Group No: 63
Problem No.: 6

Microprocessors Programming and Interfacing
Design Assignment

DIGITAL IC TESTER

BHAVESH KUMAR TEKWANI	2017A3PS0338P
ADITYA RAMACHANDRAN	2017A3PS0339P
SHRAMAY PALTA	2017A3PS0340P
SOHAM AGARWAL	2017A3PS0345P

An assignment submitted in
Partial fulfilment of the requirement of the course
EEE F241: MICROPROCESSOR PROGRAMMING AND INTERFACING

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, RAJASTHAN-333031

25 APRIL, 2019

Problem Statement:

Design a Microprocessor based tester to test the logical functioning of the following chips:

- 7408
- 7486
- 7432

The IC to be tested will be inserted in a 14 pin ZIF socket. The IC number is to be entered via a keyboard. The results of the test along with the IC number are to be displayed on LCD as"74xy PASS" or "74xy FAIL". Design the necessary hardware and write the necessary ALP for implementing the above-mentioned task.

Assumptions:

The following are the assumptions made regarding the system:

- The 8086 Chip is already programmed with the specified code from an external source.
- There is a jump instruction at the address that the processor generates to the place where our code is stored (i.e.to FF000H).
- The user doesn't put the chips in the wrong sockets and he presses the appropriate button.
- User enters a 4-digit number as the IC number.
- The user puts the IC in the ZIF socket before entering the IC number.
- GND Pins of the 8086 are grounded and Vcc & MIN/MAX pins are connected to +5V.

System Description:

The digital IC tester is implemented by using the 8086 microprocessor. The processing of the inputs and outputs is done by the microprocessor. The display part on the microprocessor is modeled using LCD. After the successful testing of the IC, the result is displayed on the LCD.

The basic function of the digital IC tester is to test a digital IC for correct logical functioning as described in the truth table and/or function table. It can test digital ICs having a maximum of 14 pins. Since it is programmable, any number of ICs can be tested within the constraint of the memory available. This model applies the necessary signals to the inputs of the IC, monitoring the outputs at each stage and comparing them with the outputs in the truth table. Any discrepancy in the functioning of the IC results in a fail indication, displays the faulty and good gates on the LCD. The testing procedure is accomplished with the help of keys present on the main.

At this stage we had completed to test the most common used digital IC's used in our laboratories, which are 7408, 7486 and 7432 and successfully completed writing assembly code for these IC’s.

- ✓ The system is based on an 8086 microprocessor using an 8255 for Input / Output interfacing.
- ✓ A total of 4k ROM and 2k RAM memory are interfaced to 8086 microprocessor.
- ✓ The memory chips are interfaced by memory mapped interfacing while all other devices are interfaced to the microprocessor by I/0 mapped interfacing.
- ✓ 16 push buttons are used to form a 4x4 matrix keypad with keys 0,1,2,3,4,5,6,7,8,9,BACKSPACE,RESET.
- ✓ Output is in the form of text messages displayed on the LCD.
- ✓ For the simulation part instead of ZIF socket, one of the three ICs is connected to test the circuit (Proteus does not have 14-pin ZIF socket).

Hardware Description:

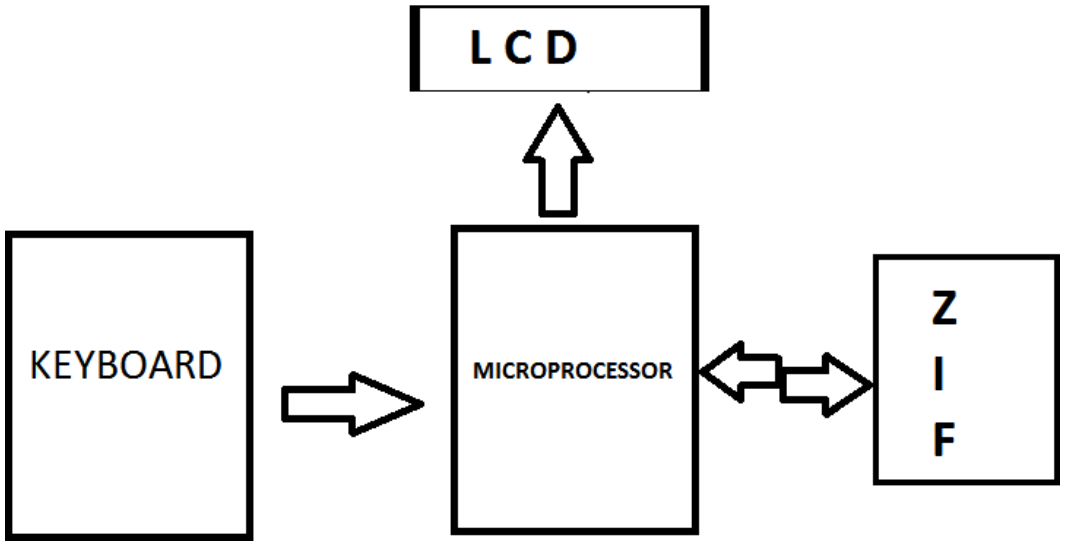
The following is the list of components used in the system:

Chip Number	Quantity	Chip	Purpose
8086	1	Microprocessor	Central Processing Unit
6132	2	RAM	Read Write Memory to house Data segment and Stack segment
2732	2	EPROM	Read Only Erasable Programmable memory to house the code

8255	2	Programmable Peripheral Interface	
74LS245	2	8 bit buffer Bidirectional buffers	
74LS373	3	8bitlatches	
74154	1	Decoder	
LM020L	1	LCD	Display
74LS138	1	Decoder	

Apart from the above 16 push buttons for keyboard matrix, one SPDF for 8086 reset switch, and logic gates are used.

BLOCK DIAGRAM OF IC TESTER



Memory Mapping:

STARTING ADDRESS-00000H
Number of 2732 chips-2
ROM-4K
We divide 4K ROM in-
2K even bank and 2K odd bank.
Number of 6116 chips-2
RAM-2K
We divide 2K RAM in –
1K even bank and 1K odd bank
ROM1E-00000H, 00002H,……, 00FFEH
ROM1O-00001H, 00003H,……,00FFFH
RAM1E-01000H, 01002H,……,017FEH
RAM1O-01001H, 01003H,……,017FFH

I/O organization:

Two 8255 (Programmable Peripheral Interface) are used to communicate with other input and output devices. It is organized in the following manner.

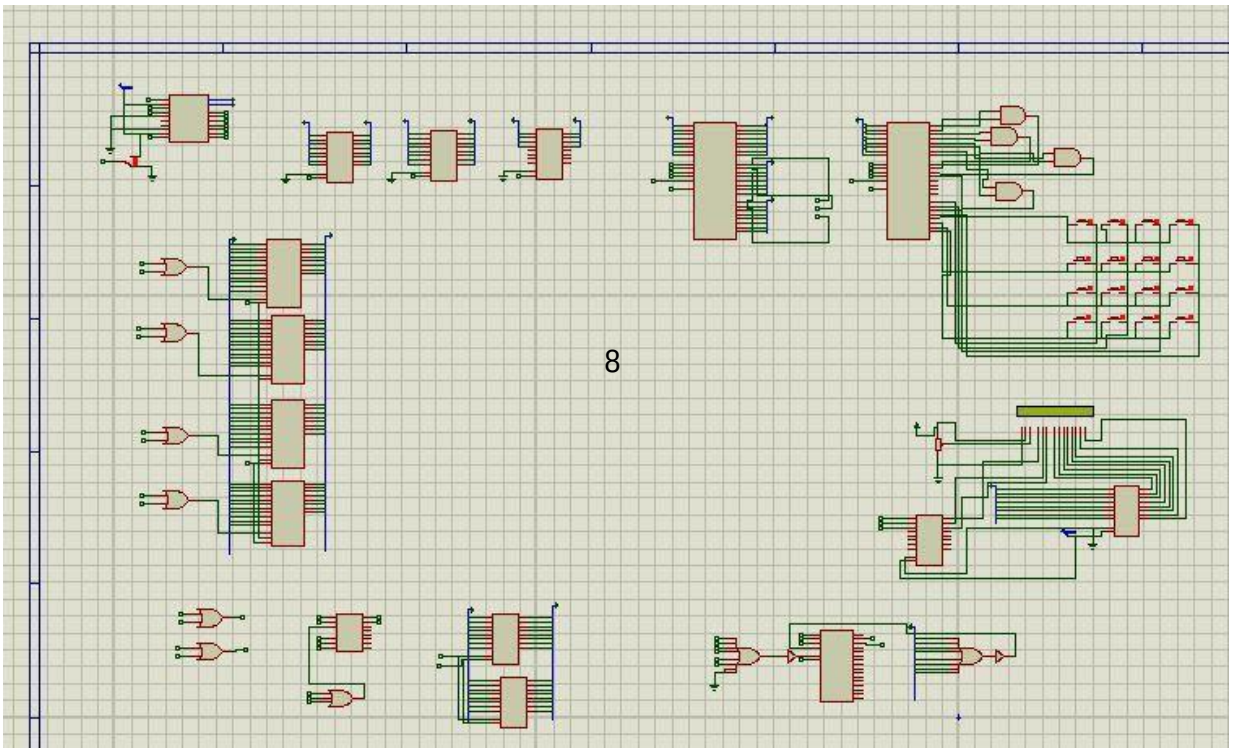
8255(1):

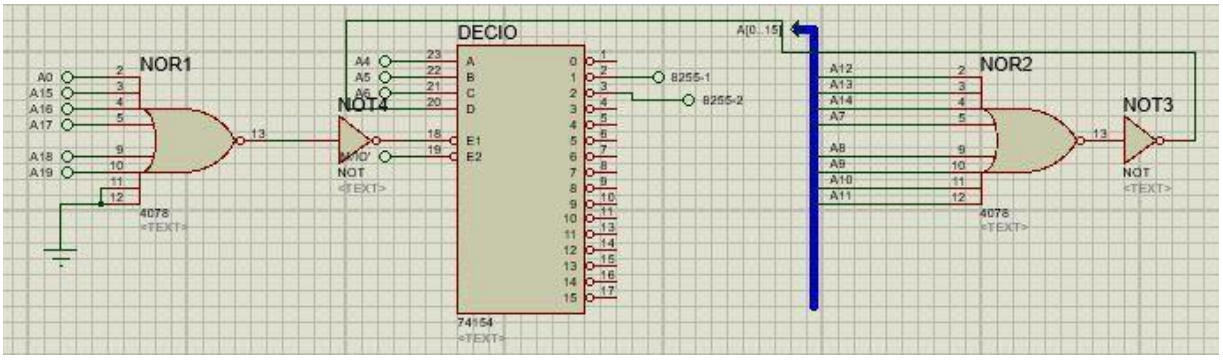
Port	Port Address	Mode	Input / Output
A	10h	0	Output
B	12h	0	Output
C lower	14h	0	Empty
C upper	14h	0	Empty
Control Register	16h		

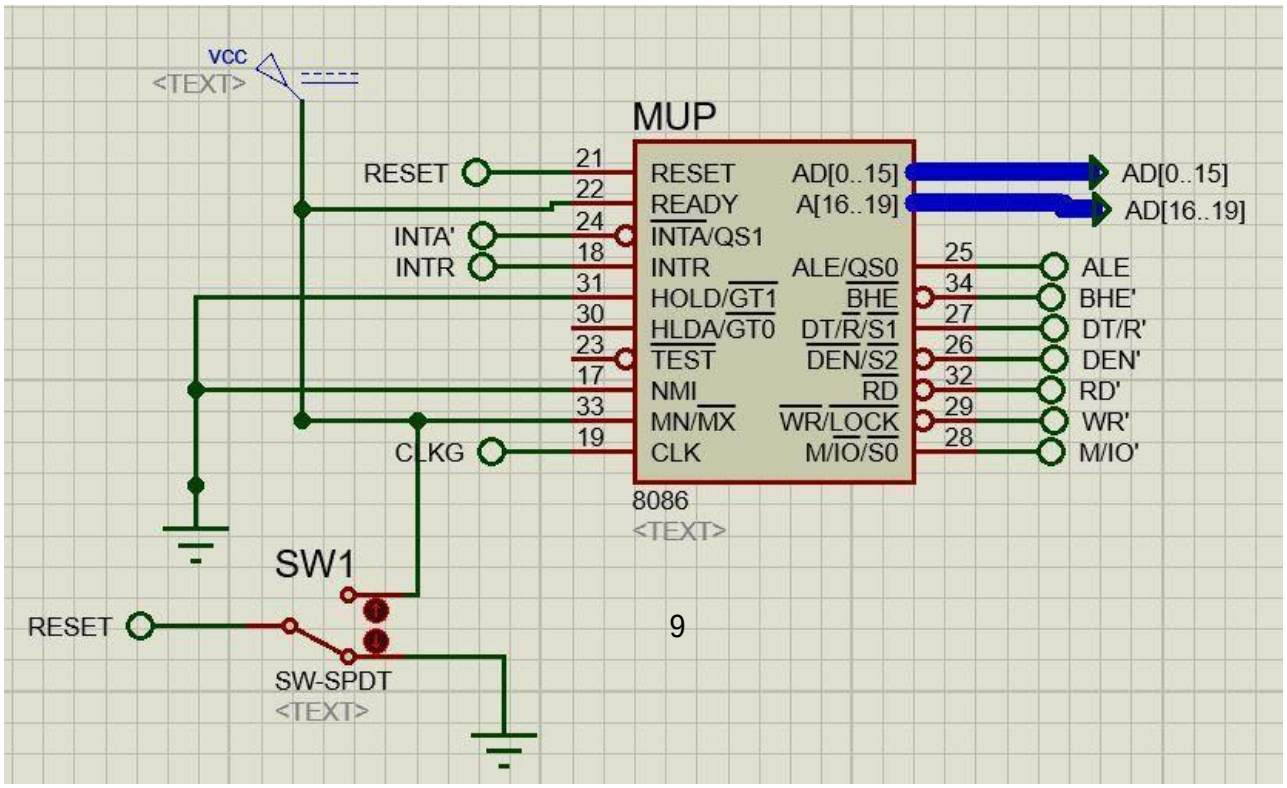
8255(2):

Port	Port Address	Mode	Input / Output
A	20H	0	Output
B	22H	0	Input
C lower	24H	0	Output
C upper	24H	0	Input
Control Register	26H		

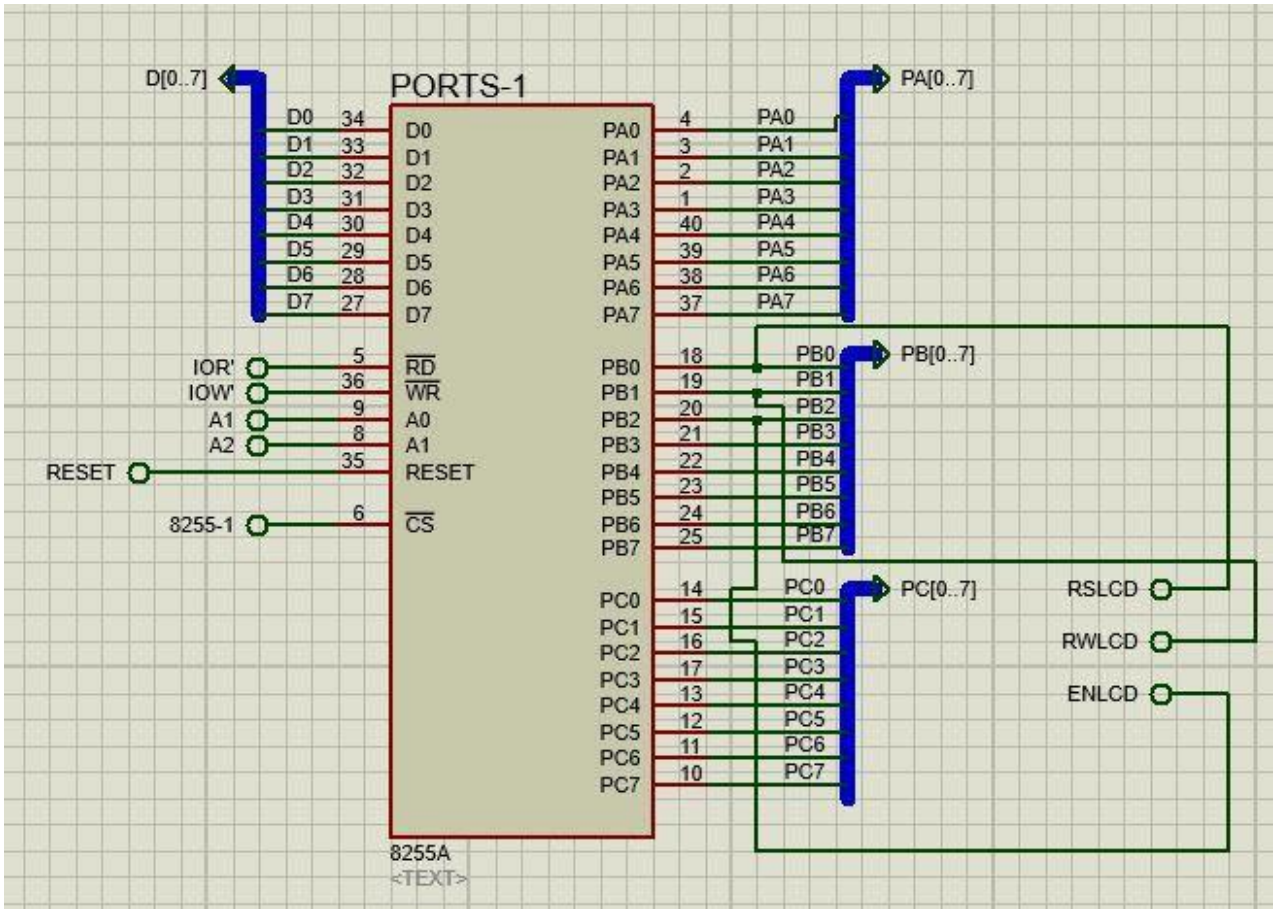
Circuit Diagrams:

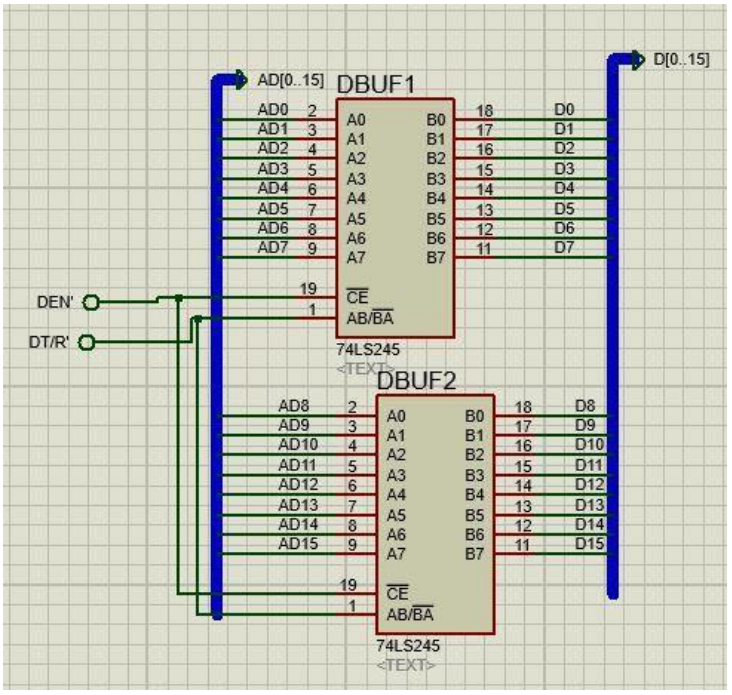


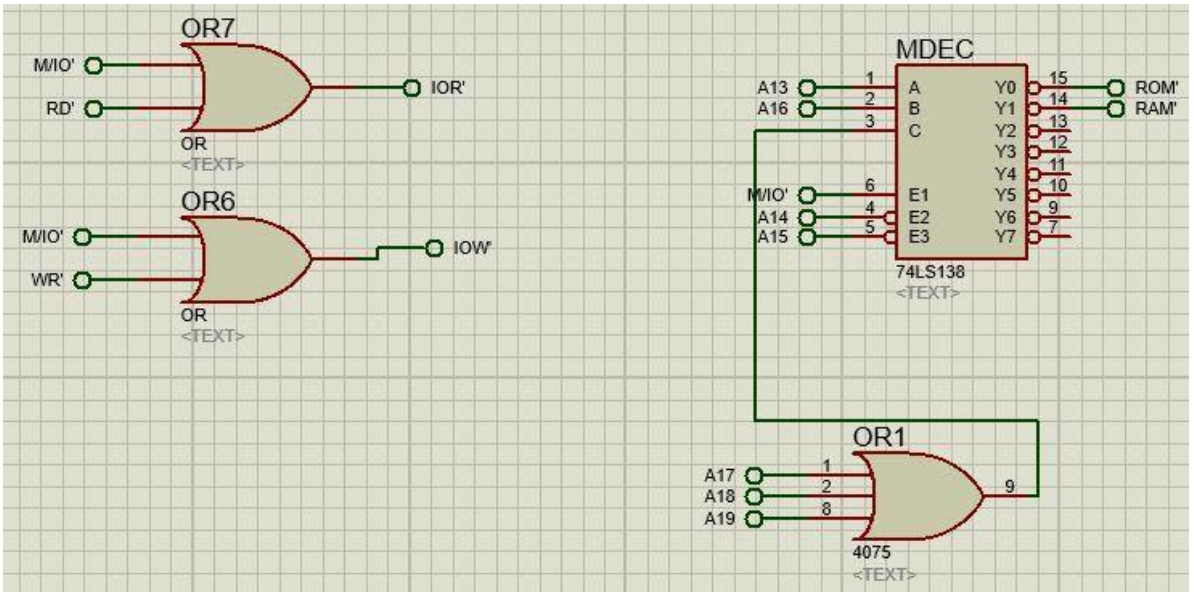


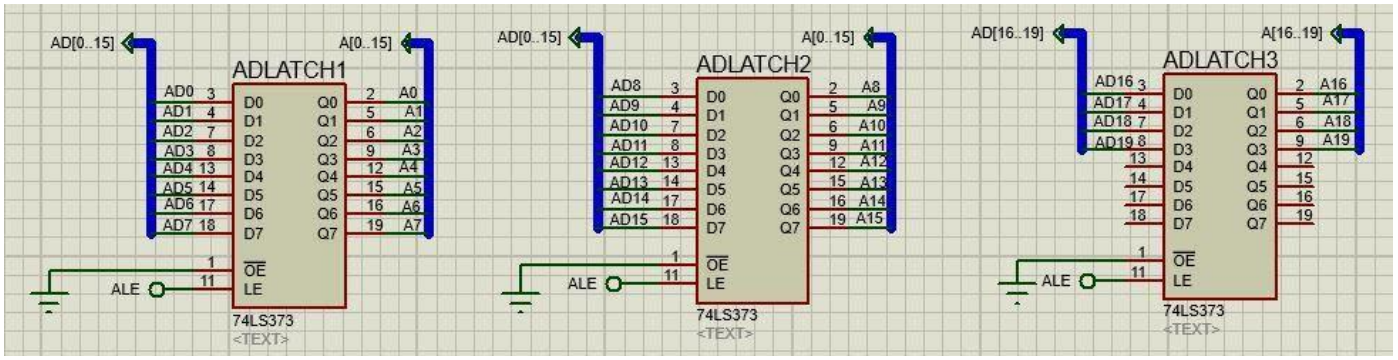

VCC
<TEXT>
MUP
RESET
INTA'
INTR
CLKG
21
22
24
18
31
30
23
17
33
19
RESET
READY
INTA/QS1
INTR
HOLD/GT1
HLDA/GT0
TEST
NMI
MN/MX
CLK
AD[0..15]
A[16..19]
ALE/QS0
BHE
DT/R/S1
DEN/S2
RD
WR/LOCK
M/IO/S0
AD[0..15]
AD[16..19]
25
34
27
26
32
29
28
ALE
BHE'
DT/R'
DEN'
RD'
WR'
M/IO'
8086
<TEXT>
SW1
RESET
SW-SPDT
<TEXT>

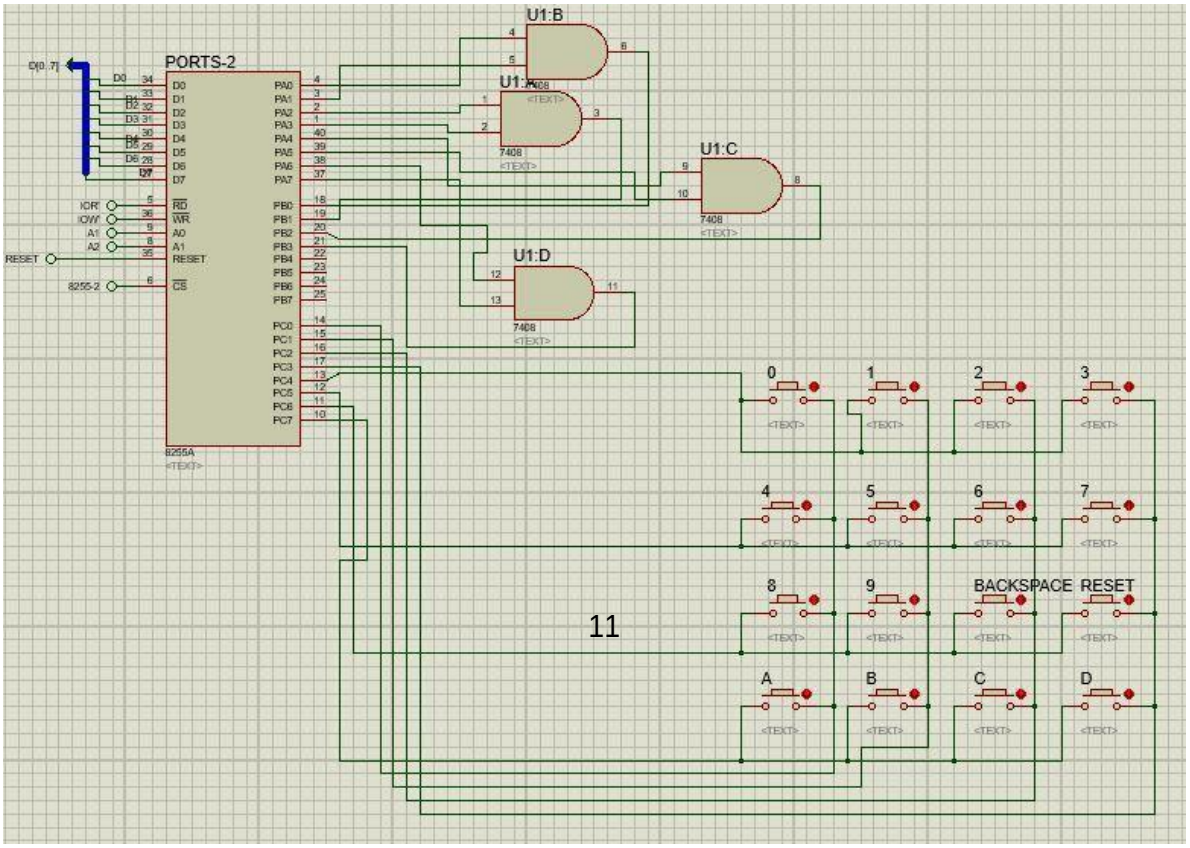
9


D[0..7]
PORTS-1
D0
D1
D2
D3
D4
D5
D6
D7
34
33
32
31
30
29
28
27
IOR'
IOW'
A1
A2
RESET
8255-1
5
36
9
8
35
6
RD
WR
A0
A1
RESET
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
4
3
2
1
40
39
38
37
18
19
20
21
22
23
24
25
14
15
16
17
13
12
11
10
PA[0..7]
PB[0..7]
PC[0..7]
RSLCD
RWLCD
ENLCD
8255A
<TEXT>

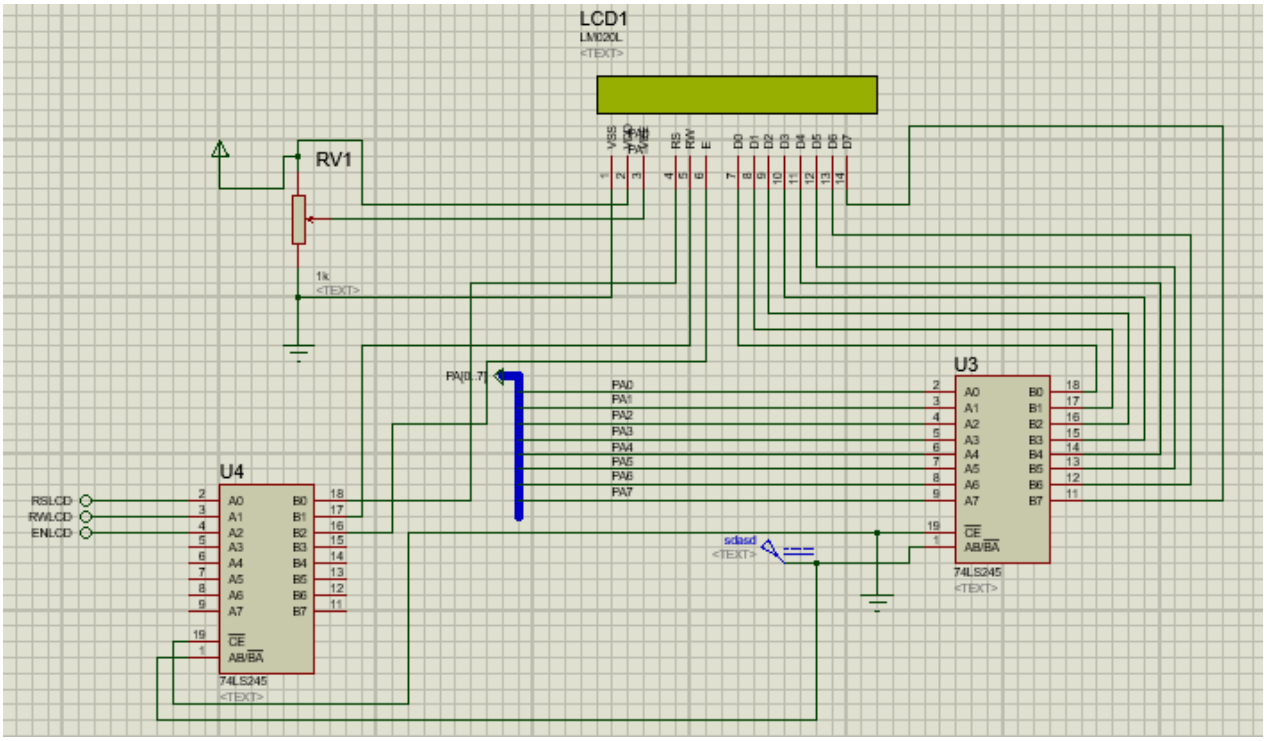

AD[0..15]
DBUF1
D[0..15]
AD0 AD1 AD2 AD3 AD4 AD5 AD6 AD7
A0 A1 A2 A3 A4 A5 A6 A7
B0 B1 B2 B3 B4 B5 B6 B7
D0 D1 D2 D3 D4 D5 D6 D7
DEN'
DT/R'
CE
AB/BA
74LS245
DBUF2
AD8 AD9 AD10 AD11 AD12 AD13 AD14 AD15
D8 D9 D10 D11 D12 D13 D14 D15
74LS245

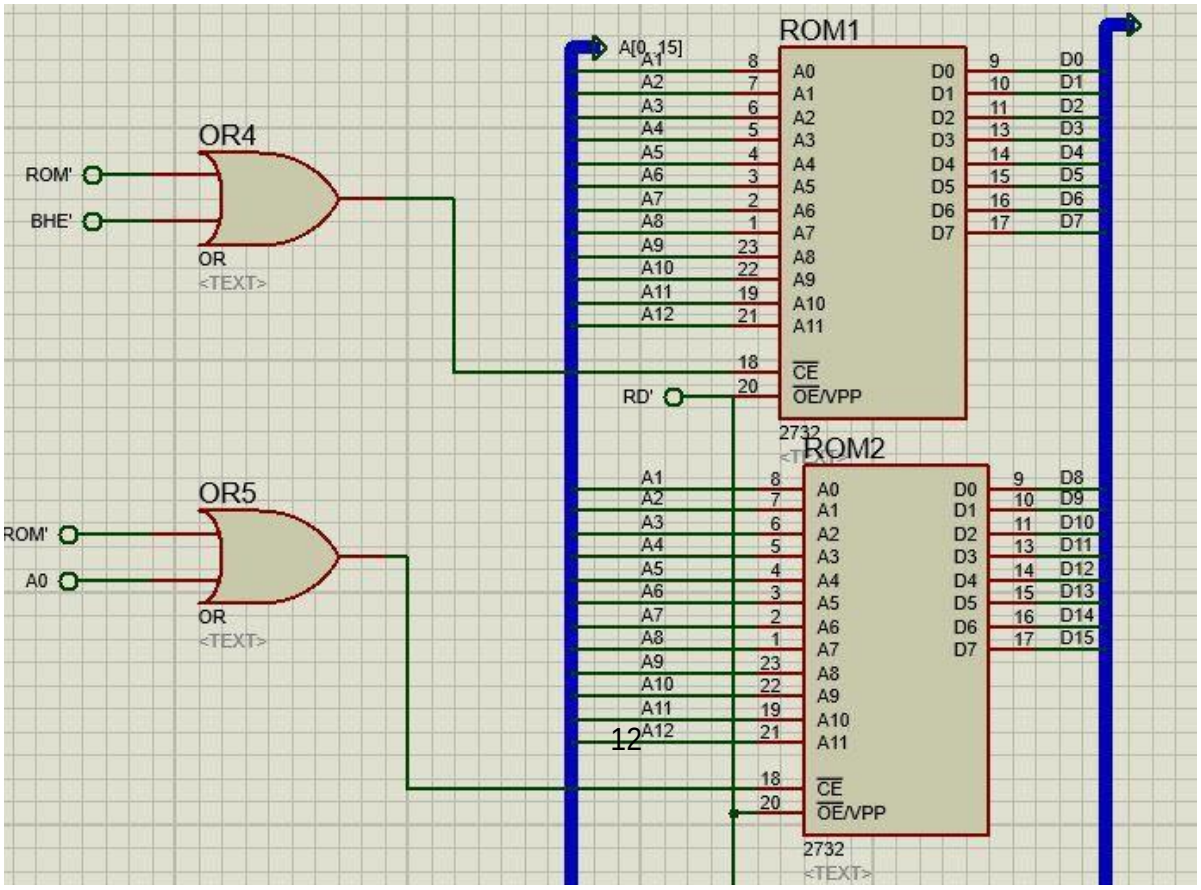

OR7
M/IO'
RD'
IOR'
OR
OR6
WR'
IOW'
MDEC
A13
A16
M/IO'
A14
A15
A
B
C
E1
E2
E3
Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
ROM'
RAM'
74LS138
OR1
A17
A18
A19
4075

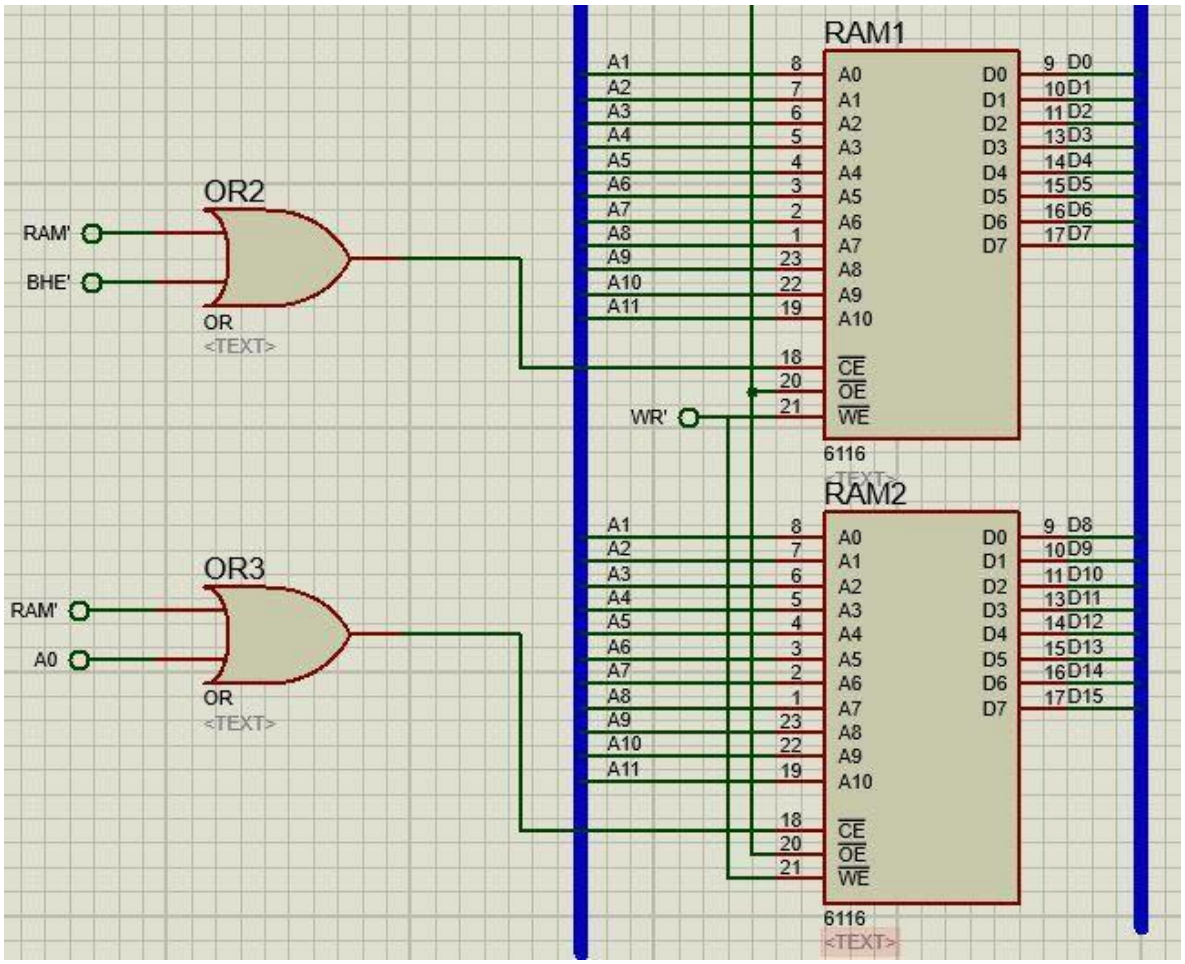

AD[0..15]
A[0..15]
ADLATCH1
AD0 AD1 AD2 AD3 AD4 AD5 AD6 AD7
D0 D1 D2 D3 D4 D5 D6 D7
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
A0 A1 A2 A3 A4 A5 A6 A7
ALE
OE
LE
74LS373
ADLATCH2
AD8 AD9 AD10 AD11 AD12 AD13 AD14 AD15
A8 A9 A10 A11 A12 A13 A14 A15
AD[16..19]
A[16..19]
ADLATCH3
AD16 AD17 AD18 AD19
A16 A17 A18 A19


PORTS-2
8255A
U1:B
U1:A
U1:C
U1:D
7408
0
1
2
3
4
5
6
7
8
9
BACKSPACE
RESET
A
B
C
D

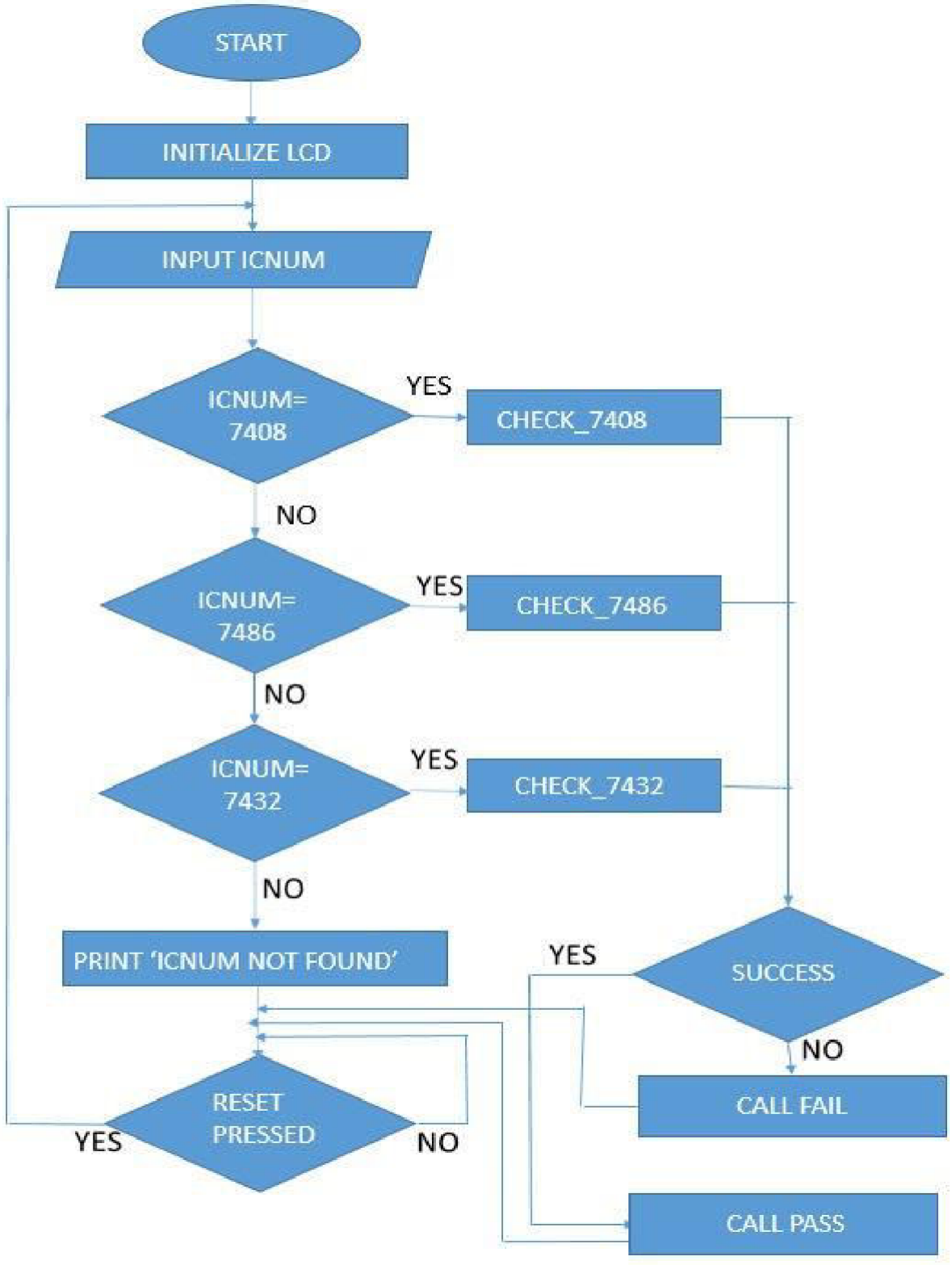
11


LCD1
LM020L
RV1
1k
U3
U4
74LS245
RSLCD
RWLCD
ENLCD
PA[0..7]

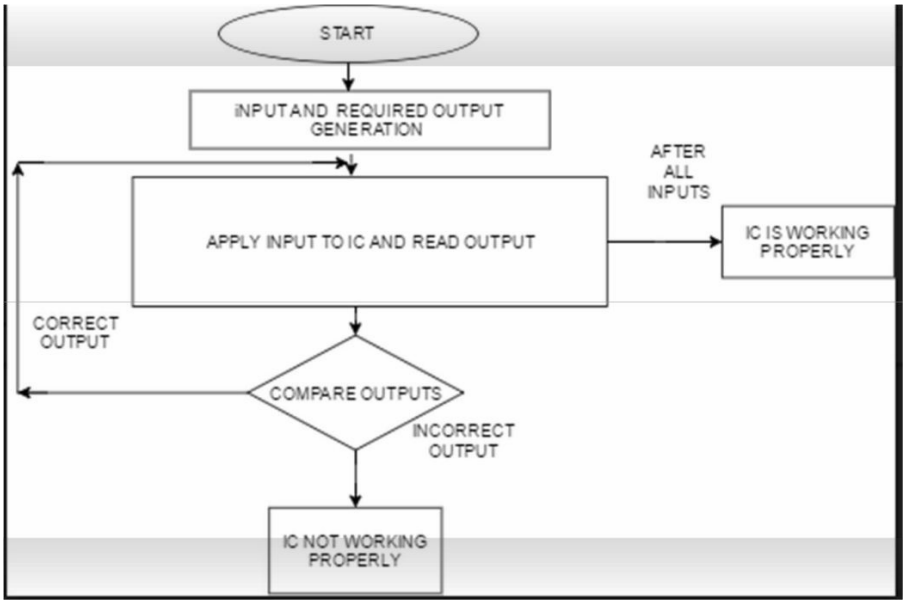

ROM1
A[0..15]
OR4
ROM'
BHE'
OR
<TEXT>
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
CE
OE/VPP
D0
D1
D2
D3
D4
D5
D6
D7
RD'
2732
ROM2
OR5
ROM'
A0
OR
<TEXT>
12
2732
<TEXT>


RAM1
OR2
RAM'
BHE'
OR
<TEXT>
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
CE
OE
WE
D0
D1
D2
D3
D4
D5
D6
D7
WR'
6116
RAM2
OR3
RAM'
A0
OR
<TEXT>
6116
<TEXT>

FLOWCHART:


START
INITIALIZE LCD
INPUT ICNUM
ICNUM= 7408
YES
CHECK_7408
NO
ICNUM= 7486
YES
CHECK_7486
NO
ICNUM= 7432
YES
CHECK_7432
NO
PRINT 'ICNUM NOT FOUND'
SUCCESS
YES
NO
CALL FAIL
RESET PRESSED
YES
NO
CALL PASS

FLOWCHART TO TEST ICs:



CODE:

```
.MODEL TINY
.DATA

FAIL_M        DB      " FAIL"
PASS_M        DB      " PASS"

NotFound      DB     " NOT FOUND"    ; when incorrect IC is entered
Flag          DB     00H             ; 00H-IC currently not found(i.e. IC num not in 7408,7486,7432) ,01H- correct IC num entered

IC7408        DB     "7408" ; ICs that we need to check
IC7432        DB     "7432"
IC7486        DB     "7486"

input2        DB     "ENTER -  "; asks user to enter an IC number
NumChars      DW      00H    ; keeps count of number of characters entered

COUNT         DW      0000H    ; keeps count of number of characters to be printed by WriteStr procedure

;Test values for checking correctness
IN7408        DB      00H,55H,0AAH,0FFH   ; AND gate
OUT7408       DB      00H,00H,00H,0FH
```

```
IN7486          DB      00H,55H,0AAH,0FFH    ; XOR gate
OUT7486         DB      00H,0FH,0FH,00H
IN7432          DB      00H,55H,0AAH,0FFH    ; OR gate
OUT7432         DB      00H,0FH,0FH,0FH

;display table for LCD
TABLE_D         DW      0H,  01H, 2H,  3H
                DW      4H,  5H,  6H,  7H
                DW      8H,  9H,  0AH, 0BH
                DW      0CH, 0DH, 0EH, 0FH
; kb table for keyboard input
TABLE_K         DB     0EEh,0EDH,0EBH,0E7H
                DB     0DEH,0DDH,0DBH,0D7H
                DB     0BEH,0BDH,0BBH,0B7H
                DB     07EH,07DH,07BH,077H

; First 8255
port1a          equ     10h
port1b          equ     12h
port1c          equ     14h
creg1           equ     16h
; Second 8255
port2a          equ     20h
port2b          equ     22h
port2c          equ     24h
creg2           equ     26h

All_Keys            DB      "0123456789SRABCD"  ;list of keypad keys
```

```
ICNUM          DB      "XXXX"              ;stores the IC number entered by the
user

.CODE
.STARTUP

MOV       AL,010001010B  ; port1a and lower port1c as output and port1b
and upper port1c as input
OUT         creg2,AL

CALL LCD_INIT   ;initialises LCD unit

Start:
   MOV NumChars,0
   MOV Flag,00H
   CALL CLS           ; clears LCD
   LEA DI,input2   ; di stores address of "ENTER-"
   MOV COUNT,9        ; no of characters to be printed
   CALL WriteStr      ; prints input message to LCD

ReadKey:
   MOV COUNT,01H
   CALL KEYBRD        ; keypush is read. AX contains offset in kb table
   LEA DI,All_Keys        ; All_Keys is list of keypad keys
   ADD DI,AX          ; pointing DI to key pressed
   MOV SI,NumChars
   MOV AL,[DI]        ; reads key pushed into al
   CMP AL,"R"         ; On Reset go to start of program
```

```
    JE Start
    CMP AL,"S"          ; if backspace and at least one character has been
entered go to backspace proc
    JNE  StoreKey
    CMP NumChars,00H    ; No char enetered. Go back to ReadKey
    JE ReadKey
    ; Deleting the previous character
    CALL BACKSPACE
    DEC NumChars
    JMP ReadKey

; AL contains the last key pressed
StoreKey:
    MOV ICNUM[SI],AL    ; stores the key pressed. ICNUM[SI] = [SI +
ICNUM]
    CALL WriteStrNext   ; write the char in AL to LCD
    INC NumChars
    ; IC number has 4 characters
    CMP NumChars,04H
    JZ WriteICName
    JMP ReadKey

; writes ic number entered by user onto LCD again after clearing it
WriteICName:
    LEA DI,ICNUM
    MOV COUNT,04H
    CALL WriteStr
; Now checking which IC is it, or if input is wrong
```

```
isIt7408:
    LEA BX,IC7408
    CALL CMP_IC_NUM         ; check if ICNUM is equal to 4 digits in BX
    CMP Flag,01H
    JNE isIt7486            ; if Flag then check if IC is good or bad
    CALL Check7408
    JMP S4

isIt7486:
    LEA BX,IC7486
    CALL CMP_IC_NUM
    CMP Flag,01H
    JNE isIt7432
    CALL Check7486
    JMP S4

isIt7432:
    LEA BX,IC7432
    CALL CMP_IC_NUM
    CMP Flag,01H
    JNE NO_IC
    CALL Check7432
    JMP S4

; Invalid Input
NO_IC:
    LEA DI,NotFound
    MOV COUNT,10    ; if no ic found then writes ICNUM "not found" on LCD
```

```
    CALL WriteStrNext

S4:
    CALL KEYBRD
    LEA DI,All_Keys
    ADD DI,AX              ; Take the key pressed and put it in AL
    MOV AL,[DI]
    CMP AL,"R"             ; If reset is pressed go back to start
    JE Start
    JMP S4

.EXIT

; Procedure for reading key pressed from the keyboard
KEYBRD PROC NEAR
    PUSHF
    PUSH BX
    PUSH CX
    PUSH DX   ; SAVING THE REGISTERS USED
    MOV        AL,0FFH
    OUT        port2c,AL

    ; Checking all All_Keys are open
    X0:         MOV           AL,00H
                OUT           port2c,AL
    ChkOpn:  IN            AL, port2c
                AND           AL,0F0H
                CMP           AL,0F0H
```

```
                JNZ          ChkOpn                ; Means the key is still
pressed, go back to X1
                CALL         D20MS                 ;key debounce check
                MOV          AL,00H
                OUT          port2c ,AL            ;provide column values as output
through lower port C

             ; BL has 0 on col no. Al has 0 on row no.
GetRowCol:  IN           AL,  port2c
                AND          AL,0F0H
                CMP          AL,0F0H
                JZ           GetRowCol
                CALL         D20MS                 ;key debounce check
                MOV          AL,00H
                OUT          port2c ,AL        ;provide column values as output
through lower port C
                IN           AL,  port2c
                AND          AL,0F0H
                CMP          AL,0F0H
                JZ           GetRowCol             ;key debounce check for
unintentional keypress
             ; Checking the first row
                MOV          AL, 0EH         ;E = 1110
                MOV          BL,AL
                OUT          port2c,AL
                IN           AL, port2c
                AND          AL,0F0H
                CMP          AL,0F0H
```

```
        JNZ         RowCol
        ; Checking the second row
        MOV         AL, 0DH       ; D = 1101
        MOV         BL,AL
        OUT         port2c ,AL
        IN          AL, port2c
        AND         AL,0F0H
        CMP         AL,0F0H
        JNZ         RowCol
        ; Checking the third row
        MOV         AL, 0BH       ; B = 1011
        MOV         BL,AL
        OUT         port2c,AL
        IN          AL, port2c
        AND         AL,0F0H
        CMP         AL,0F0H
        JNZ         RowCol
        ; Checking the fourth row
        MOV         AL, 07H       ; 7 = 0111
        MOV         BL,AL
        OUT         port2c,AL
        IN          AL, port2c
        AND         AL,0F0H
        CMP         AL,0F0H
        JZ          GetRowCol
  ; BL(lower nibble) has 0 on col no. Al(upper nibble) has 0 on row no.
  ; This converts into RowCol format to be checked from kb table. Puts into
Al.
```

```
RowCol:     OR          AL,BL
            MOV         CX,0FH
            MOV         DI,00H

   ; Goes through all the 16 possible key combo to tell which key is pressed.
FindKey:    CMP         AL,TABLE_K[DI]
            JZ          Over
            INC         DI
            LOOP        FindKey

Over:       MOV AX,DI      ; move the offset of key pressed to AX
            POP DX
            POP CX
            POP BX
            POPF
            RET
KEYBRD   ENDP



; LCD shows the output as "IC NUM -  " initially
LCD_INIT PROC NEAR
   ; initializing LCD for 2 lines & 5*7 matrix
   MOV AL, 38H
   CALL WriteCommand            ;write the command to LCD
   CALL DELAY                   ;delay before next command
   CALL DELAY
   CALL DELAY
```

```
  ; LCD ON, Show cursor
    MOV AL, 0EH
    CALL WriteCommand
    CALL DELAY
    ; clear LCD
    MOV AL, 01
    CALL WriteCommand
    CALL DELAY
    ; command for shifting cursor right
    MOV AL, 06
    CALL WriteCommand
    CALL DELAY
    RET
LCD_INIT ENDP


; This procedure just clears the LCD
CLS PROC
    MOV AL, 01          ;AL=00000001, RSLCD is set to 1
    CALL WriteCommand
    CALL DELAY
    CALL DELAY
    RET
CLS ENDP

; this procedure writes command in AL to LCD
; How? -> It first sends the commands to port A which is connected to D0-
D7 of LCD
```

```
; After that it sets enable pin from high to low with RS=0 for selecting command register
; and R/W = 0 for write operation. This enable thing is mandatory for the command to be executed.
WriteCommand PROC
    MOV DX, port1a
    OUT DX, AL          ; AL contains the command, command already sent to port A
    MOV DX, port1b
    ; Enable High
    MOV AL, 00000100B
    OUT DX, AL
    ; A small pause
    NOP
    NOP
    ; Enable Low
    MOV AL, 00000000B
    OUT DX, AL
    RET
WriteCommand ENDP


; This procedure writes a string with starting add at DI having count characters to LCD
; It just picks up one character at a time and send to WriteChar for writing it
WriteStr PROC NEAR
    CALL CLS
```

```
   LoopOver:
      MOV AL, [DI]
      CALL WriteChar       ;issue it to LCD
      CALL DELAY           ; delay before next character
      CALL DELAY
      INC DI               ; Move to next character
      DEC COUNT
      JNZ LoopOver
   RET
WriteStr ENDP


; WriteStr without CLS. That is next after what is already written.
WriteStrNext PROC NEAR
   LoopOver2:
      MOV AL, [DI]
      CALL WriteChar       ;issue it to LCD
      CALL DELAY           ; delay before next character
      CALL DELAY
      INC DI               ; Move to next character
      DEC COUNT
      JNZ LoopOver2
   RET
WriteStrNext ENDP

; Write single character in AL to LCD
; R/W = 0 because we are writing. RS is 1 because data register is to be
selected.
```

```
WriteChar PROC
   PUSH DX
   MOV DX,port1a          ; DX=port A address
   OUT DX, AL             ; issue the char to LCD
   MOV AL, 00000101B
   MOV DX, port1b         ;port B address
   OUT DX, AL
   MOV AL, 00000001B
   OUT DX, AL
   POP DX
   RET
WriteChar ENDP



BACKSPACE PROC NEAR
   PUSH DX
   PUSH AX
   MOV AL,00010000B    ; used for shifting cursor to one space left
   Call WriteCommand
   CALL DELAY          ; wait before next command
   CALL DELAY
   MOV AL,' '
   CALL WriteChar      ; overwrite " "
   CALL DELAY
   CALL DELAY          ; wait before issuing next command
   MOV AL,00010000B    ; shifting cursor to left
   Call WriteCommand
   POP AX              ;retrive registers
```

```
    POP DX
    RET
BACKSPACE ENDP




; delay of 20ms
DELAY PROC
    MOV CX, 1325 ;1325*15.085 usec = 20 msec
    WasteTime:
            NOP
            NOP
            LOOP WasteTime
    RET
DELAY ENDP



; Compares BX and ICNUM for equality
CMP_IC_NUM PROC NEAR
    MOV SI,0000H
    CMP_NUM:
       MOV AL,ICNUM[SI]
       CMP AL,[BX+SI]
       JE NXT_NUM
       JMP EP_CMP_IC
       NXT_NUM:
       CMP SI,03H
```

```
      JE PASS_CMP_IC   ; If all chars equal set Flag to 1 before return
      INC SI
      JMP CMP_NUM
   PASS_CMP_IC:
      MOV Flag,01H
   EP_CMP_IC:
      RET
CMP_IC_NUM ENDP



Check7408 PROC NEAR  ;checks if 7408 is good or bad[pass or fail]
   MOV DI,00H

   Testing7408:
      MOV AL,IN7408[DI]
      OUT port2a,AL       ; Input for the 7408 IC
      IN AL,port2b        ; Output value of 7408 goes to AL
      AND AL,0FH          ; Masking. Only lower nibble needed.
      CMP AL,OUT7408[DI]  ; Verify by comparing with expected output.
      JE Next7408
      CALL FAIL
      JMP Ret7408
   Next7408:
      CMP DI,03H          ; All four chars are same
      JE Pass7408         ; Means Pass.
      INC DI              ; If all chars are not read yet, proceed to next char
      JMP Testing7408
```

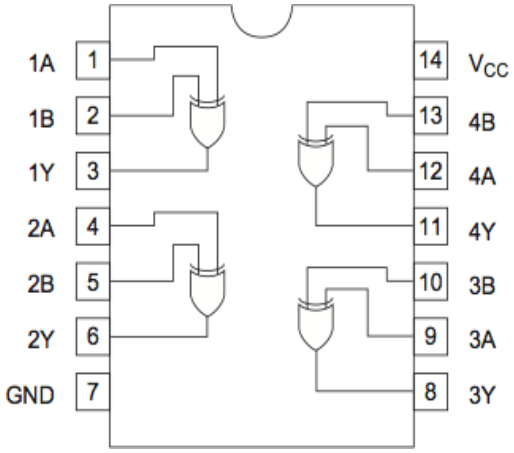
```
   Pass7408:
      CALL PASS
   Ret7408:
      RET
Check7408 ENDP



Check7432 PROC NEAR  ;checks if 7432 is good or bad[pass or fail]
   MOV DI,00H

   Testing7432:
      MOV AL,IN7432[DI]
      OUT port2a,AL       ; Input for the 7432 IC
      IN AL,port2b        ; Output value of 7432 goes to AL
      AND AL,0FH          ; Masking. Only lower nibble needed.
      CMP AL,OUT7432[DI]  ; Verify by comparing with expected output.
      JE Next7432
      CALL FAIL
      JMP Ret7432
   Next7432:
      CMP DI,03H          ; All four chars are same
      JE Pass7432         ; Means Pass.
      INC DI              ; If all chars are not read yet, proceed to next char
      JMP Testing7432
   Pass7432:
      CALL PASS
   Ret7432:
      RET
```

```
Check7432 ENDP



Check7486 PROC NEAR  ;checks if 7486 is good or bad[pass or fail]
   MOV DI,00H

   Testing7486:
      MOV AL,IN7486[DI]
      OUT port2a,AL       ; Input for the 7486 IC
      IN AL,port2b        ; Output value of 7486 goes to AL
      AND AL,0FH          ; Masking. Only lower nibble needed.
      CMP AL,OUT7486[DI]  ; Verify by comparing with expected output.
      JE Next7486
      CALL FAIL
      JMP Ret7486
   Next7486:
      CMP DI,03H          ; All four chars are same
      JE Pass7486         ; Means Pass.
      INC DI              ; If all chars are not read yet, proceed to next char
      JMP Testing7486
   Pass7486:
      CALL PASS
   Ret7486:
      RET
Check7486 ENDP

; When IC check fails
```

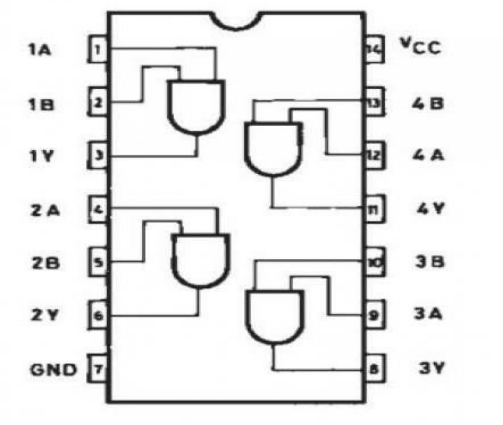
```
FAIL PROC NEAR
   PUSHF
   PUSH DI
   MOV COUNT,05
   LEA DI,FAIL_M      ; ' FAIL'
   CALL WriteStrNext    ; Writes next to the IC number
   POP DI
   POPF
   RET
FAIL ENDP



; When the IC passes the test
PASS PROC NEAR
   PUSHF
   PUSH DI
   MOV COUNT,05        ; Number of letters
   LEA DI,PASS_M     ; ' PASS'
   CALL WriteStrNext   ; Writes next to the IC number
   POP DI
   POPF
   RET
PASS ENDP
; delay generated will be approx 0.25 secs
D20MS:
        mov CX,2220
        xn:
           loop         xn
```

```
RET

END
```

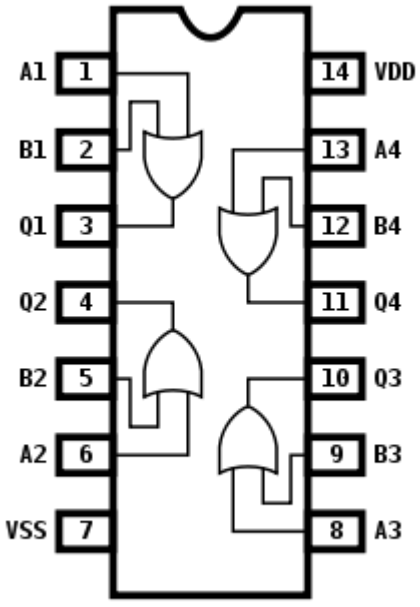
DATASHEETS OF ICs USED:



XOR 7486



AND 7408



OR 7432