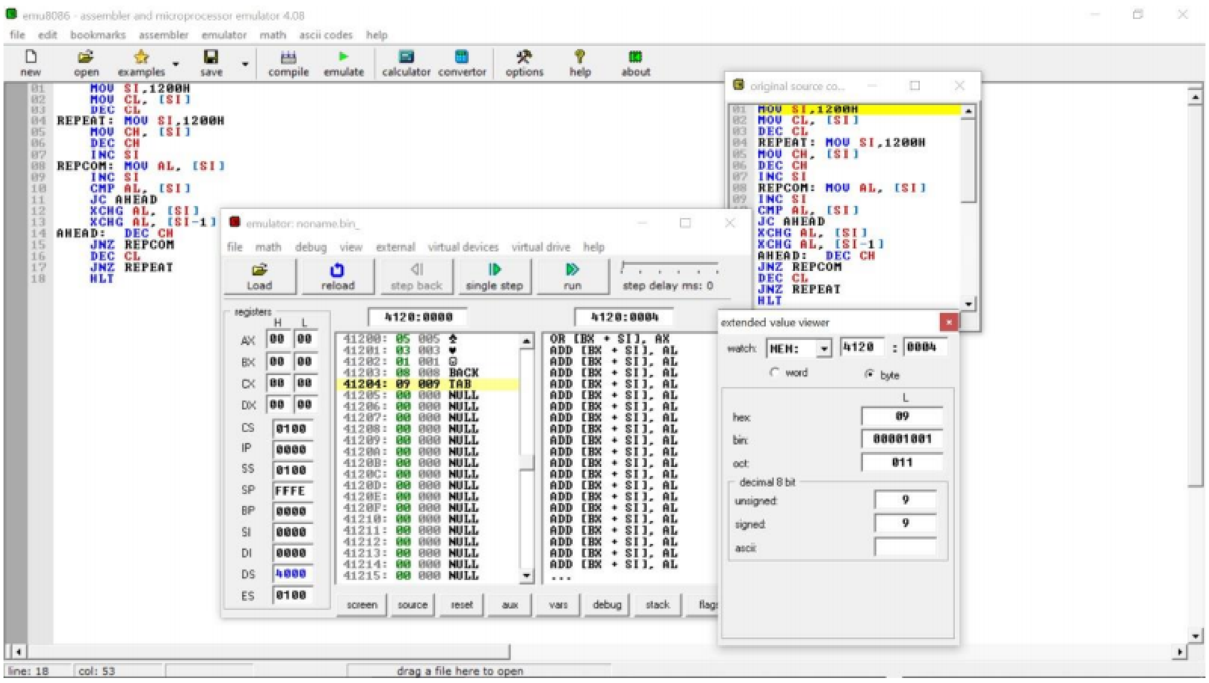


```
      MOV SI,1200H
      MOV CL, [SI]
      DEC CL
REPEAT: MOV SI,1200H
      MOV CH, [SI]
      DEC CH
      INC SI
REPCOM: MOV AL, [SI]
      INC SI
      CMP AL, [SI]
      JC AHEAD
      XCHG AL, [SI]
      XCHG AL, [SI-1]
AHEAD:  DEC CH
      JNZ REPCOM
      DEC CL
      JNZ REPEAT
      HLT
```

```
AX 00 00   BX 00 00   CX 00 00   DX 00 00
CS 0100  IP 0000  SS 0100  SP FFFE  BP 0000  SI 0000  DI 0000  DS 4000  ES 0100

4120:0000                     4120:0003
41200: 05 005 ♠               OR [BX + SI], AL
41201: 03 003 ♥               ADD [BX + SI], AL
41202: 01 001 ☺               ADD [BX + SI], AL
41203: 08 008 BACK            ADD [BX + SI], AL
41204: 00 000 NULL            ADD [BX + SI], AL
41205: 00 000 NULL            ADD [BX + SI], AL
41206: 00 000 NULL            ADD [BX + SI], AL
41207: 00 000 NULL            ADD [BX + SI], AL
41208: 00 000 NULL            ADD [BX + SI], AL
41209: 00 000 NULL            ADD [BX + SI], AL
4120A: 00 000 NULL            ADD [BX + SI], AL
4120B: 00 000 NULL            ADD [BX + SI], AL
4120C: 00 000 NULL            ADD [BX + SI], AL
4120D: 00 000 NULL            ADD [BX + SI], AL
4120E: 00 000 NULL            ADD [BX + SI], AL
4120F: 00 000 NULL            ADD [BX + SI], AL
41210: 00 000 NULL            ADD [BX + SI], AL
41211: 00 000 NULL            ADD [BX + SI], AL
41212: 00 000 NULL            ADD [BX + SI], AL
41213: 00 000 NULL            ADD [BX + SI], AL
41214: 00 000 NULL            ADD [BX + SI], AL
41215: 00 000 NULL            ...
```

extended value viewer — watch: MEM: 4120 : 0003, byte — hex: 08, bin: 00001000, oct: 010, decimal 8 bit unsigned: 8, signed: 8

Input: 9

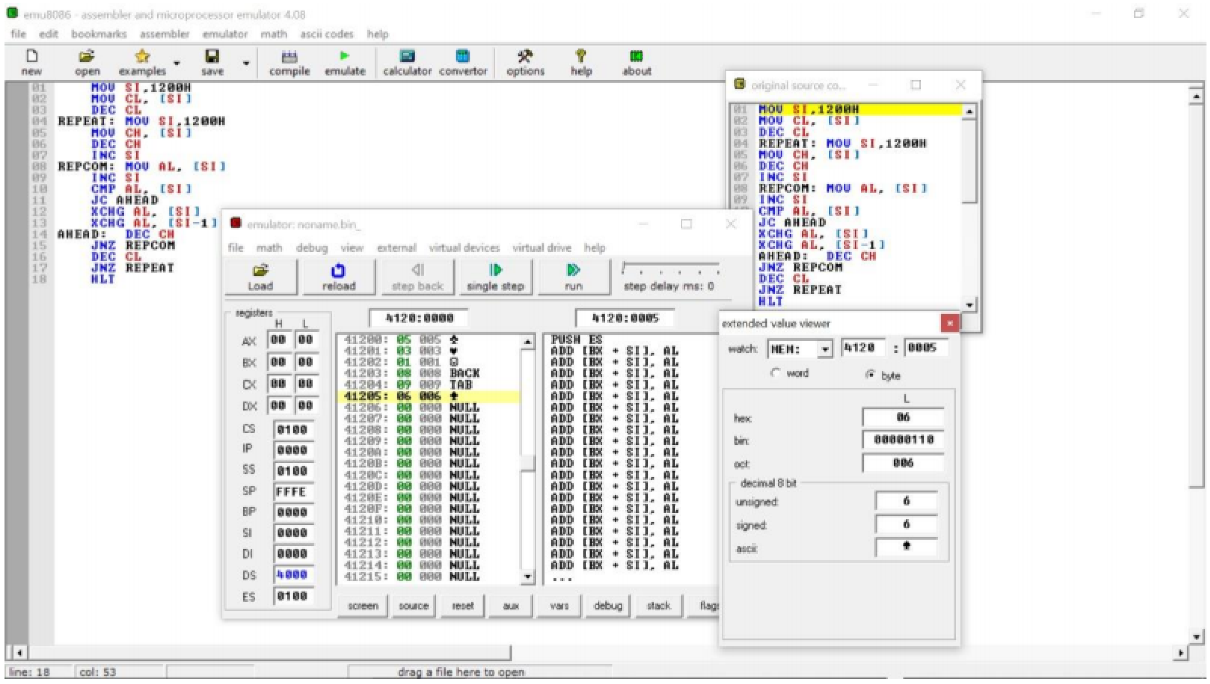


```
      MOV SI,1200H
      MOV CL, [SI]
      DEC CL
REPEAT: MOV SI,1200H
      MOV CH, [SI]
      DEC CH
      INC SI
REPCOM: MOV AL, [SI]
      INC SI
      CMP AL, [SI]
      JC AHEAD
      XCHG AL, [SI]
      XCHG AL, [SI-1]
AHEAD:  DEC CH
      JNZ REPCOM
      DEC CL
      JNZ REPEAT
      HLT
```

```
AX 00 00   BX 00 00   CX 00 00   DX 00 00
CS 0100  IP 0000  SS 0100  SP FFFE  BP 0000  SI 0000  DI 0000  DS 4000  ES 0100

4120:0000                     4120:0004
41200: 05 005 ♠               OR [BX + SI], AX
41201: 03 003 ♥               ADD [BX + SI], AL
41202: 01 001 ☺               ADD [BX + SI], AL
41203: 08 008 BACK            ADD [BX + SI], AL
41204: 09 009 TAB             ADD [BX + SI], AL
41205: 00 000 NULL            ADD [BX + SI], AL
41206: 00 000 NULL            ADD [BX + SI], AL
41207: 00 000 NULL            ADD [BX + SI], AL
41208: 00 000 NULL            ADD [BX + SI], AL
41209: 00 000 NULL            ADD [BX + SI], AL
4120A: 00 000 NULL            ADD [BX + SI], AL
4120B: 00 000 NULL            ADD [BX + SI], AL
4120C: 00 000 NULL            ADD [BX + SI], AL
4120D: 00 000 NULL            ADD [BX + SI], AL
4120E: 00 000 NULL            ADD [BX + SI], AL
4120F: 00 000 NULL            ADD [BX + SI], AL
41210: 00 000 NULL            ADD [BX + SI], AL
41211: 00 000 NULL            ADD [BX + SI], AL
41212: 00 000 NULL            ADD [BX + SI], AL
41213: 00 000 NULL            ADD [BX + SI], AL
41214: 00 000 NULL            ADD [BX + SI], AL
41215: 00 000 NULL            ...
```

extended value viewer — watch: MEM: 4120 : 0004, byte — hex: 09, bin: 00001001, oct: 011, decimal 8 bit unsigned: 9, signed: 9

Input: 6



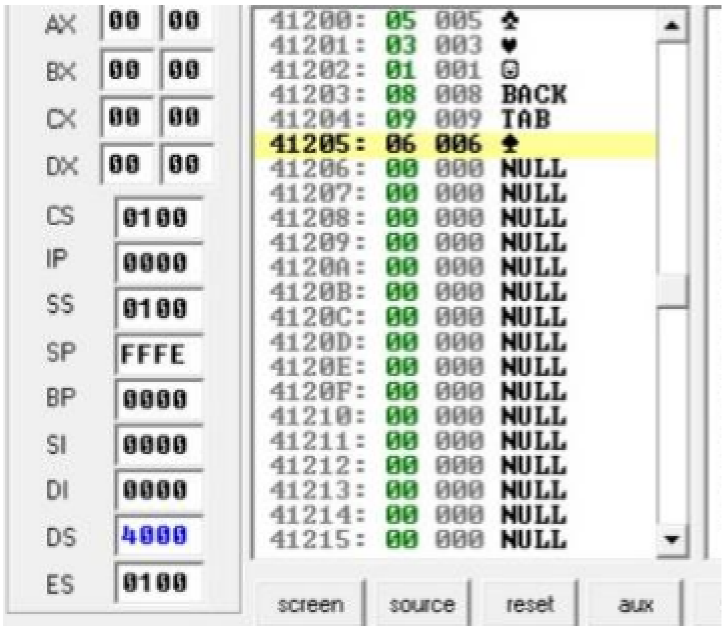
```
        MOV SI,1200H
        MOV CL, [SI]
        DEC CL
REPEAT: MOV SI,1200H
        MOV CH, [SI]
        DEC CH
        INC SI
REPCOM: MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC AHEAD
        XCHG AL, [SI]
        XCHG AL, [SI-1]
AHEAD:  DEC CH
        JNZ REPCOM
        DEC CL
        JNZ REPEAT
        HLT
```

```
4120:0000                     4120:0005
41200: 05 005 ♣               PUSH ES
41201: 03 003 ♥               ADD [BX + SI], AL
41202: 01 001 ☺               ADD [BX + SI], AL
41203: 08 008 BACK            ADD [BX + SI], AL
41204: 09 009 TAB             ADD [BX + SI], AL
41205: 06 006 ♠               ADD [BX + SI], AL
41206: 00 000 NULL            ADD [BX + SI], AL
41207: 00 000 NULL            ADD [BX + SI], AL
41208: 00 000 NULL            ADD [BX + SI], AL
41209: 00 000 NULL            ADD [BX + SI], AL
4120A: 00 000 NULL            ADD [BX + SI], AL
4120B: 00 000 NULL            ADD [BX + SI], AL
4120C: 00 000 NULL            ADD [BX + SI], AL
4120D: 00 000 NULL            ADD [BX + SI], AL
4120E: 00 000 NULL            ADD [BX + SI], AL
4120F: 00 000 NULL            ADD [BX + SI], AL
41210: 00 000 NULL            ADD [BX + SI], AL
41211: 00 000 NULL            ADD [BX + SI], AL
41212: 00 000 NULL            ADD [BX + SI], AL
41213: 00 000 NULL            ADD [BX + SI], AL
41214: 00 000 NULL            ADD [BX + SI], AL
41215: 00 000 NULL            ...
```

```
registers
    H  L
AX  00 00
BX  00 00
CX  00 00
DX  00 00
CS  0100
IP  0000
SS  0100
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  4000
ES  0100
```

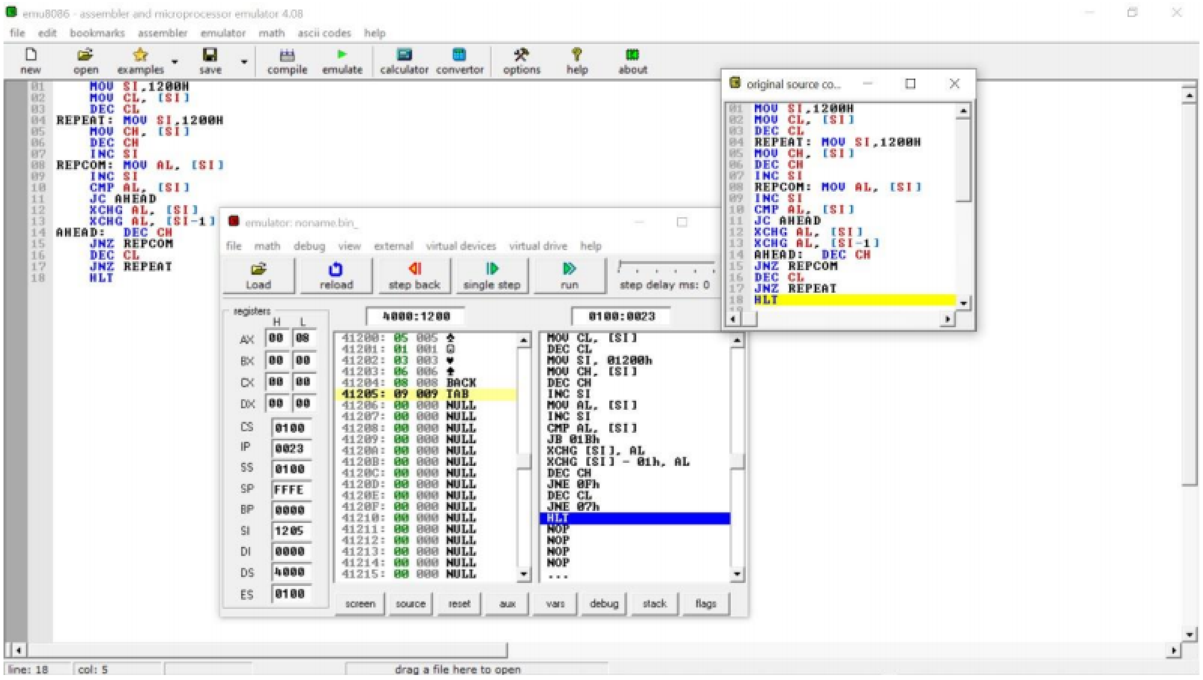
```
extended value viewer
watch: MEM: 4120 : 0005
( ) word  (•) byte
                L
hex:            06
bin:            00000110
oct:            006
decimal 8 bit
unsigned:       6
signed:         6
ascii:          ♠
```

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
original source co...
emulator: noname.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
line: 18 col: 53 drag a file here to open



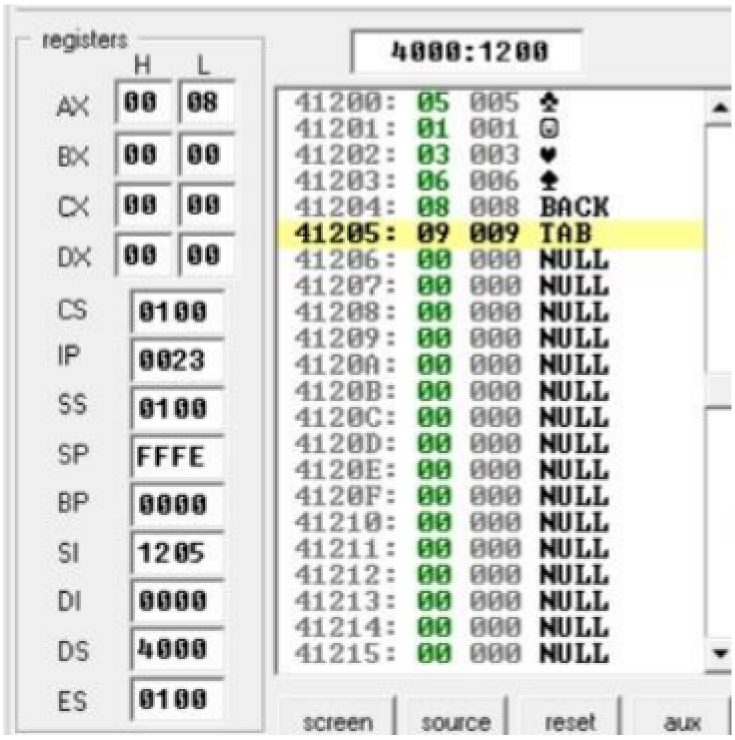
```
AX  00 00     41200: 05 005 ♣
BX  00 00     41201: 03 003 ♥
CX  00 00     41202: 01 001 ☺
DX  00 00     41203: 08 008 BACK
              41204: 09 009 TAB
              41205: 06 006 ♠
CS  0100      41206: 00 000 NULL
IP  0000      41207: 00 000 NULL
SS  0100      41208: 00 000 NULL
SP  FFFE      41209: 00 000 NULL
BP  0000      4120A: 00 000 NULL
SI  0000      4120B: 00 000 NULL
DI  0000      4120C: 00 000 NULL
DS  4000      4120D: 00 000 NULL
ES  0100      4120E: 00 000 NULL
              4120F: 00 000 NULL
              41210: 00 000 NULL
              41211: 00 000 NULL
              41212: 00 000 NULL
              41213: 00 000 NULL
              41214: 00 000 NULL
              41215: 00 000 NULL
```

OUTPUT:



```
        MOV SI,1200H
        MOV CL, [SI]
        DEC CL
REPEAT: MOV SI,1200H
        MOV CH, [SI]
        DEC CH
        INC SI
REPCOM: MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC AHEAD
        XCHG AL, [SI]
        XCHG AL, [SI-1]
AHEAD:  DEC CH
        JNZ REPCOM
        DEC CL
        JNZ REPEAT
        HLT
```

```
4000:1200                         0100:0023
AX 00 08   41200: 05 005 ♣        MOV CL, [SI]
BX 00 00   41201: 01 001 ☺        DEC CL
CX 00 00   41202: 03 003 ♥        MOV SI, 01200h
DX 00 00   41203: 06 006 ♠        MOV CH, [SI]
CS 0100    41204: 08 008 BACK     DEC CH
IP 0023    41205: 09 009 TAB      INC SI
SS 0100    41206: 00 000 NULL     MOV AL, [SI]
SP FFFE    41207: 00 000 NULL     INC SI
BP 0000    41208: 00 000 NULL     CMP AL, [SI]
SI 1205    41209: 00 000 NULL     JB 01Bh
DI 0000    4120A: 00 000 NULL     XCHG [SI], AL
DS 4000    4120B: 00 000 NULL     XCHG [SI] - 01h, AL
ES 0100    4120C: 00 000 NULL     DEC CH
           4120D: 00 000 NULL     JNE 0Fh
           4120E: 00 000 NULL     DEC CL
           4120F: 00 000 NULL     JNE 07h
           41210: 00 000 NULL     HLT
           41211: 00 000 NULL     NOP
           41212: 00 000 NULL     NOP
           41213: 00 000 NULL     NOP
           41214: 00 000 NULL     NOP
           41215: 00 000 NULL     ...
```



```
registers          4000:1200
    H  L
AX  00 08   41200: 05 005 ♣
BX  00 00   41201: 01 001 ☺
CX  00 00   41202: 03 003 ♥
DX  00 00   41203: 06 006 ♠
CS  0100    41204: 08 008 BACK
IP  0023    41205: 09 009 TAB
SS  0100    41206: 00 000 NULL
SP  FFFE    41207: 00 000 NULL
BP  0000    41208: 00 000 NULL
SI  1205    41209: 00 000 NULL
DI  0000    4120A: 00 000 NULL
DS  4000    4120B: 00 000 NULL
ES  0100    4120C: 00 000 NULL
            4120D: 00 000 NULL
            4120E: 00 000 NULL
            4120F: 00 000 NULL
            41210: 00 000 NULL
            41211: 00 000 NULL
            41212: 00 000 NULL
            41213: 00 000 NULL
            41214: 00 000 NULL
            41215: 00 000 NULL
```

SECTION 2

Study how to interface a keyboard with 8086 Processor along with its hardware and software details. Make a technical report of the same and along with your comments and observations.

When you press a key on your computer, you are activating a switch. There are many different ways of making these switches. An overview of the construction and operation of some of the most common types.

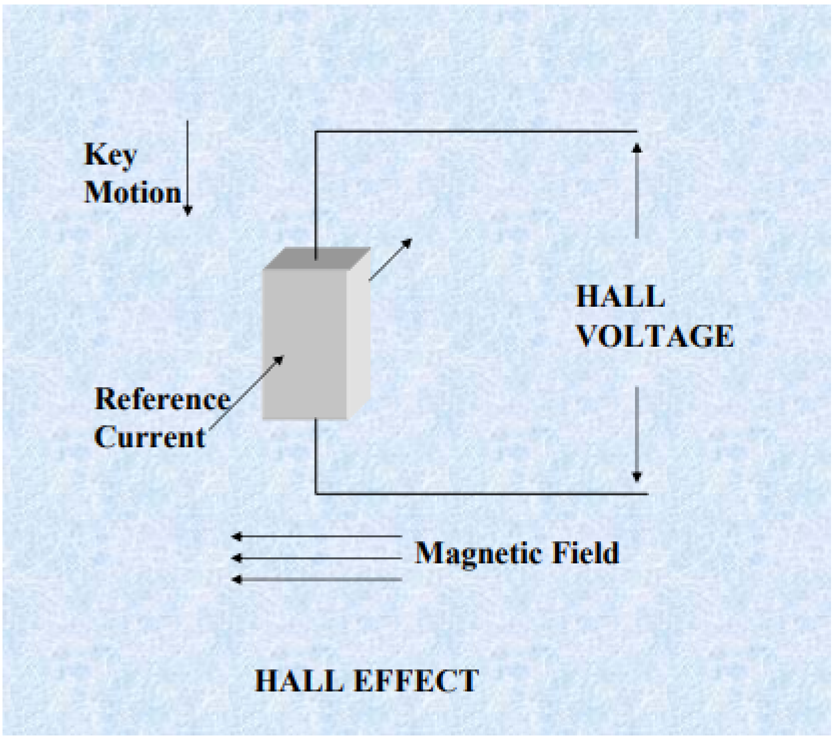
1. Mechanical key switches: In mechanical-switch keys, two pieces of metal are pushed together when you press the key. The actual switch elements are often made of a phosphor-bronze alloy with gold plating on the contact areas. The key switch usually contains a spring to return the key to the non pressed position and perhaps a small piece of foam to help damp out bouncing.
Some mechanical key switches now consist of a molded silicone dome with a small piece of conductive rubber foam short two trace on the printed-circuit board to produce the key pressed signal.
• Mechanical switches are relatively inexpensive but they have several disadvantages. First, they suffer from contact bounce. A pressed key may make and break contact several times before it makes solid contact.
• Second, the contacts may become oxidized or dirty with age so they no longer make a dependable connection
• Higher-quality mechanical switches typically have a rated lifetime of about 1 million keystrokes. The silicone dome type typically lasts 25 million keystrokes.

2. Membrane key switches: These switches are really a special type of mechanical switches. They consist of a three-layer plastic or rubber sandwich.
• The top layer has a conductive line of silver ink running under each key position. The bottom layer has a conductive line of silver ink running under each column of keys.
• When you press a key, you push the top ink line through the hole to contact the bottom ink line.
• The advantages of membrane keyboards is that they can be made as very thin, sealed units.
• They are often used on cash registers in fast food restaurants. The lifetime of membrane keyboards varies over a wide range.

3. Capacitive key switches: A capacitive key switch has two small metal plates on the printed circuit board and another metal plate on the bottom of a piece of foam.
• When you press the key, the movable plate is pushed closer to the fixed plate. This changes the capacitance between the fixed plates. Sense amplifier circuitry detects this change in capacitance and produces a logic level signal that indicates a key has been pressed.
• The big advantages of a capacitive switch is that it has no mechanical contacts to become oxidized or dirty.
• A small disadvantage is the specific circuitry needed to detect the change in capacitance. • Capacitive key switches typically have a rated lifetime of about 20 million keystrokes.

4. Hall effect keyswitches: This is another type of switch which has no mechanical contact. It takes advantage of the deflection of a moving charge by a magnetic field.
• A reference current is passed through a semiconductor crystal between two opposing faces. When a key is pressed, the crystal is moved through a magnetic field which has its flux lines perpendicular to the direction of current flow in the crystal.

• Moving the crystal through the magnetic field causes a small voltage to be developed between two of the other opposing faces of the crystal.
• This voltage is amplified and used to indicate that a key has been pressed. Hall effect sensors are also used to detect motion in many electrically controlled machines.
• Hall effect keyboards are more expensive because of the more complex switch mechanism, but they are very dependable and have typically rated lifetime of 100 million or more keystrokes.



In most keyboards, the key switches are connected in a matrix of rows and columns, as shown in fig.
• We will use simple mechanical switches for our examples, but the principle is the same for other type of switches.
• Getting meaningful data from a keyboard, it requires the following three major tasks:

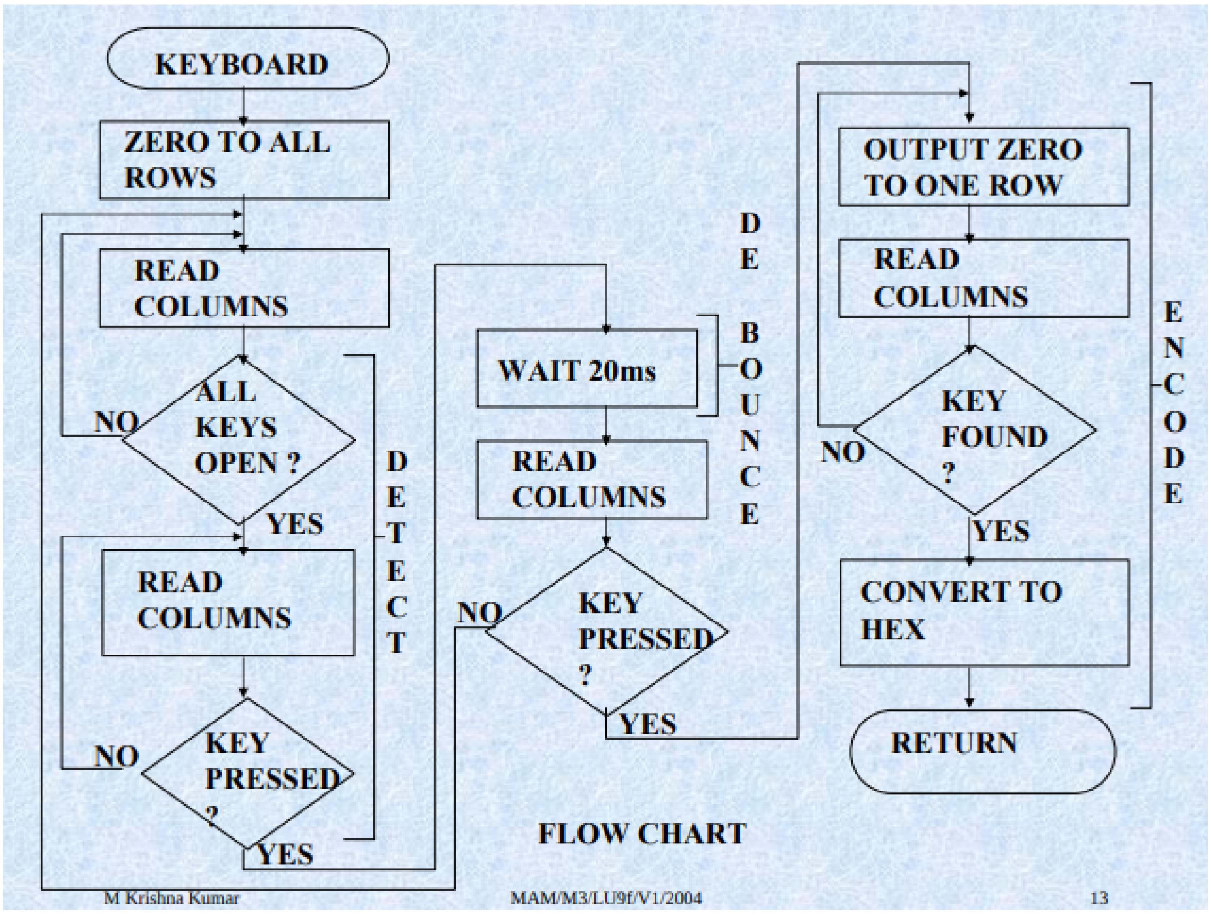
1. Detecting a keypress.
2. Debounce the keypress.
3. Encode the keypress

• Three tasks can be done with hardware, software, or a combination of two, depending on the application.

1. Software Keyboard Interfacing:

Circuit connection and algorithm : The following fig (a) shows how a hexadecimal keypad can be connected to a couple of microcomputer ports so the three interfacing tasks can be done as part of a program. • The rows of the matrix are connected to four output port lines. The column lines of the matrix are connected to four input-port lines. To make the program simpler, the row lines are also connected to four input lines.

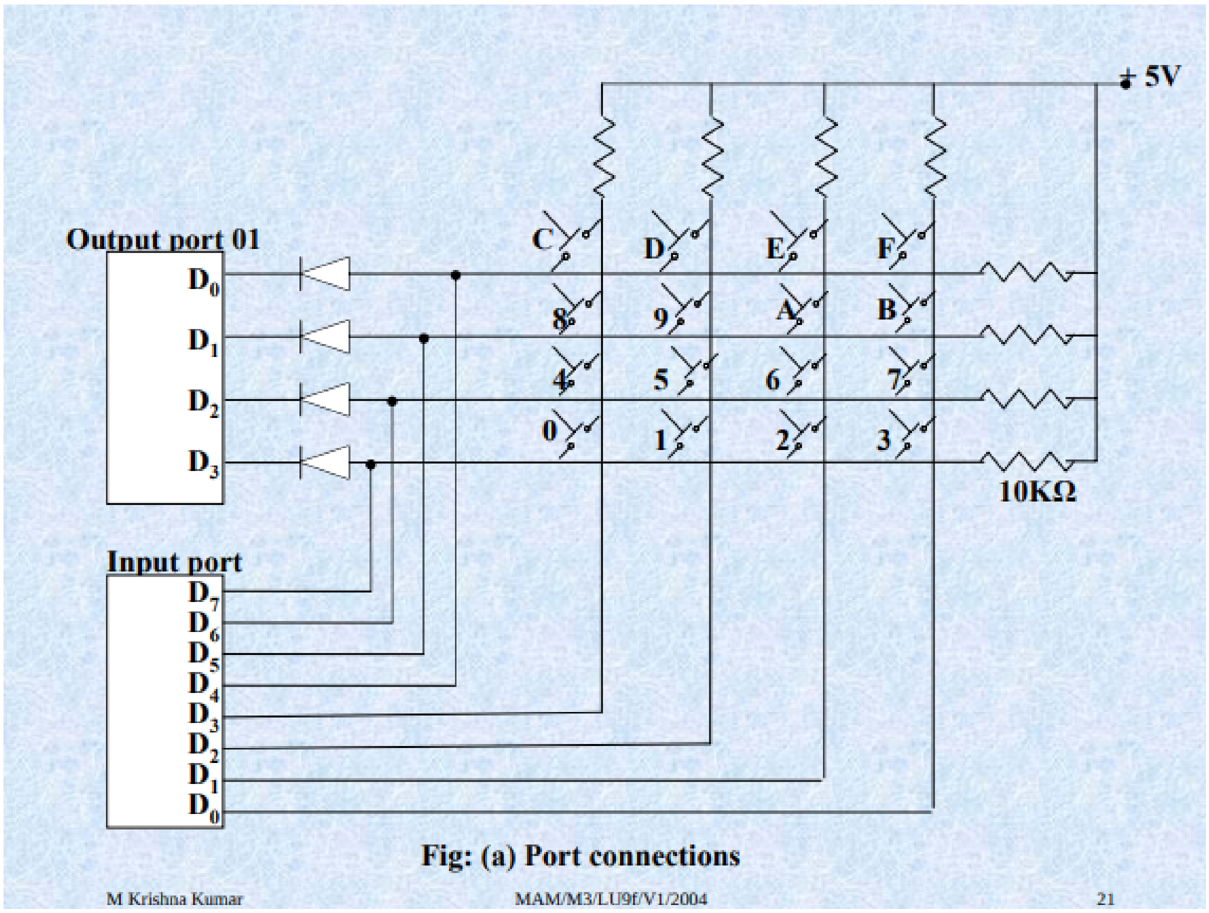
• When no keys are pressed, the column lines are held high by the pull-up resistor connected to +5V. Pressing a key connects a row to a column. If a low is output on a row and a key in that row is pressed, then the low will appear on the column which contains that key and can be detected on the input port.
• If you know the row and column of the pressed key, you then know which key was pressed, and you can convert this information into any code you want to represent that key.
• The following flow chart for a procedure to detect, debounce and produce the hex code for a pressed key.
• An easy way to detect if any key in the matrix is pressed is to output 0's to all rows and then check the column to see if a pressed key has connected a low to a column.
• In the algorithm we first output lows to all the rows and check the columns over and over until the columns are all high. This is done before the previous key has been released before looking for the next one. In the standard keyboard terminology, this is called two-key lockout.



• Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns, indicating that a key has been pressed. This second loop does the detect task for us. A simple 20- ms delay procedure then does the debounce task.
• After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed and the initial detection was caused by a noise pulse or a light brushing past a key. If any of the columns are still low, then the assumption is made that it was a valid keypress.

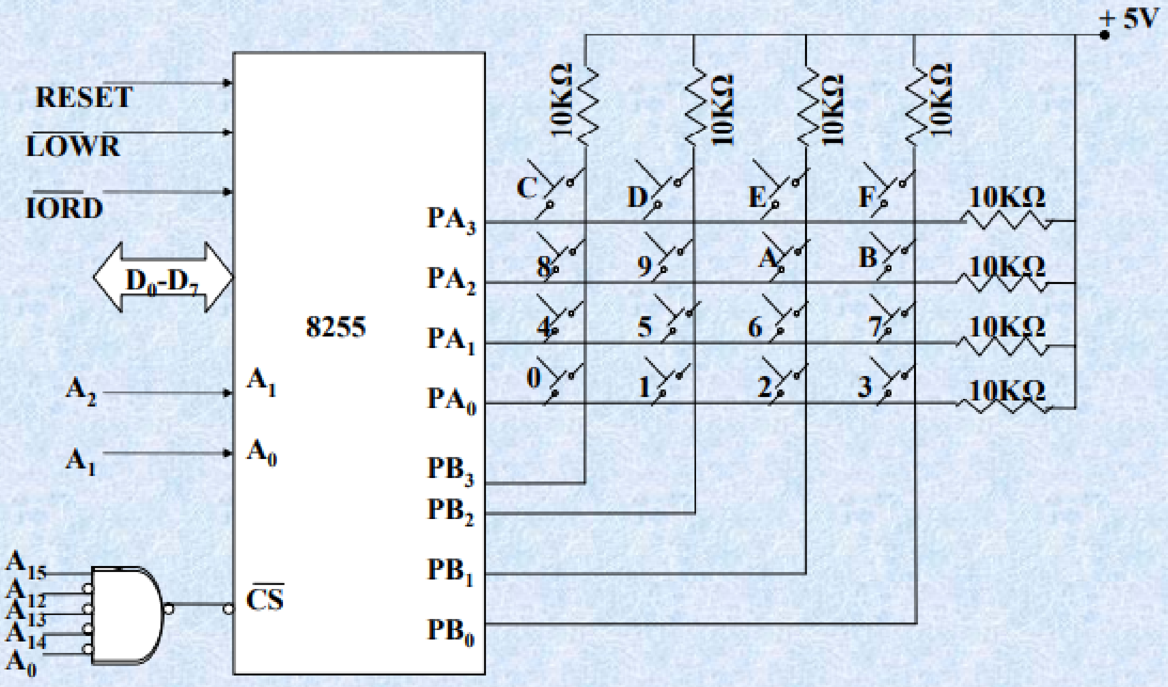
• The final task is to determine the row and column of the pressed key and convert this row and column information to the hex code for the pressed key. To get the row and column information, a low is output to one row and the columns are read. If none of the columns is low, the pressed key is not in that row. So the row is rotated to the next row and the columns are checked again. The process is repeated until a low on a row produces a low on one of the columns. • The pressed key then is in the row which is low at that time.
• The connection fig shows the byte read in from the input port will contain a 4-bit code which represents the row of the pressed key and a 4-bit code which represents the column of the pressed key.
• **Error trapping:** The concept of detecting some error condition such as “ no match found” is called error trapping. Error trapping is a very important part of real programs. Even in simple programs, think what might happen with no error trap if two keys in the same row were pressed at exactly the same time and a column code with two lows in it was produced.
• This code would not match any of the row-column codes in the table, so after all the values in the table were checked, the assigned register in the program would be decremented from 0000H to FFFFH. The compare decrement cycle would continue through 65,536 memory locations until, by changing the value in a memory location matched the row-column code. The contents of the lower byte register at what point would be passed back to the calling routine. The changes are 1 in 256 that would be the correct value for one of the pressed keys. You should keep an error trap in a program whenever there is a chance for it.

2. Keyboard Interfacing with Hardware: For the system where the CPU is too busy to be bothered doing these tasks in software, an external device is used to do them. • One of the MOS devices which can do this is the General Instruments AY5-2376 which can be connected to the rows and columns of a keyboard switch matrix. • The AY5-2376 independently detects a keypress by cycling a low down through the rows and checking the columns. When it finds a key pressed, it waits for a debounce time.
• If the key is still pressed after the debounce time, the AY5- 2376 produces the 8-bit code for the pressed key and sends it out to the microcomputer port on 8 parallel lines. The microcomputer knows that a valid ASCII code is on the data lines, the AY5-2376 outputs a strobe pulse.
• The microcomputer can detect this strobe pulse and read in ASCII code on a polled basis or it can detect the strobe pulse on an interrupt basis.
• With the interrupt method the microcomputer doesn’t have to pay any attention to the keyboard until it receives an interrupt signal.
• So this method uses very little of the microcomputer time. The AY5-2376 has a feature called two-key rollover. This means that if two keys are pressed at nearly the same time, each key will be detected, debounced and converted to ASCII.
• The ASCII code for the first key and a strobe signal for it will be sent out then the ASCII code for the second key and a strobe signal for it will be sent out and compare this with two-key lockout.



Example :
Interface a 4 * 4 keyboard with 8086 using 8255 and write an ALP for detecting a key closure and return the key code in AL. The debounce period for a key is 10ms. Use software debouncing technique. DEBOUNCE is an available 10ms delay routine.

Solution:
Port A is used as an output port for selecting a row of keys while Port B is used as an input port for sensing a closed key. Thus the keyboard lines are selected one by one through port A and the port B lines are polled continuously till a key closure is sensed.
The routine DEBOUNCE is called for key debouncing. The key code is depending upon the selected row and a low sensed column.

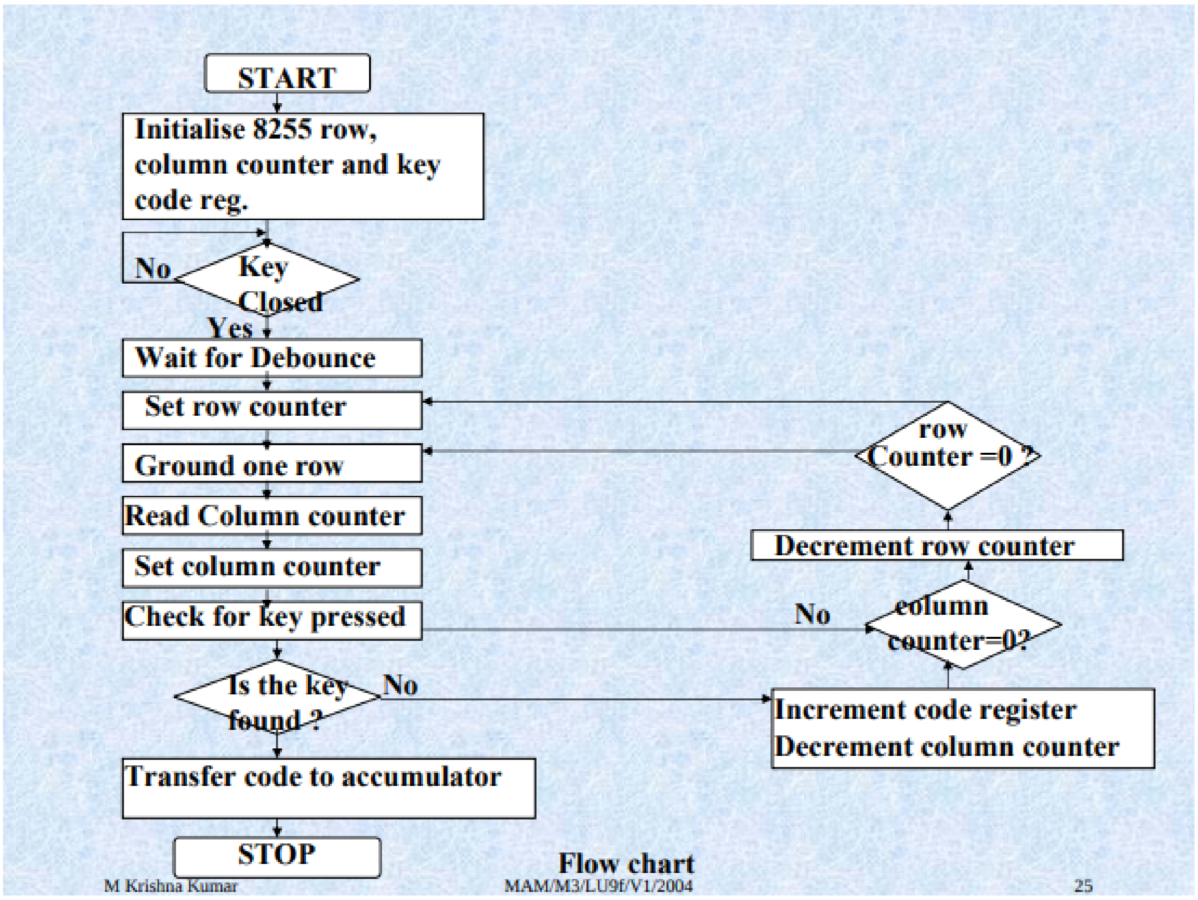


Interfacing 4 * 4 Keyboard

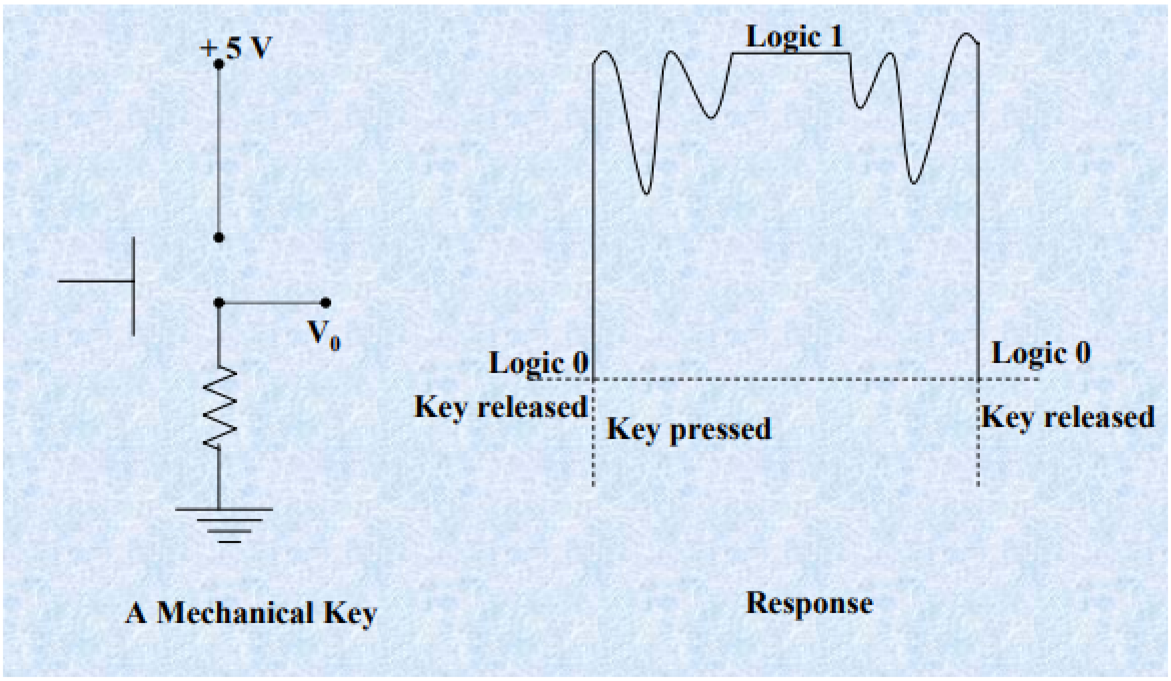
M Krishna Kumar MAM/M3/LU9f/V1/2004 23

• The higher order lines of port A and port B are left unused. The address of port A and port B will respectively be 8000H and 8002H while the address of CWR will be 8006H. The flow chart of the complete program is as given. The control word for this problem will be 82H. Code segment CS is used for storing the program code.

• **Key Debounce :** Whenever a mechanical push-button is pressed or released once, the mechanical components of the key do not change the position smoothly, rather it generates a transient response .



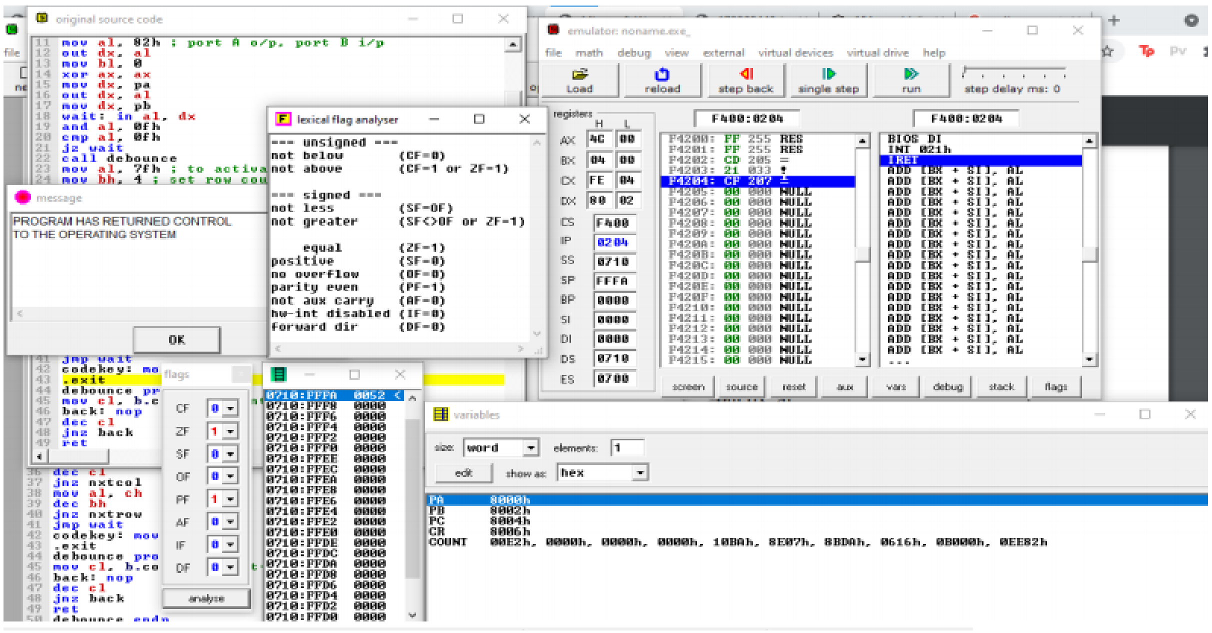
• These transient variations may be interpreted as the multiple key pressure and responded accordingly by the microprocessor system.
• To avoid this problem, two schemes are suggested: the first one utilizes a bistable multivibrator at the output of the key to debounce .
• The other scheme suggests that the microprocessor should be made to wait for the transient period (usually 10ms), so that the transient response settles down and reaches a steady state.
• A logic '0' will be read by the microprocessor when the key is pressed. • In a number of high precision applications, a designer may have two options- the first is to have more than one 8-bit port, read (write) the port one by one and then from the multibyte data, the second option allows forming 16-bit ports using two 8-bit ports and use 16-bit read or write operations.



CODE:

```
.model small
.data
pa dw 8000h
pb dw 8002h
pc dw 8004h
cr dw 8006h
count dw 0e2h
.code
.startup
mov dx, cr
mov al, 82h ; port A o/p, port B i/p
out dx, al
mov bl, 0
xor ax, ax
mov dx, pa
out dx, al
mov dx, pb
wait:
 in al, dx
 and al, 0fh
 cmp al, 0fh
 jz wait
 call debounce
 mov al, 7fh ; to activate a row, send 0
 mov bh, 4 ; set row counter
nxtrow:
 rol al, 1
 mov ch, al
 mov dx, pa
 out dx, al
 mov dx, pb
in al, dx
 and al, 0fh
 mov cl, 4 ; set column counter
nxtcol:
 ror al, 1
 jnc codekey ; key closure is found when CF=0
 inc bl ; increment bl for next binary key code
 dec cl
 jnz nxtcol
 mov al, ch
 dec bh
 jnz nxtrow
 jmp wait
codekey:
 mov al, bl
 .exit
debounce proc near
mov cl, b.count
back: nop
dec cl
jnz back
ret
debounce endp
end
```

OUTPUT:



```
mov al, 82h ; port A o/p, port B i/p
out dx, al
mov bl, 0
xor ax, ax
mov dx, pa
out dx, al
mov dx, pb
wait: in al, dx
and al, 0fh
cmp al, 0fh
jz wait
call debounce
mov al, 7fh ; to activa
mov bh, 4 ; set row cou
```

```
PROGRAM HAS RETURNED CONTROL
TO THE OPERATING SYSTEM
```

```
=== unsigned ===
not below        (CF=0)
not above        (CF=1 or ZF=1)

=== signed ===
not less         (SF=OF)
not greater      (SF<>OF or ZF=1)

    equal        (ZF=1)
positive         (SF=0)
no overflow      (OF=0)
parity even      (PF=1)
not aux carry    (AF=0)
hw-int disabled  (IF=0)
forward dir      (DF=0)
```

```
AX  4C  00
BX  04  00
CX  FE  04
DX  80  02
CS  F400
IP  0204
SS  0710
SP  FFFA
BP  0000
SI  0000
DI  0000
DS  0710
ES  0700
```

```
F400:0204
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ±
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
F4211: 00 000 NULL
F4212: 00 000 NULL
F4213: 00 000 NULL
F4214: 00 000 NULL
F4215: 00 000 NULL
```

```
F400:0204
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

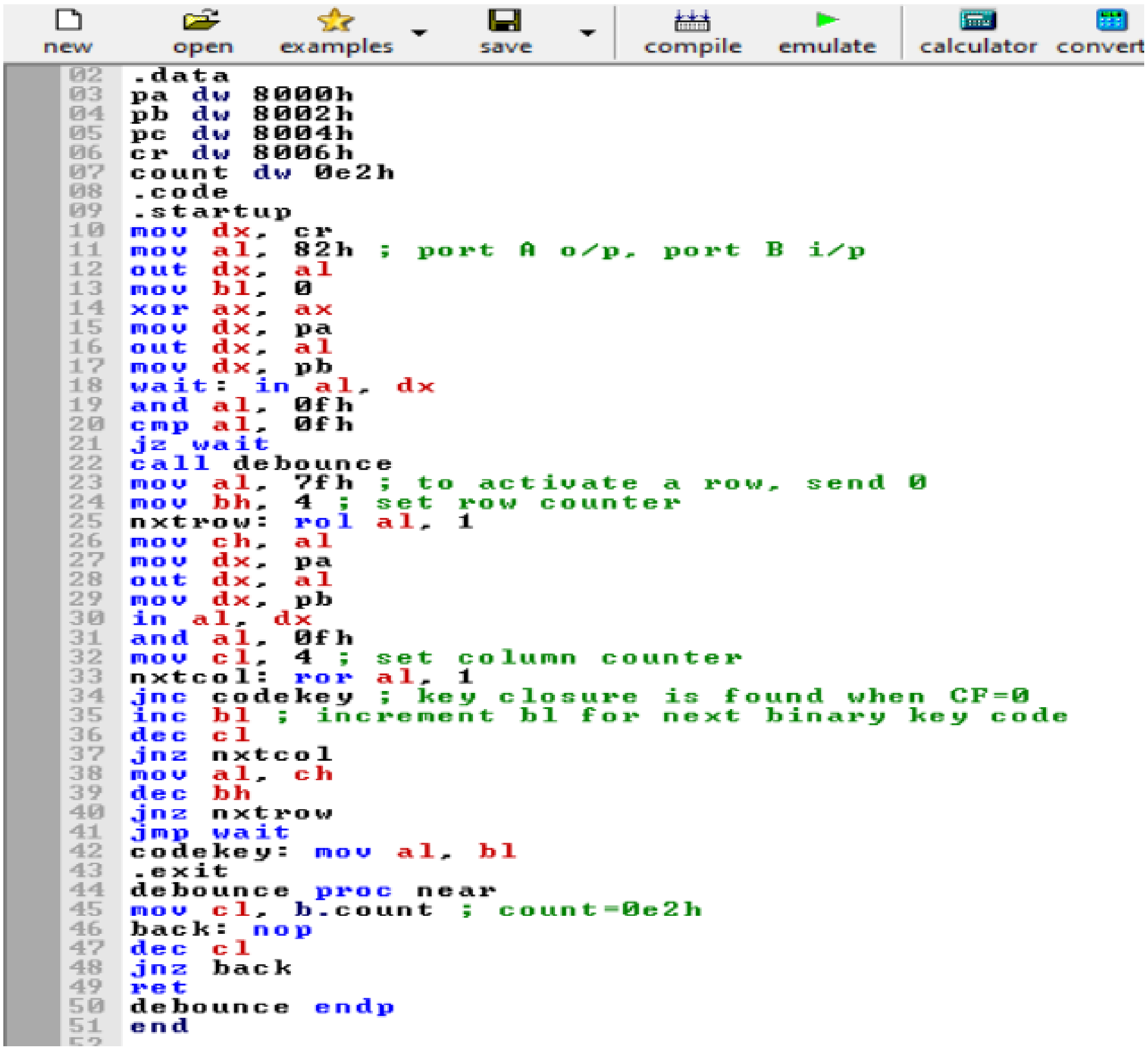
```
jmp wait
codekey: mo
.exit
debounce pr
mov cl, b.c
back: nop
dec cl
jnz back
ret
```

```
CF 0
ZF 1
SF 0
OF 0
PF 1
AF 0
IF 0
DF 0
```

```
dec cl
jnz nxtcol
mov al, ch
dec bh
jnz nxtrow
jmp wait
codekey: mov
.exit
debounce pro
mov cl, b.co
back: nop
dec cl
jnz back
ret
debounce endp
```

```
0710:FFFA  0052 <
0710:FFF8  0000
0710:FFF6  0000
0710:FFF4  0000
0710:FFF2  0000
0710:FFF0  0000
0710:FFEE  0000
0710:FFEC  0000
0710:FFEA  0000
0710:FFE8  0000
0710:FFE6  0000
0710:FFE4  0000
0710:FFE2  0000
0710:FFE0  0000
0710:FFDE  0000
0710:FFDC  0000
0710:FFDA  0000
0710:FFD8  0000
0710:FFD6  0000
0710:FFD4  0000
0710:FFD2  0000
0710:FFD0  0000
```

```
size: word    elements: 1
show as: hex

PA      0000h
PB      0002h
PC      0004h
CR      0006h
COUNT   00E2h, 0000h, 0000h, 0000h, 10BAh, 8E07h, 8BDAh, 0616h, 0B000h, 0EE82h
```



```
.data
pa dw 8000h
pb dw 8002h
pc dw 8004h
cr dw 8006h
count dw 0e2h
.code
.startup
mov dx, cr
mov al, 82h ; port A o/p, port B i/p
out dx, al
mov bl, 0
xor ax, ax
mov dx, pa
out dx, al
mov dx, pb
wait: in al, dx
and al, 0fh
cmp al, 0fh
jz wait
call debounce
mov al, 7fh ; to activate a row, send 0
mov bh, 4 ; set row counter
nxtrow: rol al, 1
mov ch, al
mov dx, pa
out dx, al
mov dx, pb
in al, dx
and al, 0fh
mov cl, 4 ; set column counter
nxtcol: ror al, 1
jnc codekey ; key closure is found when CF=0
inc bl ; increment bl for next binary key code
dec cl
jnz nxtcol
mov al, ch
dec bh
jnz nxtrow
jmp wait
codekey: mov al, bl
.exit
debounce proc near
mov cl, b.count ; count=0e2h
back: nop
dec cl
jnz back
ret
debounce endp
end
```