Department of Computer Engineering
Digital Hardware Systems
CpE 3104 - Microprocessors

Laboratory Report

Laboratory Exercise No.:	7	**Date Performed:**	November 6, 2024
Laboratory Exercise Title:	Hardware Interrupt Interfacing		
Name of Student:	Ivor Louisetyne Canque May G. Ochia	**Document Version:**	1.0

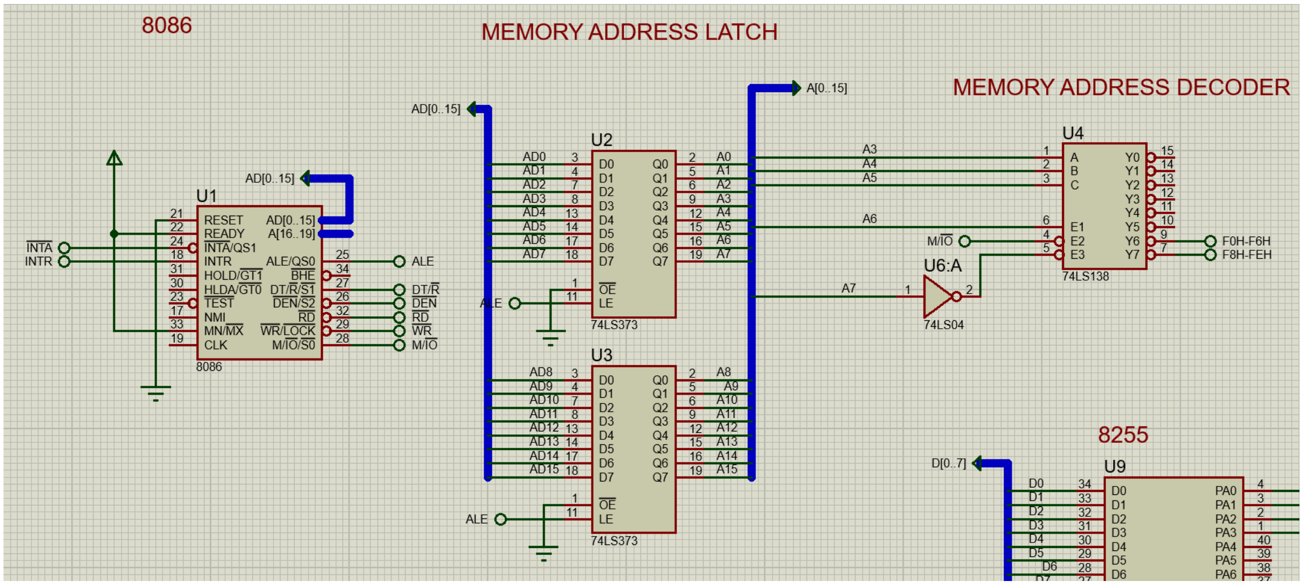
Activity #1

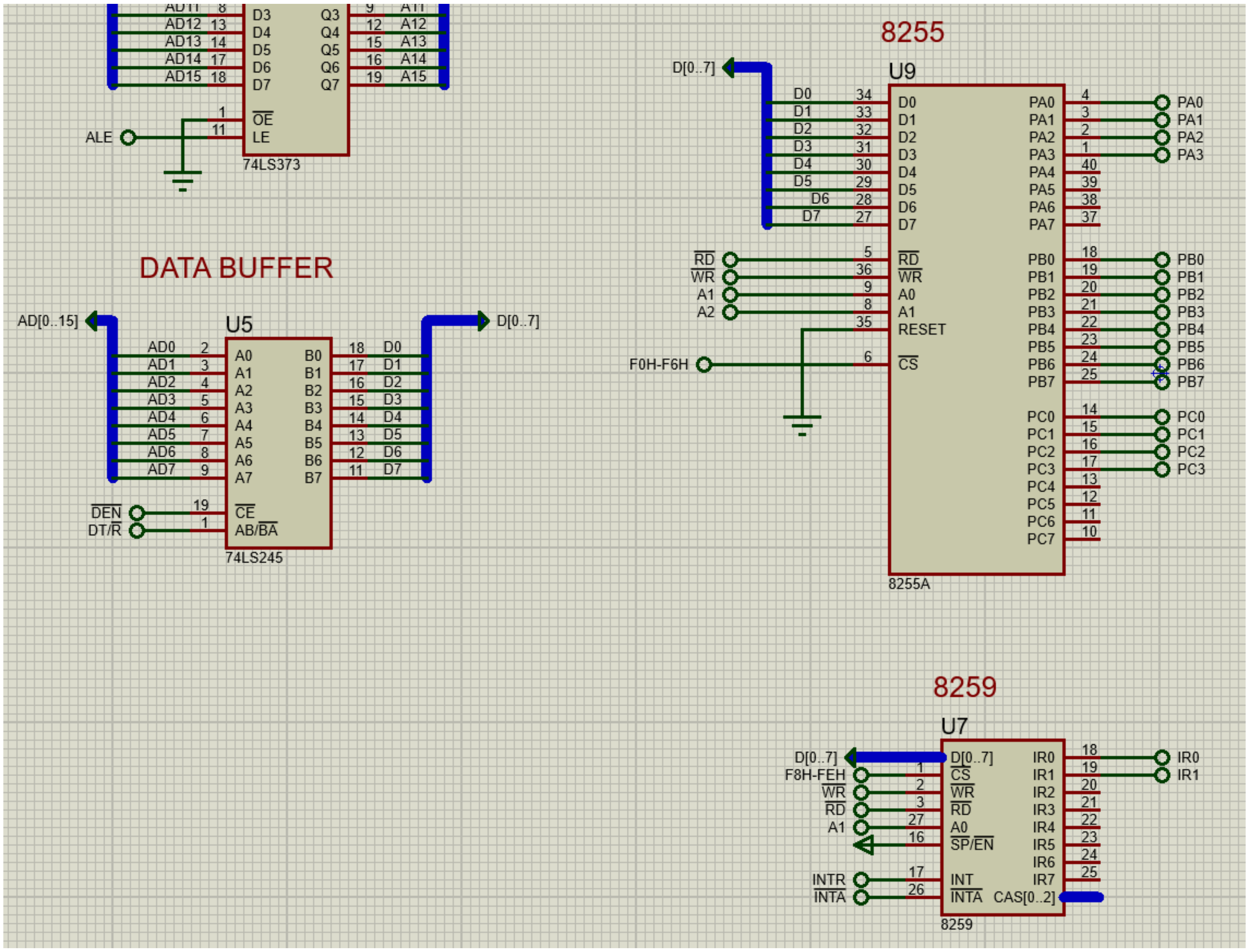
ICW4 needed, Single Mode, Call Address Interval of 8, Edge Triggered ICW1 = __13H______

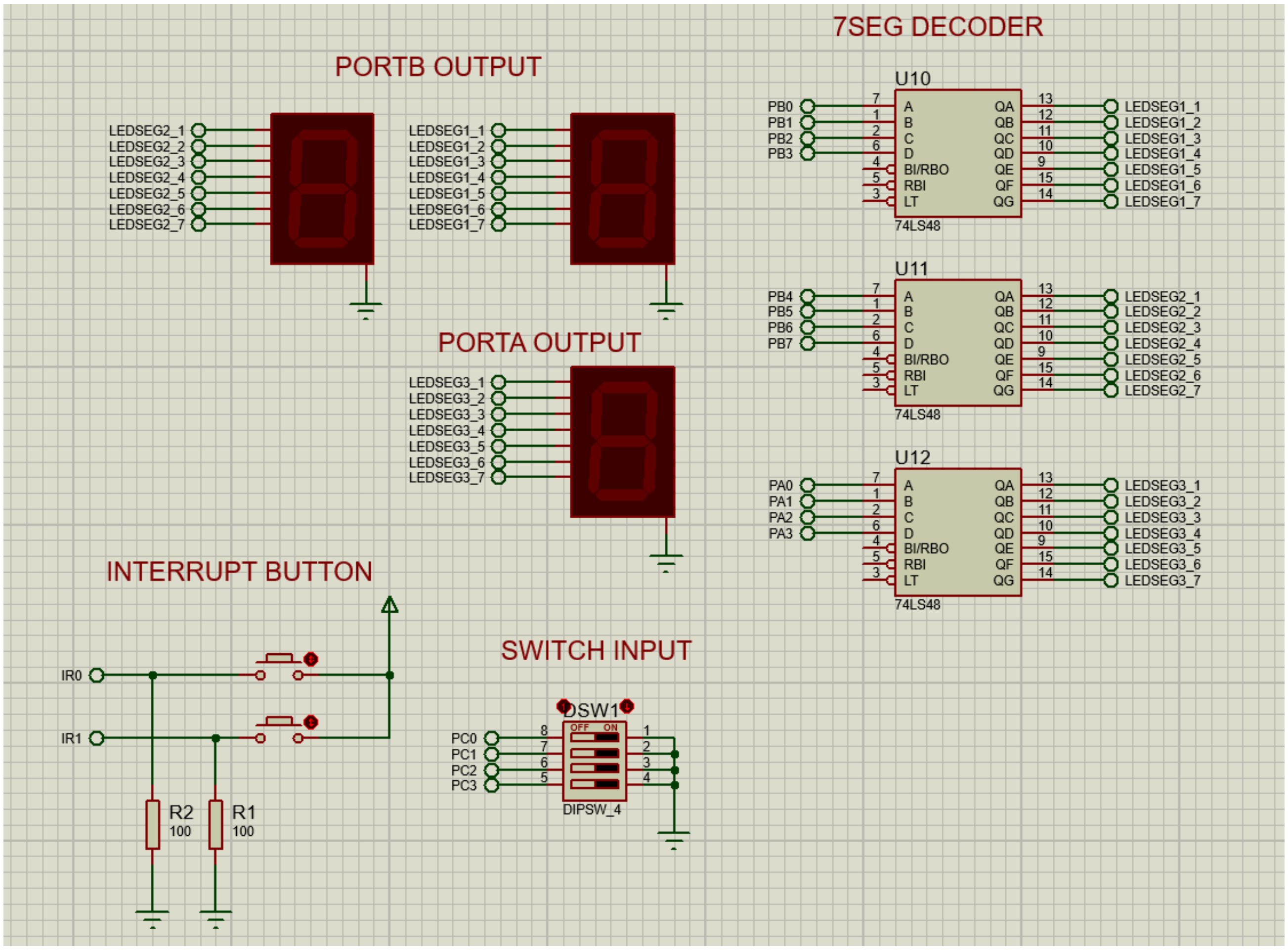
Interrupt Vector Address: 80H-87H ICW2 = ____80H____

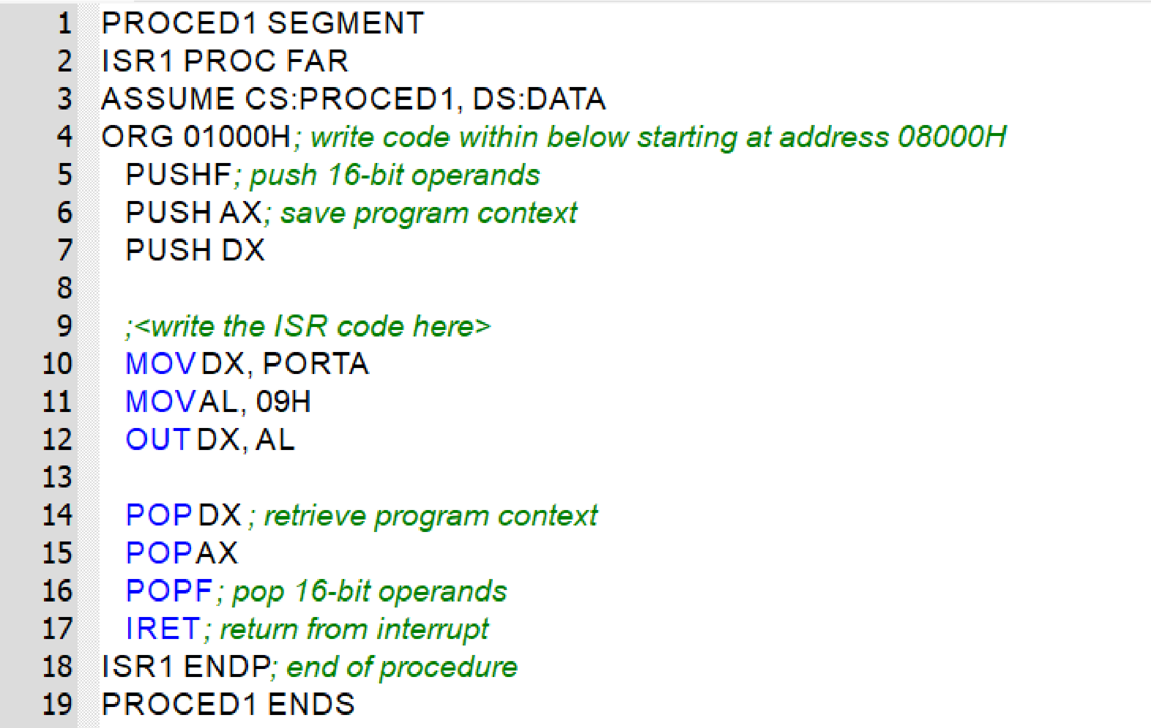
8086 Mode, Auto EOI ICW4 = _____03H___

only IR0 and IR1 are unmasked OCW11 = ____0FCH____

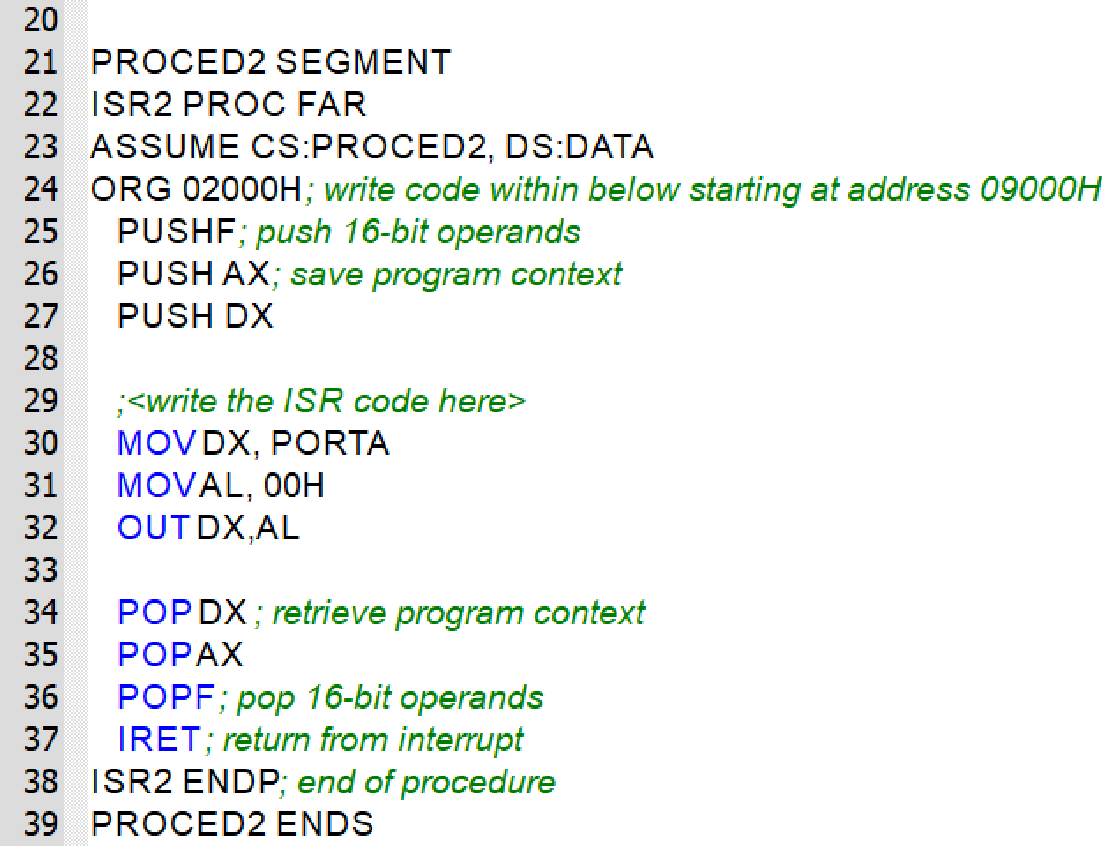



AD11 8 D3 Q3 9 A11
AD12 13 D4 Q4 12 A12
AD13 14 D5 Q5 15 A13
AD14 17 D6 Q6 16 A14
AD15 18 D7 Q7 19 A15
ALE
1 OE
11 LE
74LS373
DATA BUFFER
U5
AD[0..15]
D[0..7]
AD0 2 A0 B0 18 D0
AD1 3 A1 B1 17 D1
AD2 4 A2 B2 16 D2
AD3 5 A3 B3 15 D3
AD4 6 A4 B4 14 D4
AD5 7 A5 B5 13 D5
AD6 8 A6 B6 12 D6
AD7 9 A7 B7 11 D7
DEN 19 CE
DT/R 1 AB/BA
74LS245
8255
U9
D[0..7]
D0 34 D0 PA0 4 PA0
D1 33 D1 PA1 3 PA1
D2 32 D2 PA2 2 PA2
D3 31 D3 PA3 1 PA3
D4 30 D4 PA4 40
D5 29 D5 PA5 39
D6 28 D6 PA6 38
D7 27 D7 PA7 37
RD 5 RD PB0 18 PB0
WR 36 WR PB1 19 PB1
A1 9 A0 PB2 20 PB2
A2 8 A1 PB3 21 PB3
35 RESET PB4 22 PB4
PB5 23 PB5
F0H-F6H 6 CS PB6 24 PB6
PB7 25 PB7
PC0 14 PC0
PC1 15 PC1
PC2 16 PC2
PC3 17 PC3
PC4 13
PC5 12
PC6 11
PC7 10
8255A
8259
U7
D[0..7] D[0..7] IR0 18 IR0
F8H-FEH 1 CS IR1 19 IR1
WR 2 WR IR2 20
RD 3 RD IR3 21
A1 27 A0 IR4 22
16 SP/EN IR5 23
IR6 24
INTR 17 INT IR7 25
INTA 26 INTA CAS[0..2]
8259

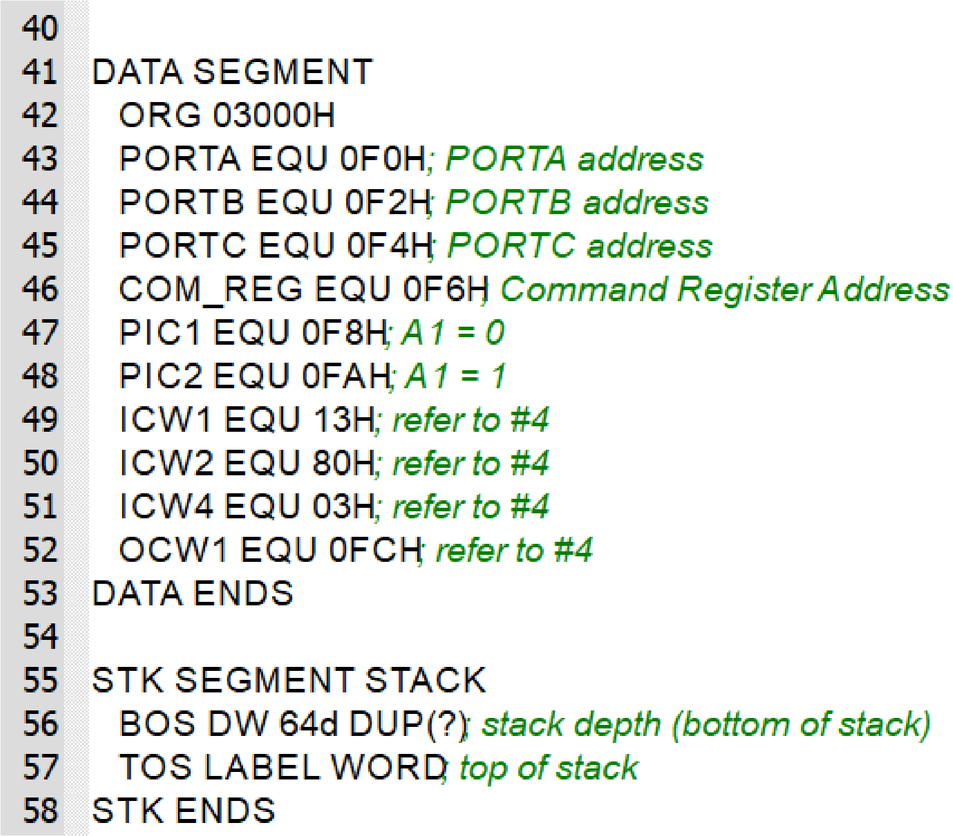




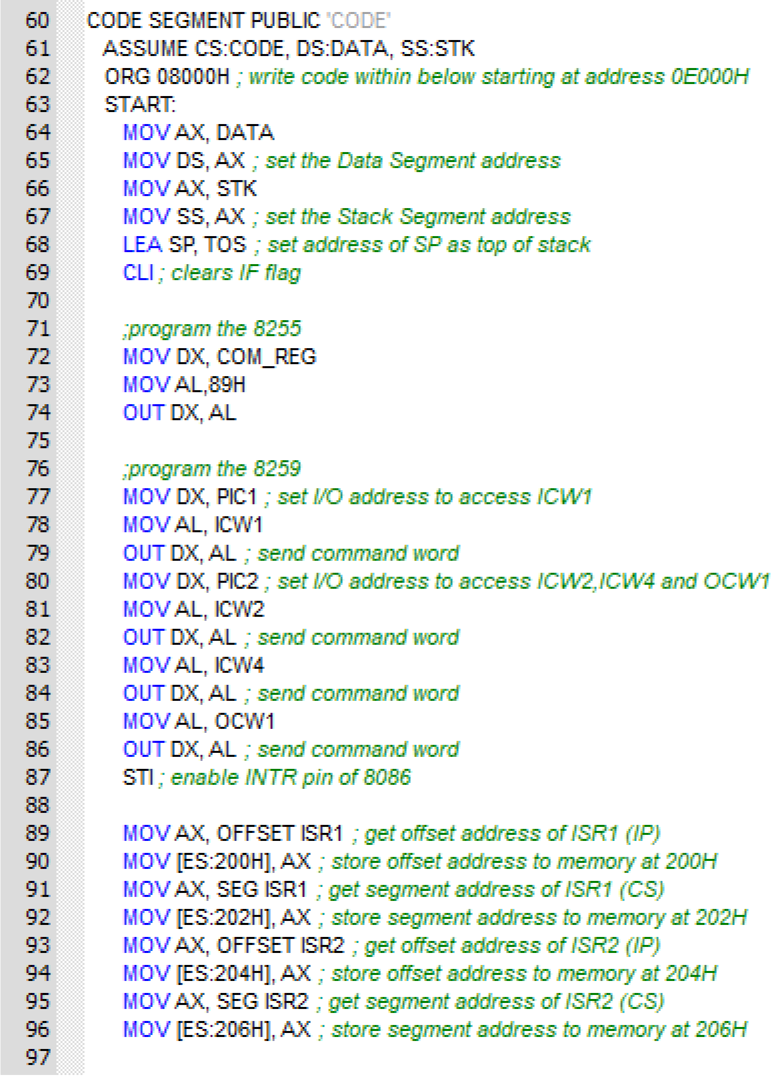
```
PROCED1 SEGMENT
ISR1 PROC FAR
ASSUME CS:PROCED1, DS:DATA
ORG 01000H; write code within below starting at address 08000H
  PUSHF; push 16-bit operands
  PUSH AX; save program context
  PUSH DX

  ;<write the ISR code here>
  MOV DX, PORTA
  MOV AL, 09H
  OUT DX, AL

  POP DX ; retrieve program context
  POP AX
  POPF; pop 16-bit operands
  IRET; return from interrupt
ISR1 ENDP; end of procedure
PROCED1 ENDS
```



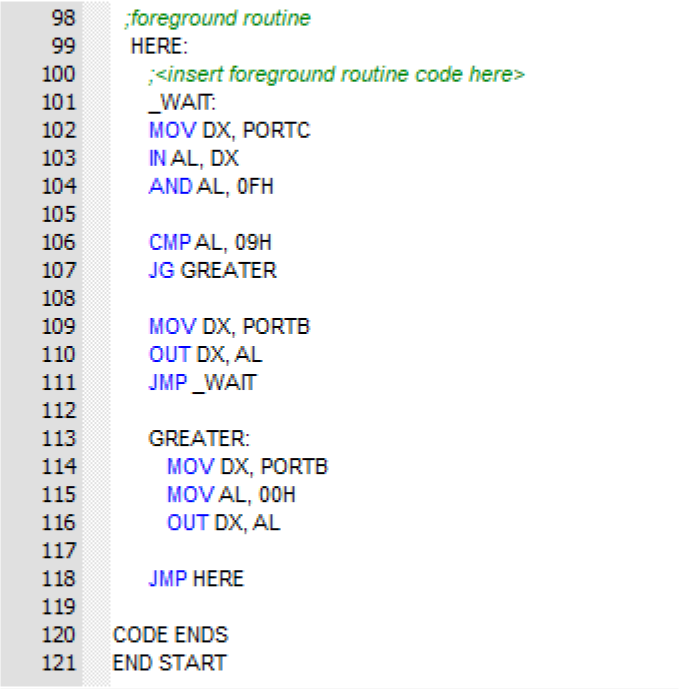
```

PROCED2 SEGMENT
ISR2 PROC FAR
ASSUME CS:PROCED2, DS:DATA
ORG 02000H; write code within below starting at address 09000H
  PUSHF; push 16-bit operands
  PUSH AX; save program context
  PUSH DX

  ;<write the ISR code here>
  MOV DX, PORTA
  MOV AL, 00H
  OUT DX,AL

  POP DX ; retrieve program context
  POP AX
  POPF; pop 16-bit operands
  IRET; return from interrupt
ISR2 ENDP; end of procedure
PROCED2 ENDS
```



```

DATA SEGMENT
  ORG 03000H
  PORTA EQU 0F0H; PORTA address
  PORTB EQU 0F2H; PORTB address
  PORTC EQU 0F4H; PORTC address
  COM_REG EQU 0F6H; Command Register Address
  PIC1 EQU 0F8H; A1 = 0
  PIC2 EQU 0FAH; A1 = 1
  ICW1 EQU 13H; refer to #4
  ICW2 EQU 80H; refer to #4
  ICW4 EQU 03H; refer to #4
  OCW1 EQU 0FCH; refer to #4
DATA ENDS

STK SEGMENT STACK
  BOS DW 64d DUP(?); stack depth (bottom of stack)
  TOS LABEL WORD; top of stack
STK ENDS
```

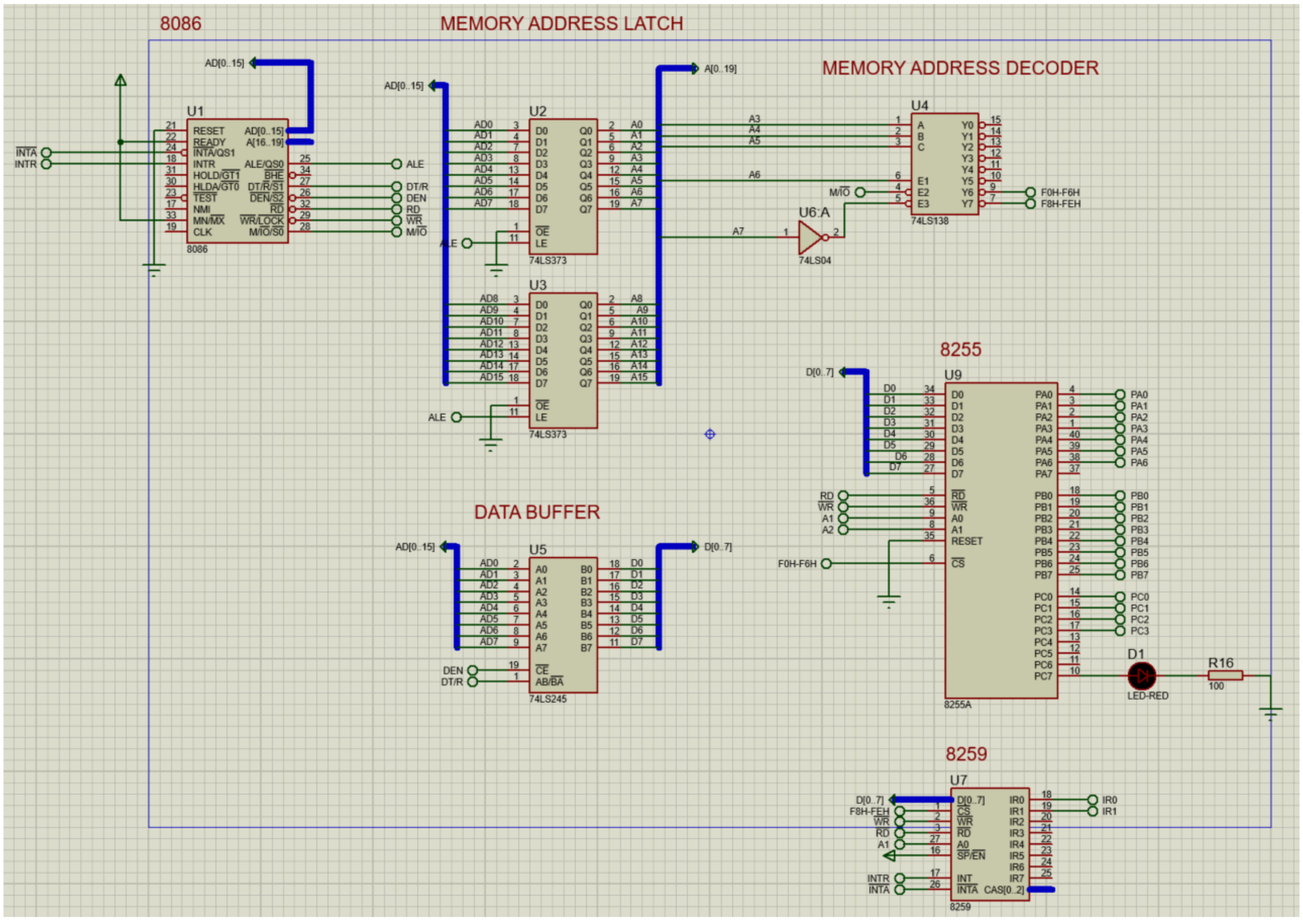


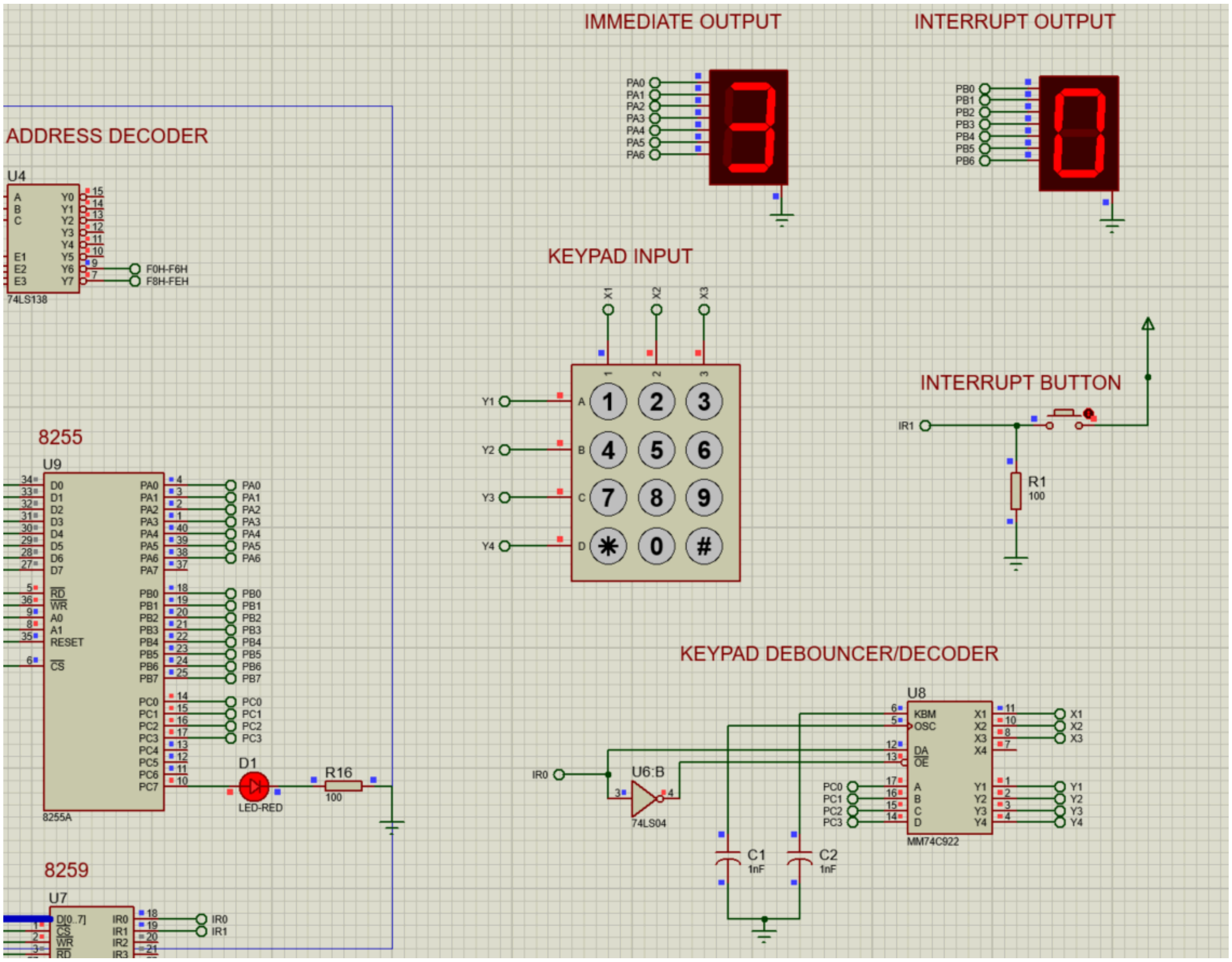
```
CODE SEGMENT PUBLIC 'CODE'
  ASSUME CS:CODE, DS:DATA, SS:STK
  ORG 08000H ; write code within below starting at address 0E000H
  START:
    MOV AX, DATA
    MOV DS, AX ; set the Data Segment address
    MOV AX, STK
    MOV SS, AX ; set the Stack Segment address
    LEA SP, TOS ; set address of SP as top of stack
    CLI ; clears IF flag

    ;program the 8255
    MOV DX, COM_REG
    MOV AL,89H
    OUT DX, AL

    ;program the 8259
    MOV DX, PIC1 ; set I/O address to access ICW1
    MOV AL, ICW1
    OUT DX, AL ; send command word
    MOV DX, PIC2 ; set I/O address to access ICW2,ICW4 and OCW1
    MOV AL, ICW2
    OUT DX, AL ; send command word
    MOV AL, ICW4
    OUT DX, AL ; send command word
    MOV AL, OCW1
    OUT DX, AL ; send command word
    STI ; enable INTR pin of 8086

    MOV AX, OFFSET ISR1 ; get offset address of ISR1 (IP)
    MOV [ES:200H], AX ; store offset address to memory at 200H
    MOV AX, SEG ISR1 ; get segment address of ISR1 (CS)
    MOV [ES:202H], AX ; store segment address to memory at 202H
    MOV AX, OFFSET ISR2 ; get offset address of ISR2 (IP)
    MOV [ES:204H], AX ; store offset address to memory at 204H
    MOV AX, SEG ISR2 ; get segment address of ISR2 (CS)
    MOV [ES:206H], AX ; store segment address to memory at 206H
```

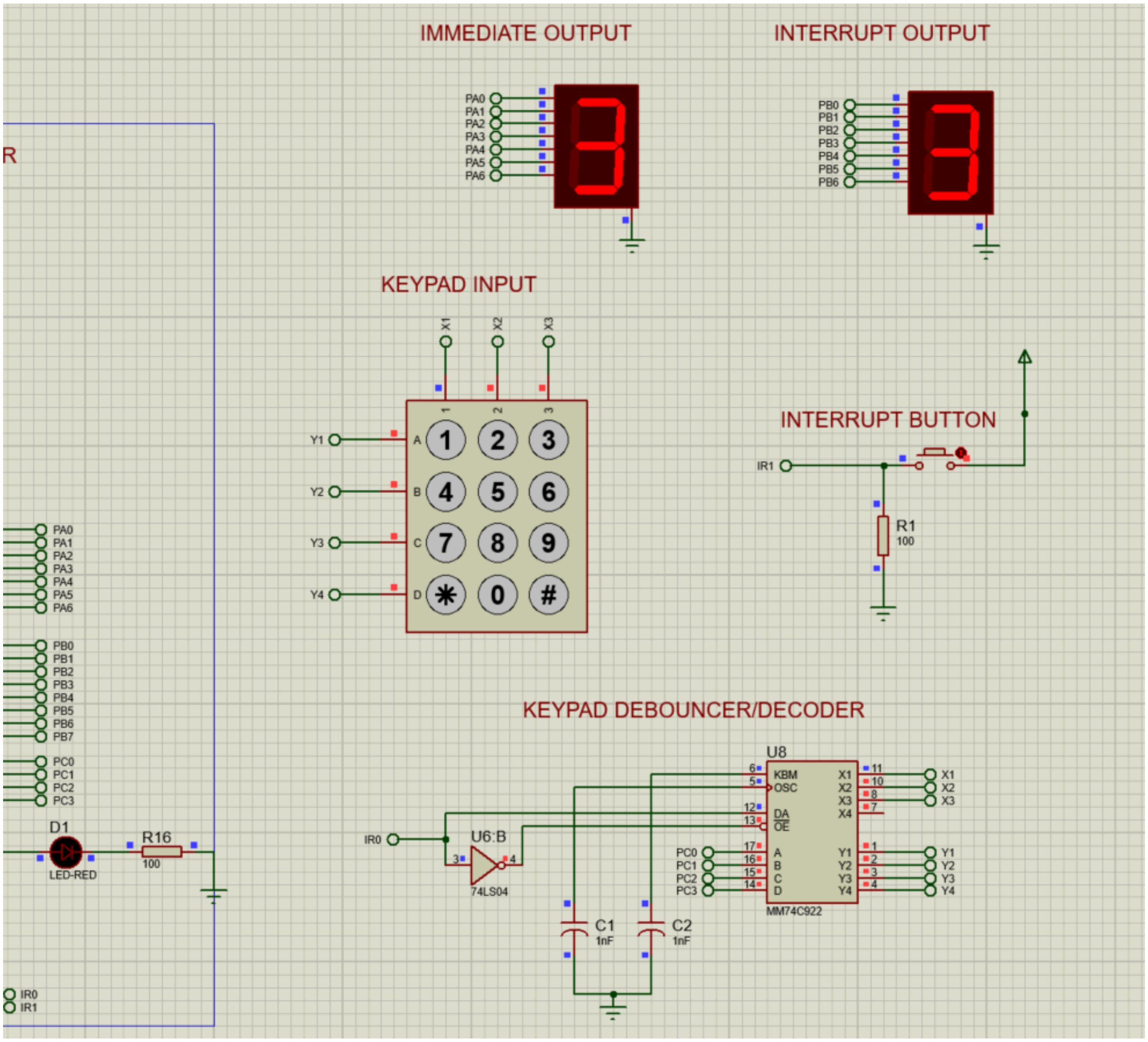


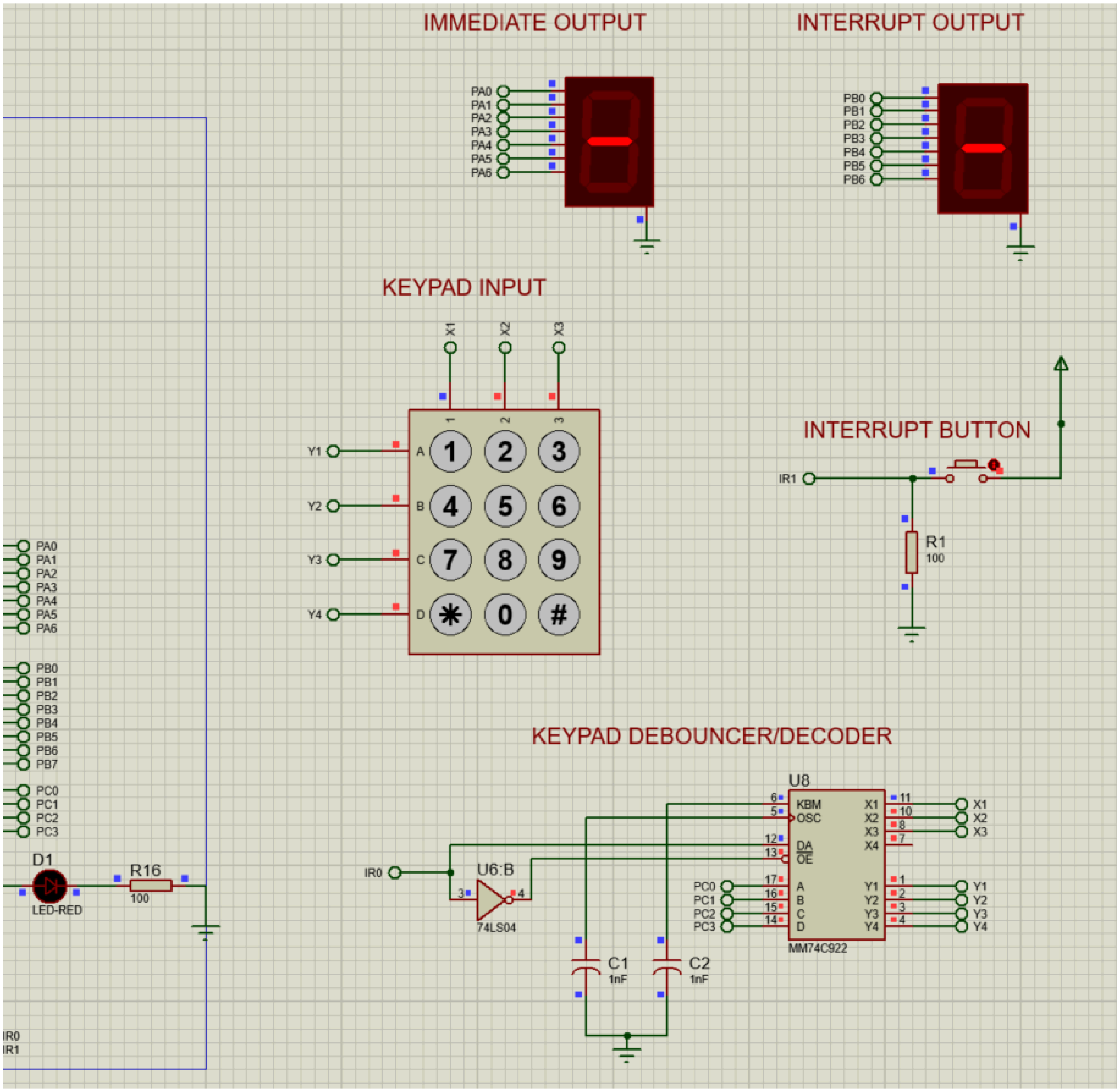
```
 ;foreground routine
  HERE:
    ;<insert foreground routine code here>
    _WAIT:
    MOV DX, PORTC
    IN AL, DX
    AND AL, 0FH

    CMP AL, 09H
    JG GREATER

    MOV DX, PORTB
    OUT DX, AL
    JMP _WAIT

    GREATER:
      MOV DX, PORTB
      MOV AL, 00H
      OUT DX, AL

    JMP HERE

CODE ENDS
END START
```

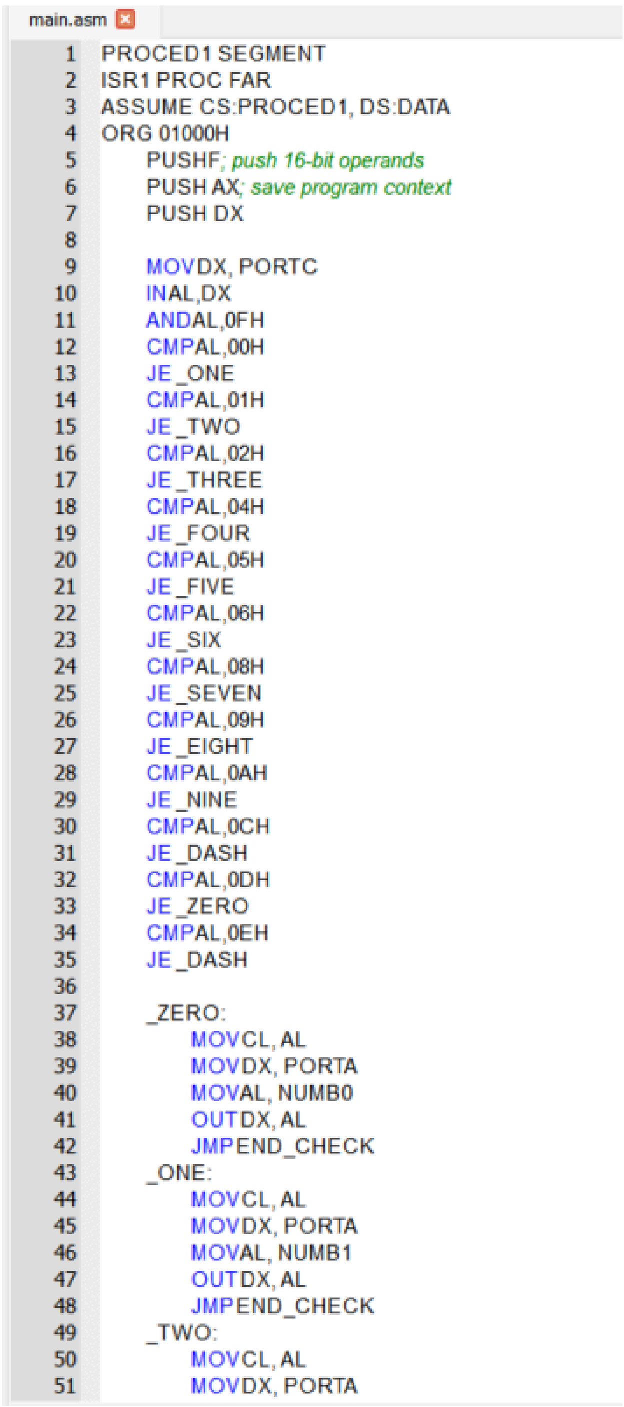
Activity #2


8086
MEMORY ADDRESS LATCH
MEMORY ADDRESS DECODER
DATA BUFFER
8255
8259
U1
8086
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
INTA
INTR
ALE
DT/R
DEN
RD
WR
M/IO
U2
74LS373
U3
74LS373
A[0..19]
U4
74LS138
A
B
C
E1
E2
E3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
F0H-F6H
F8H-FEH
U6:A
74LS04
U5
74LS245
CE
AB/BA
D[0..7]
U9
8255A
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
D1
LED-RED
R16
100
U7
8259
SP/EN
CAS[0..2]
IR0
IR1

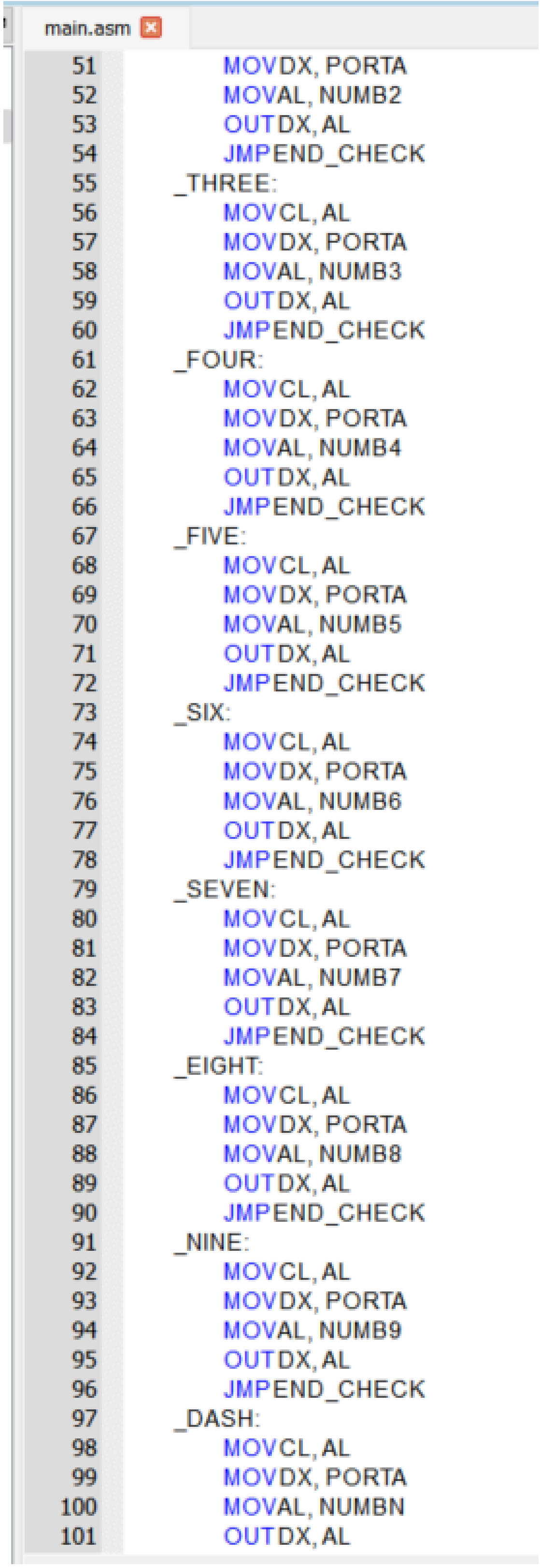

IMMEDIATE OUTPUT
INTERRUPT OUTPUT
ADDRESS DECODER
U4
A
B
C
E1
E2
E3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
74LS138
F0H-F6H
F8H-FEH
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PB0
PB1
PB2
PB3
PB4
PB5
PB6
KEYPAD INPUT
X1
X2
X3
Y1
Y2
Y3
Y4
1
2
3
4
5
6
7
8
9
✱
0
#
INTERRUPT BUTTON
IR1
R1
100
8255
U9
D0
D1
D2
D3
D4
D5
D6
D7
RD
WR
A0
A1
RESET
CS
PA7
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
8255A
D1
LED-RED
R16
100
KEYPAD DEBOUNCER/DECODER
U8
KBM
OSC
DA
OE
X4
MM74C922
IR0
U6:B
74LS04
C1
1nF
C2
1nF
8259
U7
D[0..7]
IR2
IR3


IMMEDIATE OUTPUT
INTERRUPT OUTPUT
R
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PB0
PB1
PB2
PB3
PB4
PB5
PB6
KEYPAD INPUT
X1
X2
X3
1
2
3
Y1
Y2
Y3
Y4
A
B
C
D
1 2 3
4 5 6
7 8 9
✱ 0 #
INTERRUPT BUTTON
IR1
R1
100
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
D1
LED-RED
R16
100
IR0
IR1
KEYPAD DEBOUNCER/DECODER
U8
KBM
OSC
DA
OE
A
B
C
D
X1
X2
X3
X4
Y1
Y2
Y3
Y4
MM74C922
6
5
12
13
17
16
15
14
11
10
8
7
1
2
3
4
X1
X2
X3
Y1
Y2
Y3
Y4
PC0
PC1
PC2
PC3
IR0
U6:B
3
4
74LS04
C1
1nF
C2
1nF

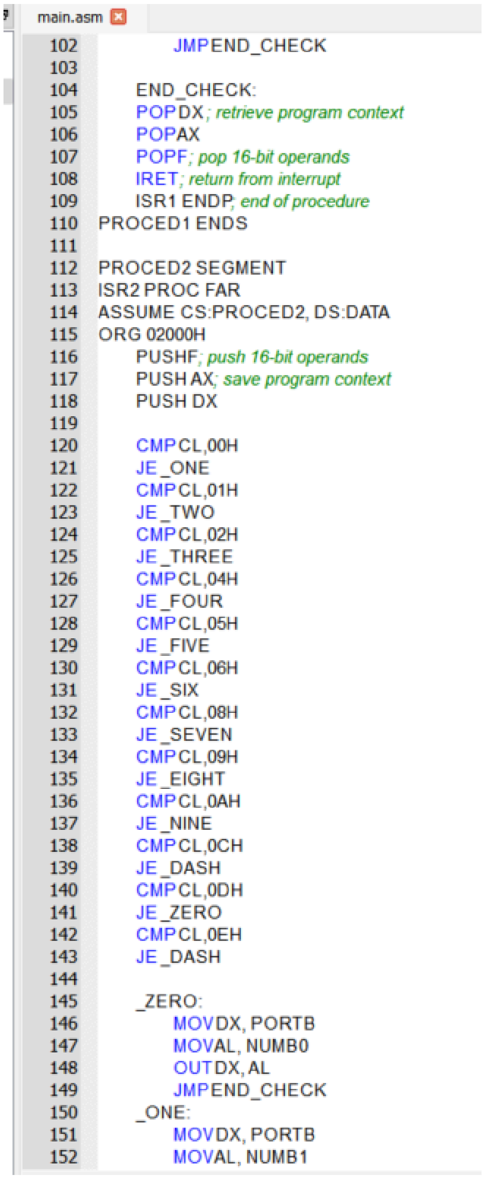

IMMEDIATE OUTPUT
INTERRUPT OUTPUT
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PB0
PB1
PB2
PB3
PB4
PB5
PB6
KEYPAD INPUT
X1
X2
X3
Y1
Y2
Y3
Y4
A
B
C
D
1
2
3
4
5
6
7
8
9
✱
0
#
INTERRUPT BUTTON
IR1
R1
100
PB7
PC0
PC1
PC2
PC3
D1
LED-RED
R16
100
IR0
IR1
KEYPAD DEBOUNCER/DECODER
U8
KBM
OSC
DA
OE
X1
X2
X3
X4
A
B
C
D
Y1
Y2
Y3
Y4
MM74C922
U6:B
74LS04
C1
1nF
C2
1nF



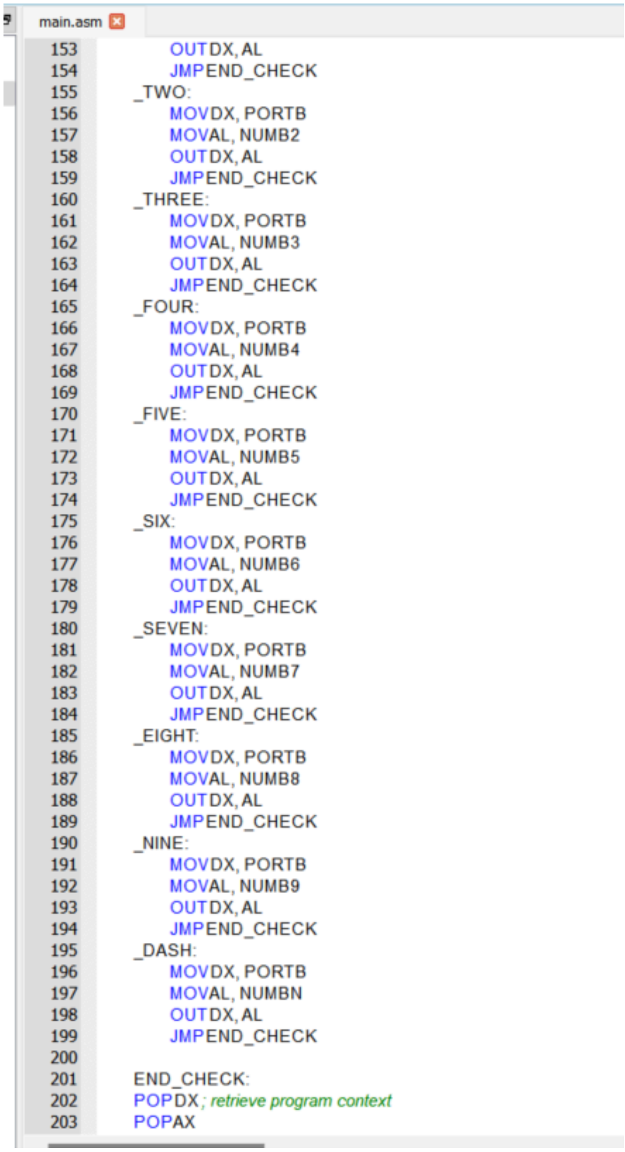
```
PROCED1 SEGMENT
ISR1 PROC FAR
ASSUME CS:PROCED1, DS:DATA
ORG 01000H
    PUSHF; push 16-bit operands
    PUSH AX; save program context
    PUSH DX

    MOVDX, PORTC
    INAL,DX
    ANDAL,0FH
    CMPAL,00H
    JE _ONE
    CMPAL,01H
    JE _TWO
    CMPAL,02H
    JE _THREE
    CMPAL,04H
    JE _FOUR
    CMPAL,05H
    JE _FIVE
    CMPAL,06H
    JE _SIX
    CMPAL,08H
    JE _SEVEN
    CMPAL,09H
    JE _EIGHT
    CMPAL,0AH
    JE _NINE
    CMPAL,0CH
    JE _DASH
    CMPAL,0DH
    JE _ZERO
    CMPAL,0EH
    JE _DASH

    _ZERO:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB0
        OUTDX, AL
        JMPEND_CHECK
    _ONE:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB1
        OUTDX, AL
        JMPEND_CHECK
    _TWO:
        MOVCL, AL
        MOVDX, PORTA
```



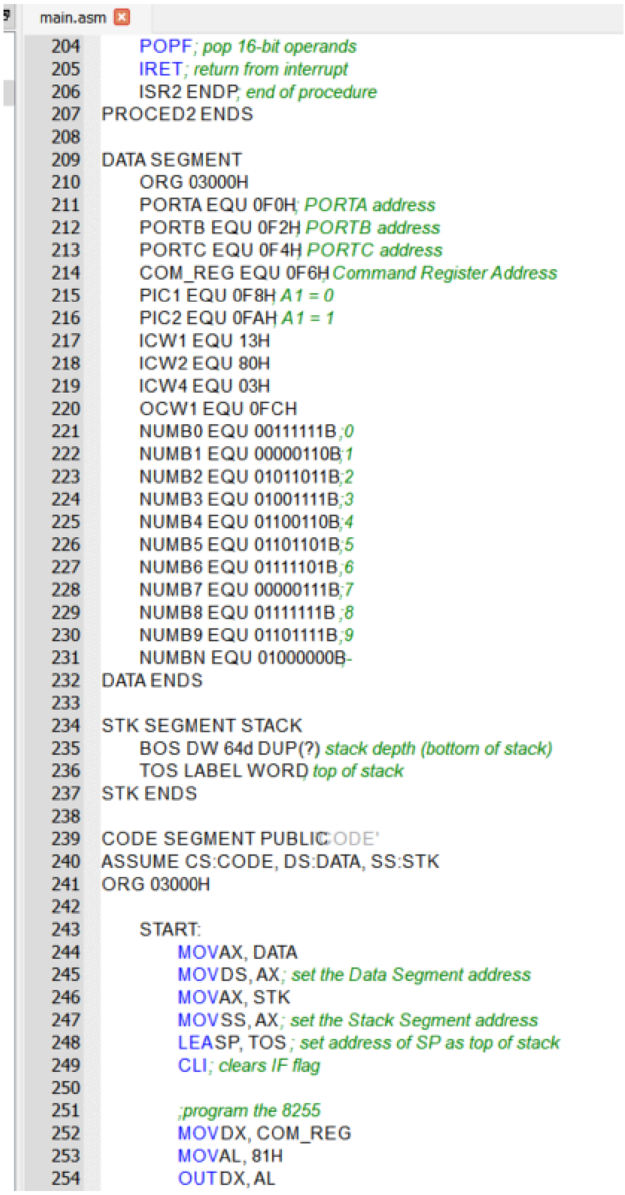
```
        MOVDX, PORTA
        MOVAL, NUMB2
        OUTDX, AL
        JMPEND_CHECK
    _THREE:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB3
        OUTDX, AL
        JMPEND_CHECK
    _FOUR:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB4
        OUTDX, AL
        JMPEND_CHECK
    _FIVE:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB5
        OUTDX, AL
        JMPEND_CHECK
    _SIX:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB6
        OUTDX, AL
        JMPEND_CHECK
    _SEVEN:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB7
        OUTDX, AL
        JMPEND_CHECK
    _EIGHT:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB8
        OUTDX, AL
        JMPEND_CHECK
    _NINE:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMB9
        OUTDX, AL
        JMPEND_CHECK
    _DASH:
        MOVCL, AL
        MOVDX, PORTA
        MOVAL, NUMBN
        OUTDX, AL
```



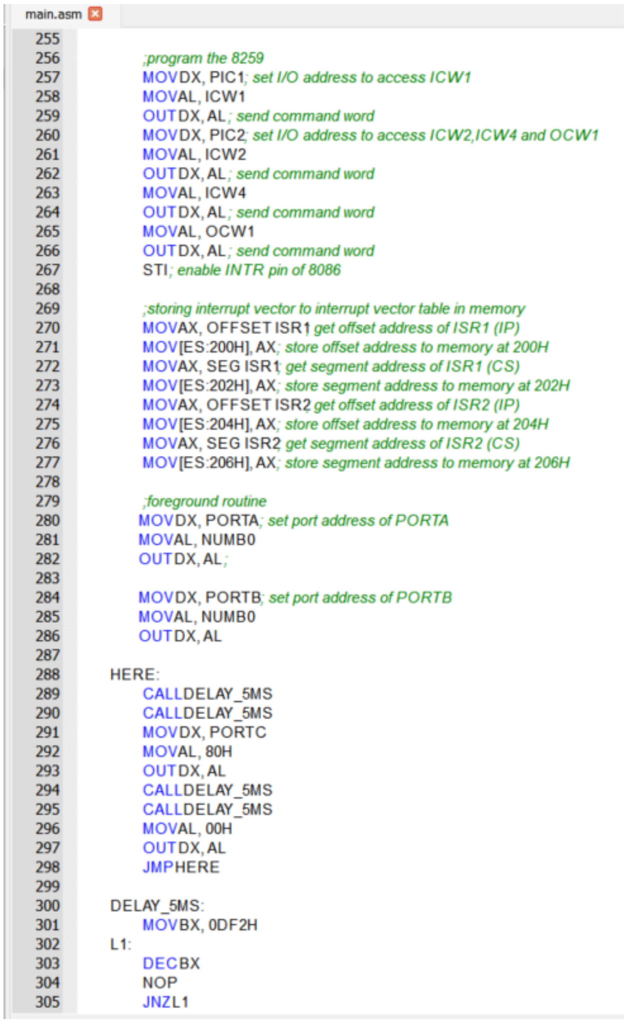
```
        JMPEND_CHECK

    END_CHECK:
    POPDX; retrieve program context
    POPAX
    POPF; pop 16-bit operands
    IRET; return from interrupt
    ISR1 ENDP; end of procedure
PROCED1 ENDS

PROCED2 SEGMENT
ISR2 PROC FAR
ASSUME CS:PROCED2, DS:DATA
ORG 02000H
    PUSHF; push 16-bit operands
    PUSH AX; save program context
    PUSH DX

    CMPCL,00H
    JE_ONE
    CMPCL,01H
    JE_TWO
    CMPCL,02H
    JE_THREE
    CMPCL,04H
    JE_FOUR
    CMPCL,05H
    JE_FIVE
    CMPCL,06H
    JE_SIX
    CMPCL,08H
    JE_SEVEN
    CMPCL,09H
    JE_EIGHT
    CMPCL,0AH
    JE_NINE
    CMPCL,0CH
    JE_DASH
    CMPCL,0DH
    JE_ZERO
    CMPCL,0EH
    JE_DASH

    _ZERO:
        MOVDX, PORTB
        MOVAL, NUMB0
        OUTDX, AL
        JMPEND_CHECK
    _ONE:
        MOVDX, PORTB
        MOVAL, NUMB1
```



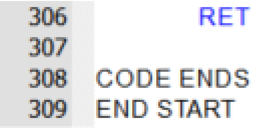
```
        OUTDX, AL
        JMPEND_CHECK
    _TWO:
        MOVDX, PORTB
        MOVAL, NUMB2
        OUTDX, AL
        JMPEND_CHECK
    _THREE:
        MOVDX, PORTB
        MOVAL, NUMB3
        OUTDX, AL
        JMPEND_CHECK
    _FOUR:
        MOVDX, PORTB
        MOVAL, NUMB4
        OUTDX, AL
        JMPEND_CHECK
    _FIVE:
        MOVDX, PORTB
        MOVAL, NUMB5
        OUTDX, AL
        JMPEND_CHECK
    _SIX:
        MOVDX, PORTB
        MOVAL, NUMB6
        OUTDX, AL
        JMPEND_CHECK
    _SEVEN:
        MOVDX, PORTB
        MOVAL, NUMB7
        OUTDX, AL
        JMPEND_CHECK
    _EIGHT:
        MOVDX, PORTB
        MOVAL, NUMB8
        OUTDX, AL
        JMPEND_CHECK
    _NINE:
        MOVDX, PORTB
        MOVAL, NUMB9
        OUTDX, AL
        JMPEND_CHECK
    _DASH:
        MOVDX, PORTB
        MOVAL, NUMBN
        OUTDX, AL
        JMPEND_CHECK

    END_CHECK:
    POPDX; retrieve program context
    POPAX
```



```
        POPF; pop 16-bit operands
        IRET; return from interrupt
        ISR2 ENDP; end of procedure
PROCED2 ENDS

DATA SEGMENT
        ORG 03000H
        PORTA EQU 0F0H; PORTA address
        PORTB EQU 0F2H; PORTB address
        PORTC EQU 0F4H; PORTC address
        COM_REG EQU 0F6H; Command Register Address
        PIC1 EQU 0F8H; A1 = 0
        PIC2 EQU 0FAH; A1 = 1
        ICW1 EQU 13H
        ICW2 EQU 80H
        ICW4 EQU 03H
        OCW1 EQU 0FCH
        NUMB0 EQU 00111111B;0
        NUMB1 EQU 00000110B;1
        NUMB2 EQU 01011011B;2
        NUMB3 EQU 01001111B;3
        NUMB4 EQU 01100110B;4
        NUMB5 EQU 01101101B;5
        NUMB6 EQU 01111101B;6
        NUMB7 EQU 00000111B;7
        NUMB8 EQU 01111111B ;8
        NUMB9 EQU 01101111B;9
        NUMBN EQU 01000000B;-
DATA ENDS

STK SEGMENT STACK
        BOS DW 64d DUP(?) ;stack depth (bottom of stack)
        TOS LABEL WORD; top of stack
STK ENDS

CODE SEGMENT PUBLIC 'CODE'
ASSUME CS:CODE, DS:DATA, SS:STK
ORG 03000H

        START:
                MOV AX, DATA
                MOV DS, AX; set the Data Segment address
                MOV AX, STK
                MOV SS, AX; set the Stack Segment address
                LEA SP, TOS ; set address of SP as top of stack
                CLI; clears IF flag

                ;program the 8255
                MOV DX, COM_REG
                MOV AL, 81H
                OUT DX, AL
```



```

                ;program the 8259
                MOV DX, PIC1; set I/O address to access ICW1
                MOV AL, ICW1
                OUT DX, AL; send command word
                MOV DX, PIC2; set I/O address to access ICW2,ICW4 and OCW1
                MOV AL, ICW2
                OUT DX, AL; send command word
                MOV AL, ICW4
                OUT DX, AL; send command word
                MOV AL, OCW1
                OUT DX, AL; send command word
                STI; enable INTR pin of 8086

                ;storing interrupt vector to interrupt vector table in memory
                MOV AX, OFFSET ISR1; get offset address of ISR1 (IP)
                MOV [ES:200H], AX; store offset address to memory at 200H
                MOV AX, SEG ISR1; get segment address of ISR1 (CS)
                MOV [ES:202H], AX; store segment address to memory at 202H
                MOV AX, OFFSET ISR2; get offset address of ISR2 (IP)
                MOV [ES:204H], AX; store offset address to memory at 204H
                MOV AX, SEG ISR2; get segment address of ISR2 (CS)
                MOV [ES:206H], AX; store segment address to memory at 206H

                ;foreground routine
               MOV DX, PORTA; set port address of PORTA
               MOV AL, NUMB0
               OUT DX, AL;

               MOV DX, PORTB; set port address of PORTB
               MOV AL, NUMB0
               OUT DX, AL

        HERE:
                CALL DELAY_5MS
                CALL DELAY_5MS
                MOV DX, PORTC
                MOV AL, 80H
                OUT DX, AL
                CALL DELAY_5MS
                CALL DELAY_5MS
                MOV AL, 00H
                OUT DX, AL
                JMP HERE

        DELAY_5MS:
                MOV BX, 0DF2H
        L1:
                DEC BX
                NOP
                JNZ L1
```



```
                RET

CODE ENDS
END START
```

Why do you think the LED is blinking steadily while other activities are going on?

In the 8086 system, the LED blinks steadily while other activities are going on because the interrupt service routine (ISR) for the LED is being triggered at regular intervals, independent of other ongoing processes. Hardware interrupts allow the CPU to momentarily pause its current task to execute the ISR, which toggles the LED state. Once the ISR completes, the CPU resumes the interrupted task seamlessly. This ability to manage multiple tasks concurrently via interrupts enables the LED to blink consistently, even while other processes are handled in the background.

What do you think is the ultimate advantage of using interrupts especially involving I/O devices?

The ultimate advantage of using interrupts, especially with I/O devices, is that they allow the 8086 to work efficiently by eliminating the need for constant polling. Instead of the 8086 actively waiting for an I/O device to complete an operation, it can continue executing other tasks until the device signals completion by triggering an interrupt. This reduces idle time, optimizes 8086 usage, and improves the overall responsiveness and performance of the system, particularly in environments requiring real-time or high-speed processing.

References