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

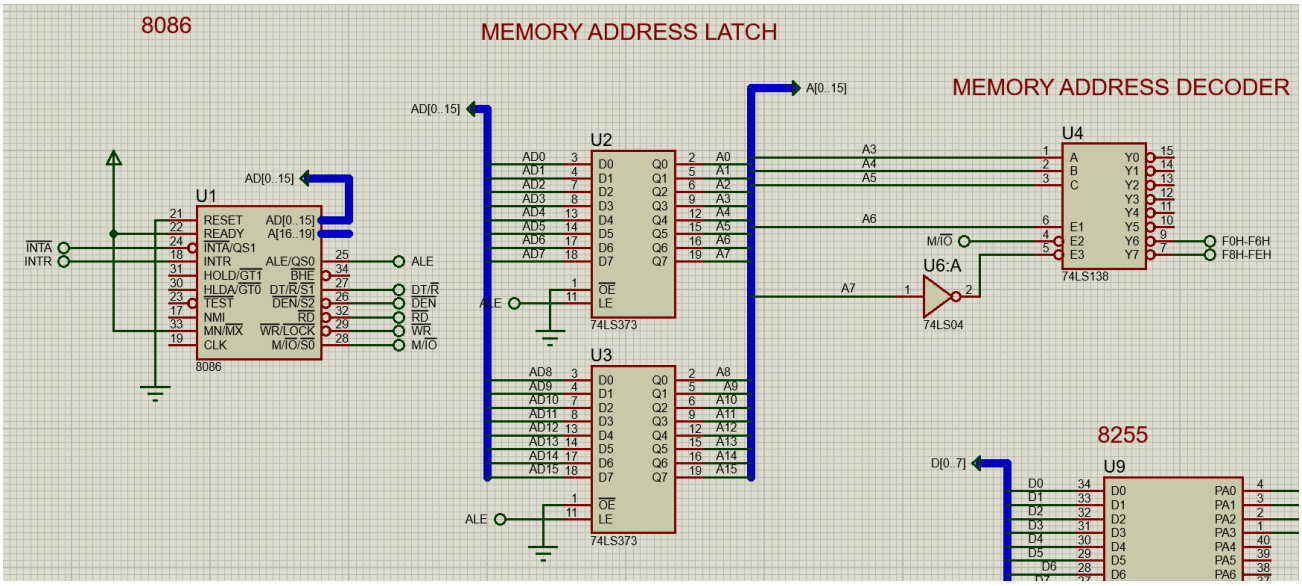
Activity #1

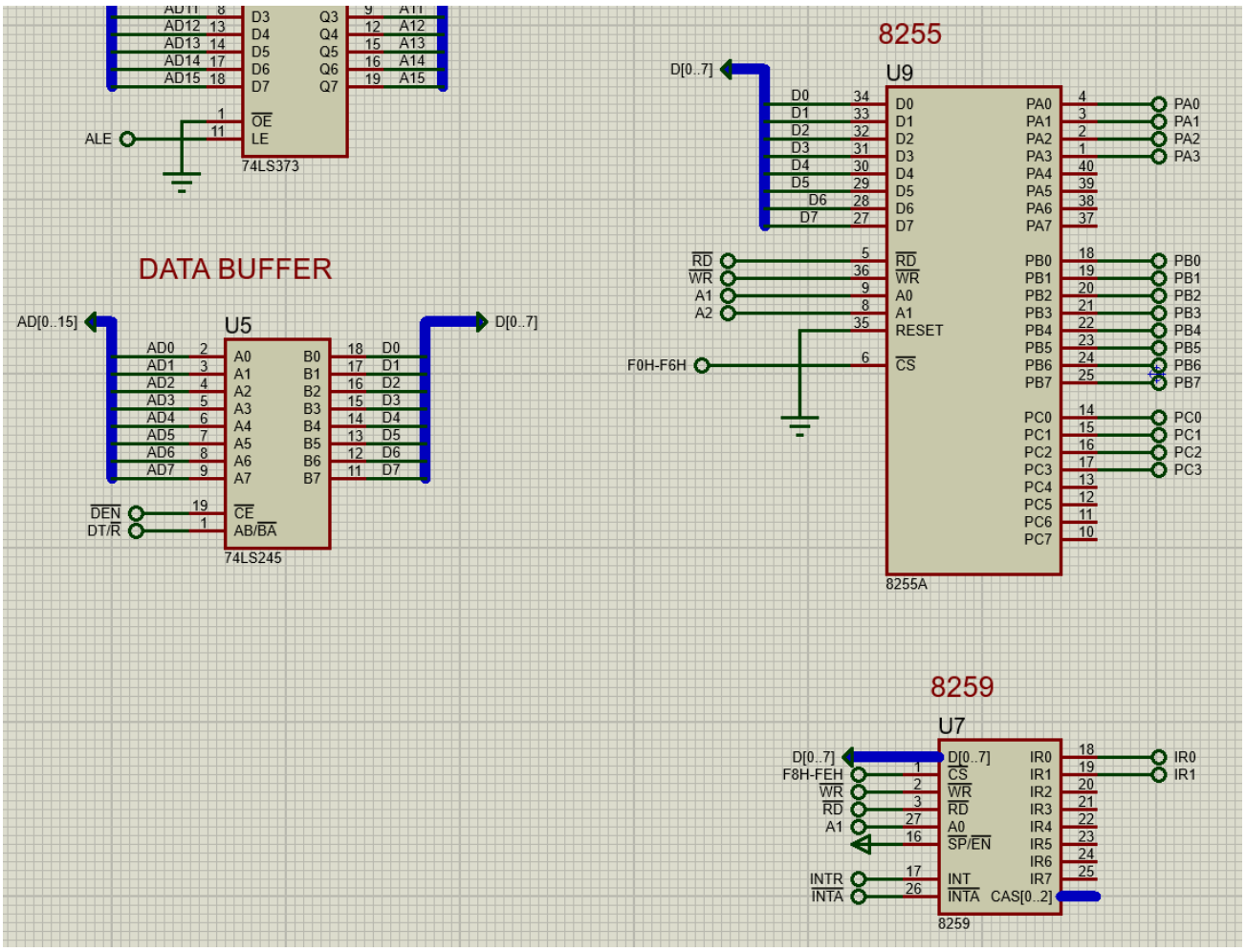
ICW4 needed, Single Mode, Call Address Interval of 8, Edge Triggered ICW1 = __13H_____

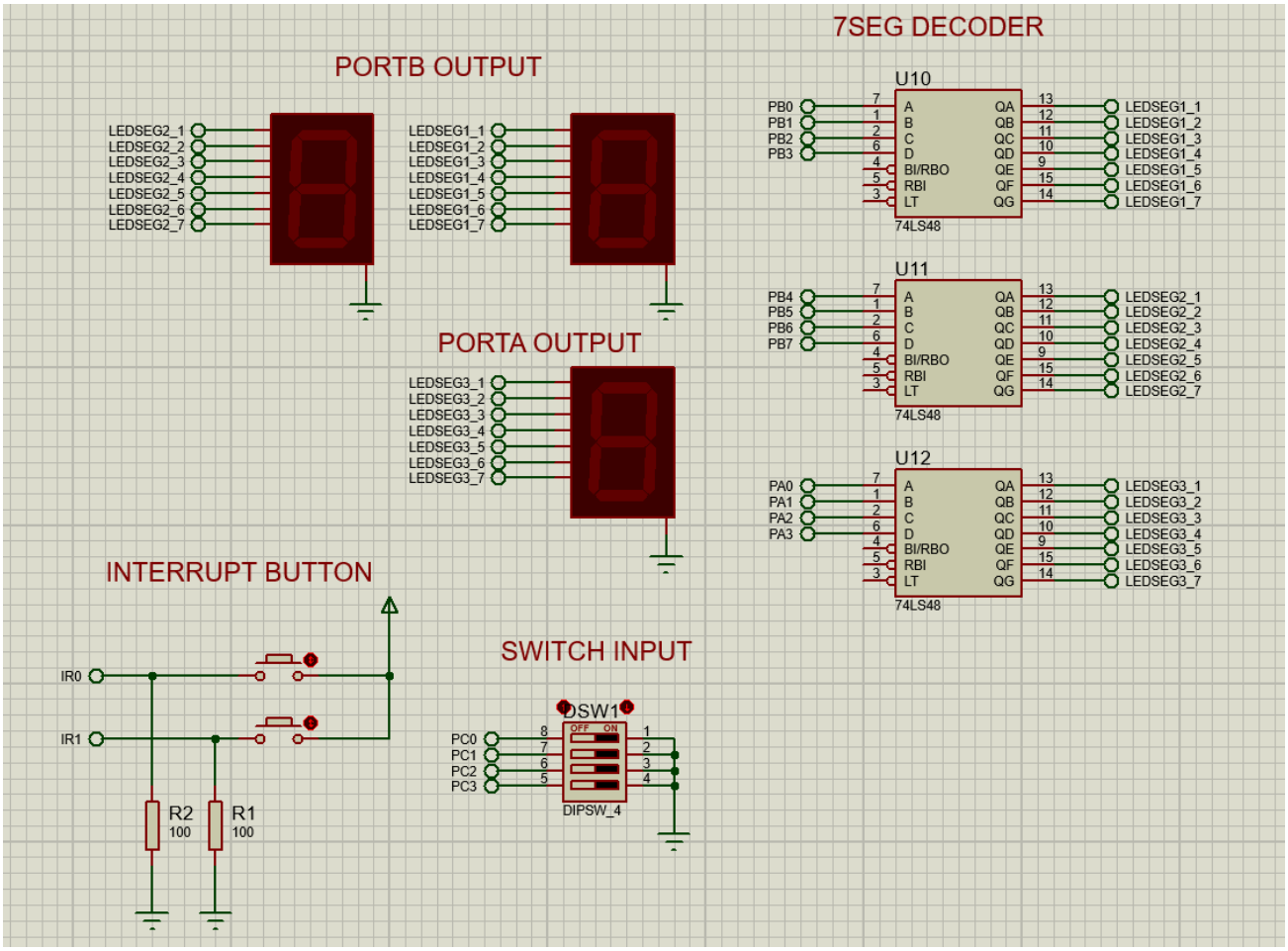
Interrupt Vector Address: 80H-87H ICW2 = ____80H____

8086 Mode, Auto EOI ICW4 = _____03H___

only IR0 and IR1 are unmasked OCW11 = ____0FCH____




AD12
AD13
AD14
AD15
D3
D4
D5
D6
D7
Q3
Q4
Q5
Q6
Q7
A12
A13
A14
A15
ALE
OE
LE
74LS373
DATA BUFFER
AD[0..15]
U5
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
D[0..7]
DEN
DT/R
CE
AB/BA
74LS245
8255
U9
D[0..7]
RD
WR
A1
A2
F0H-F6H
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
8255A
8259
U7
F8H-FEH
INTR
INTA
SP/EN
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
CAS[0..2]




```
1  PROCED1 SEGMENT
2  ISR1 PROC FAR
3  ASSUME CS:PROCED1, DS:DATA
4  ORG 01000H; write code within below starting at address 08000H
5    PUSHF; push 16-bit operands
6    PUSH AX; save program context
7    PUSH DX
8
9    ;<write the ISR code here>
10   MOV DX, PORTA
11   MOV AL, 09H
12   OUT DX, AL
13
14   POP DX ; retrieve program context
15   POP AX
16   POPF; pop 16-bit operands
17   IRET; return from interrupt
18 ISR1 ENDP; end of procedure
19 PROCED1 ENDS
```

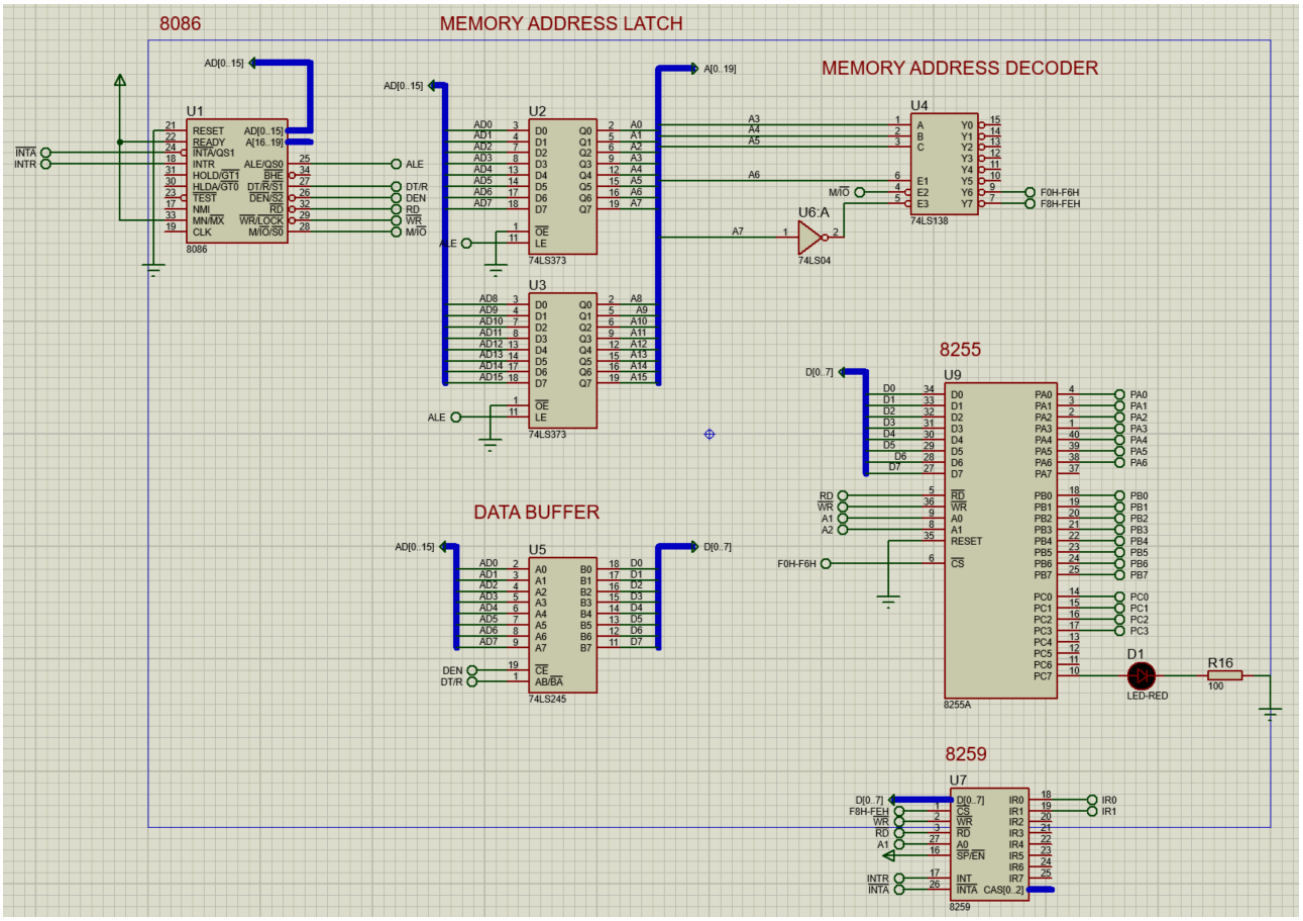
```
20
21  PROCED2 SEGMENT
22  ISR2 PROC FAR
23  ASSUME CS:PROCED2, DS:DATA
24  ORG 02000H; write code within below starting at address 09000H
25    PUSHF; push 16-bit operands
26    PUSH AX; save program context
27    PUSH DX
28
29    ;<write the ISR code here>
30    MOV DX, PORTA
31    MOV AL, 00H
32    OUT DX,AL
33
34    POP DX ; retrieve program context
35    POP AX
36    POPF; pop 16-bit operands
37    IRET; return from interrupt
38  ISR2 ENDP; end of procedure
39  PROCED2 ENDS
```

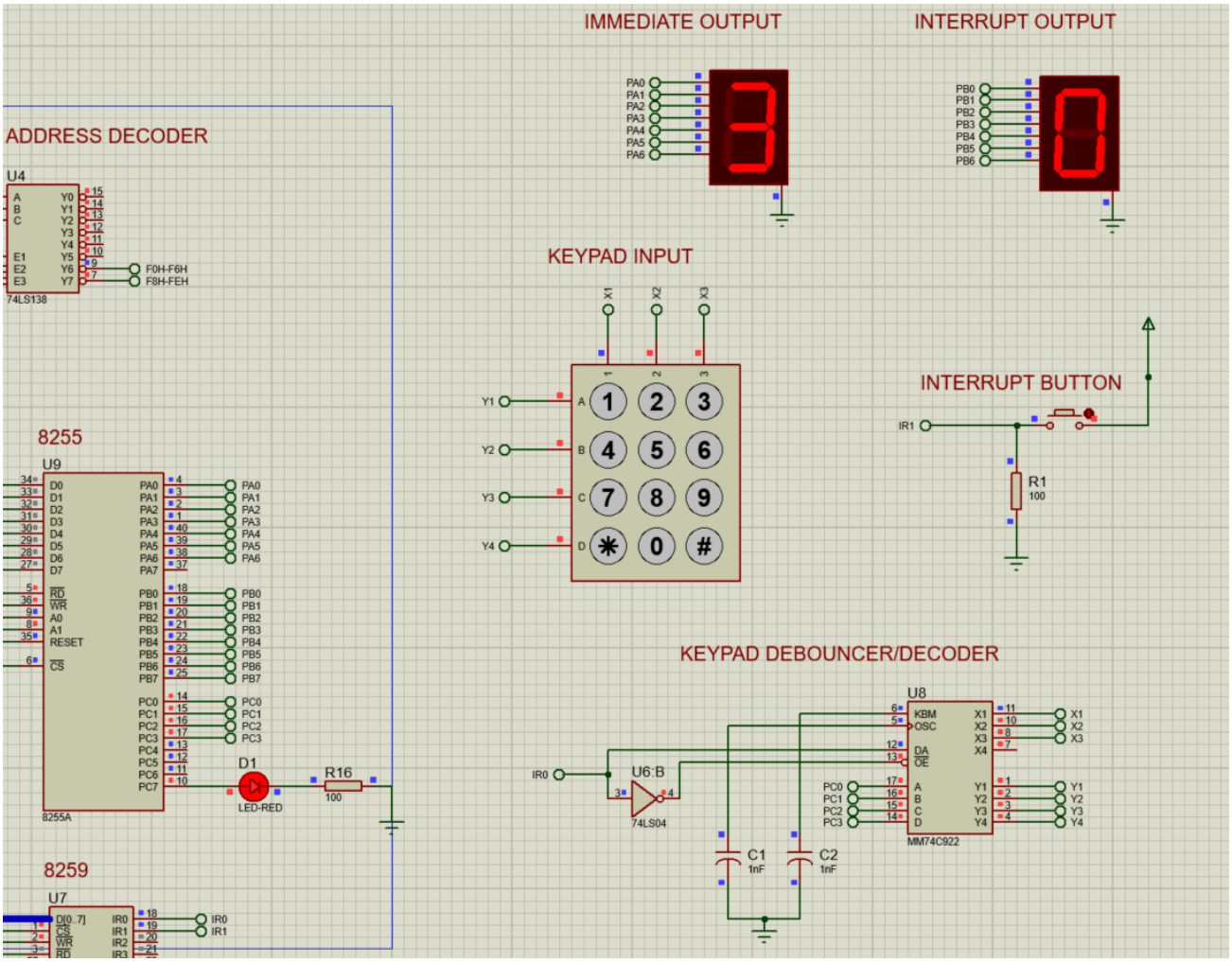
```
40
41  DATA SEGMENT
42    ORG 03000H
43    PORTA EQU 0F0H; PORTA address
44    PORTB EQU 0F2H; PORTB address
45    PORTC EQU 0F4H; PORTC address
46    COM_REG EQU 0F6H; Command Register Address
47    PIC1 EQU 0F8H; A1 = 0
48    PIC2 EQU 0FAH; A1 = 1
49    ICW1 EQU 13H; refer to #4
50    ICW2 EQU 80H; refer to #4
51    ICW4 EQU 03H; refer to #4
52    OCW1 EQU 0FCH; refer to #4
53  DATA ENDS
54
55  STK SEGMENT STACK
56    BOS DW 64d DUP(?); stack depth (bottom of stack)
57    TOS LABEL WORD; top of stack
58  STK ENDS
```

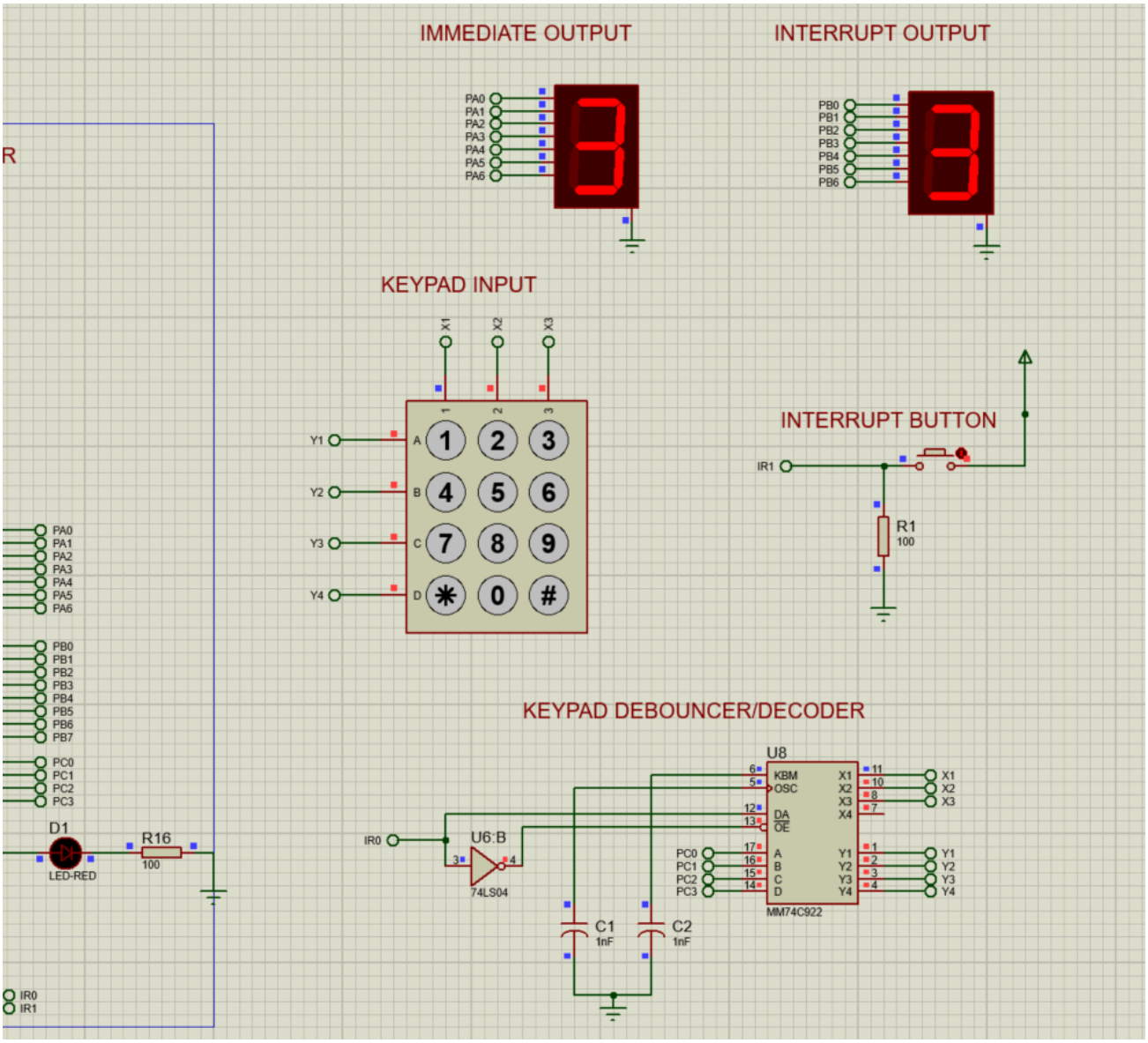
```
60  CODE SEGMENT PUBLIC 'CODE'
61    ASSUME CS:CODE, DS:DATA, SS:STK
62    ORG 08000H ; write code within below starting at address 0E000H
63    START:
64      MOV AX, DATA
65      MOV DS, AX ; set the Data Segment address
66      MOV AX, STK
67      MOV SS, AX ; set the Stack Segment address
68      LEA SP, TOS ; set address of SP as top of stack
69      CLI ; clears IF flag
70
71      ;program the 8255
72      MOV DX, COM_REG
73      MOV AL,89H
74      OUT DX, AL
75
76      ;program the 8259
77      MOV DX, PIC1 ; set I/O address to access ICW1
78      MOV AL, ICW1
79      OUT DX, AL ; send command word
80      MOV DX, PIC2 ; set I/O address to access ICW2,ICW4 and OCW1
81      MOV AL, ICW2
82      OUT DX, AL ; send command word
83      MOV AL, ICW4
84      OUT DX, AL ; send command word
85      MOV AL, OCW1
86      OUT DX, AL ; send command word
87      STI ; enable INTR pin of 8086
88
89      MOV AX, OFFSET ISR1 ; get offset address of ISR1 (IP)
90      MOV [ES:200H], AX ; store offset address to memory at 200H
91      MOV AX, SEG ISR1 ; get segment address of ISR1 (CS)
92      MOV [ES:202H], AX ; store segment address to memory at 202H
93      MOV AX, OFFSET ISR2 ; get offset address of ISR2 (IP)
94      MOV [ES:204H], AX ; store offset address to memory at 204H
95      MOV AX, SEG ISR2 ; get segment address of ISR2 (CS)
96      MOV [ES:206H], AX ; store segment address to memory at 206H
97
```

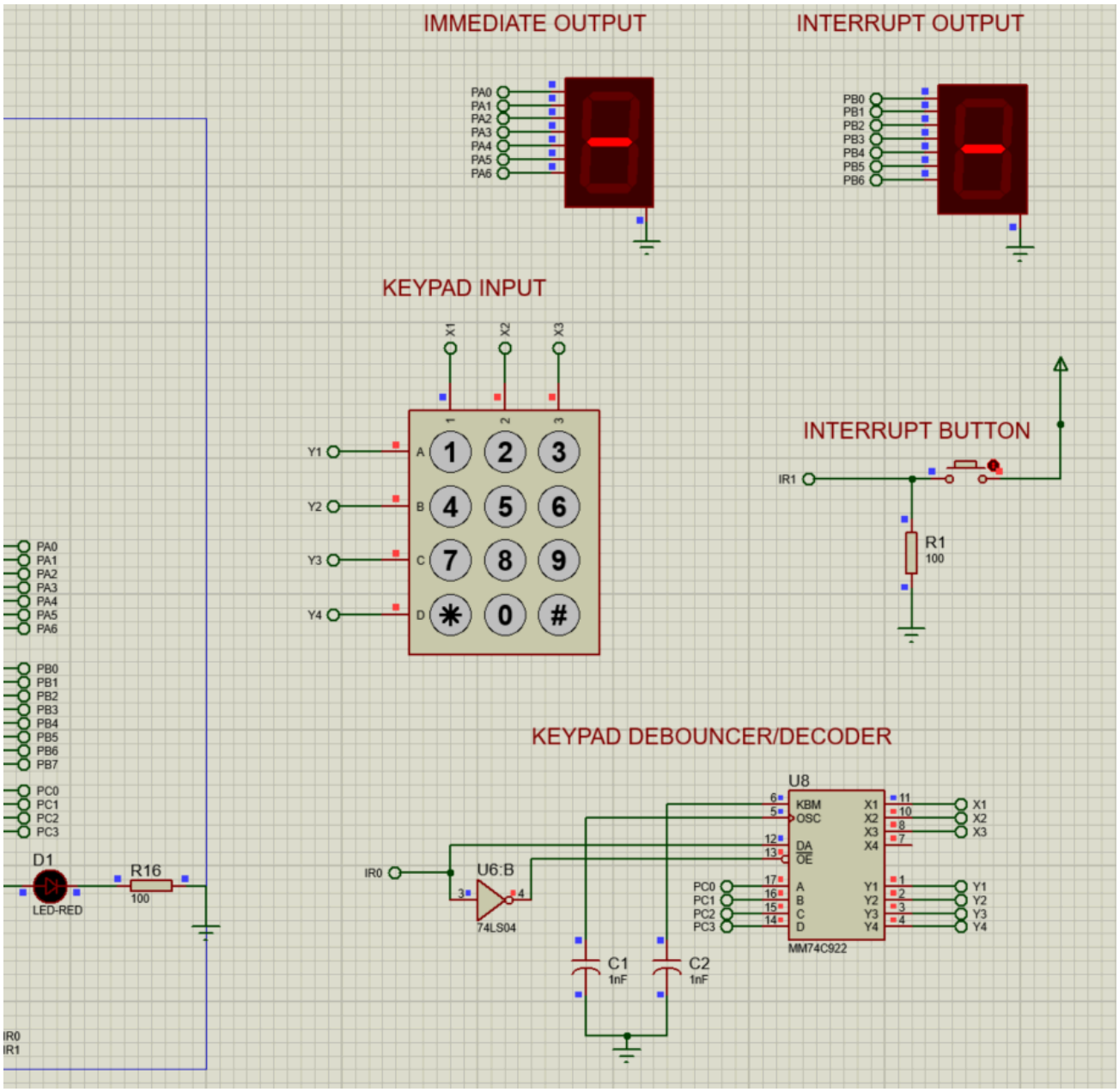
```
98    ;foreground routine
99    HERE:
100     ;<insert foreground routine code here>
101     _WAIT:
102     MOV DX, PORTC
103     IN AL, DX
104     AND AL, 0FH
105
106     CMP AL, 09H
107     JG GREATER
108
109     MOV DX, PORTB
110     OUT DX, AL
111     JMP _WAIT
112
113     GREATER:
114       MOV DX, PORTB
115       MOV AL, 00H
116       OUT DX, AL
117
118     JMP HERE
119
120   CODE ENDS
121   END START
```

Activity #2


8086
MEMORY ADDRESS LATCH
MEMORY ADDRESS DECODER
DATA BUFFER
8255
8259
U1
U2
U3
U4
U5
U6:A
U7
U9
74LS373
74LS138
74LS04
74LS245
8255A
LED-RED
R16
100
F0H-F6H
F8H-FEH


IMMEDIATE OUTPUT
INTERRUPT OUTPUT
ADDRESS DECODER
U4
74LS138
F0H-F6H
F8H-FEH
KEYPAD INPUT
INTERRUPT BUTTON
IR1
R1
100
8255
U9
8255A
D1
LED-RED
R16
100
KEYPAD DEBOUNCER/DECODER
U8
MM74C922
U6:B
74LS04
IR0
C1
1nF
C2
1nF
8259
U7


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

main.asm

```
1   PROCED1 SEGMENT
2   ISR1 PROC FAR
3   ASSUME CS:PROCED1, DS:DATA
4   ORG 01000H
5       PUSHF; push 16-bit operands
6       PUSH AX; save program context
7       PUSH DX
8
9       MOVDX, PORTC
10      INAL,DX
11      ANDAL,0FH
12      CMPAL,00H
13      JE _ONE
14      CMPAL,01H
15      JE _TWO
16      CMPAL,02H
17      JE _THREE
18      CMPAL,04H
19      JE _FOUR
20      CMPAL,05H
21      JE _FIVE
22      CMPAL,06H
23      JE _SIX
24      CMPAL,08H
25      JE _SEVEN
26      CMPAL,09H
27      JE _EIGHT
28      CMPAL,0AH
29      JE _NINE
30      CMPAL,0CH
31      JE _DASH
32      CMPAL,0DH
33      JE _ZERO
34      CMPAL,0EH
35      JE _DASH
36
37      _ZERO:
38          MOVCL, AL
39          MOVDX, PORTA
40          MOVAL, NUMB0
41          OUTDX, AL
42          JMPEND_CHECK
43      _ONE:
44          MOVCL, AL
45          MOVDX, PORTA
46          MOVAL, NUMB1
47          OUTDX, AL
48          JMPEND_CHECK
49      _TWO:
50          MOVCL, AL
51          MOVDX, PORTA
```

main.asm

```
51          MOVDX, PORTA
52          MOVAL, NUMB2
53          OUTDX, AL
54          JMPEND_CHECK
55      _THREE:
56          MOVCL, AL
57          MOVDX, PORTA
58          MOVAL, NUMB3
59          OUTDX, AL
60          JMPEND_CHECK
61      _FOUR:
62          MOVCL, AL
63          MOVDX, PORTA
64          MOVAL, NUMB4
65          OUTDX, AL
66          JMPEND_CHECK
67      _FIVE:
68          MOVCL, AL
69          MOVDX, PORTA
70          MOVAL, NUMB5
71          OUTDX, AL
72          JMPEND_CHECK
73      _SIX:
74          MOVCL, AL
75          MOVDX, PORTA
76          MOVAL, NUMB6
77          OUTDX, AL
78          JMPEND_CHECK
79      _SEVEN:
80          MOVCL, AL
81          MOVDX, PORTA
82          MOVAL, NUMB7
83          OUTDX, AL
84          JMPEND_CHECK
85      _EIGHT:
86          MOVCL, AL
87          MOVDX, PORTA
88          MOVAL, NUMB8
89          OUTDX, AL
90          JMPEND_CHECK
91      _NINE:
92          MOVCL, AL
93          MOVDX, PORTA
94          MOVAL, NUMB9
95          OUTDX, AL
96          JMPEND_CHECK
97      _DASH:
98          MOVCL, AL
99          MOVDX, PORTA
100         MOVAL, NUMBN
101         OUTDX, AL
```

main.asm

```
102             JMPEND_CHECK
103
104         END_CHECK:
105         POPDX; retrieve program context
106         POPAX
107         POPF; pop 16-bit operands
108         IRET; return from interrupt
109         ISR1 ENDP; end of procedure
110     PROCED1 ENDS
111
112     PROCED2 SEGMENT
113     ISR2 PROC FAR
114     ASSUME CS:PROCED2, DS:DATA
115     ORG 02000H
116         PUSHF; push 16-bit operands
117         PUSH AX; save program context
118         PUSH DX
119
120         CMPCL,00H
121         JE_ONE
122         CMPCL,01H
123         JE_TWO
124         CMPCL,02H
125         JE_THREE
126         CMPCL,04H
127         JE_FOUR
128         CMPCL,05H
129         JE_FIVE
130         CMPCL,06H
131         JE_SIX
132         CMPCL,08H
133         JE_SEVEN
134         CMPCL,09H
135         JE_EIGHT
136         CMPCL,0AH
137         JE_NINE
138         CMPCL,0CH
139         JE_DASH
140         CMPCL,0DH
141         JE_ZERO
142         CMPCL,0EH
143         JE_DASH
144
145         _ZERO:
146             MOVDX, PORTB
147             MOVAL, NUMB0
148             OUTDX, AL
149             JMPEND_CHECK
150         _ONE:
151             MOVDX, PORTB
152             MOVAL, NUMB1
```

main.asm

```
153             OUTDX, AL
154             JMPEND_CHECK
155         _TWO:
156             MOVDX, PORTB
157             MOVAL, NUMB2
158             OUTDX, AL
159             JMPEND_CHECK
160         _THREE:
161             MOVDX, PORTB
162             MOVAL, NUMB3
163             OUTDX, AL
164             JMPEND_CHECK
165         _FOUR:
166             MOVDX, PORTB
167             MOVAL, NUMB4
168             OUTDX, AL
169             JMPEND_CHECK
170         _FIVE:
171             MOVDX, PORTB
172             MOVAL, NUMB5
173             OUTDX, AL
174             JMPEND_CHECK
175         _SIX:
176             MOVDX, PORTB
177             MOVAL, NUMB6
178             OUTDX, AL
179             JMPEND_CHECK
180         _SEVEN:
181             MOVDX, PORTB
182             MOVAL, NUMB7
183             OUTDX, AL
184             JMPEND_CHECK
185         _EIGHT:
186             MOVDX, PORTB
187             MOVAL, NUMB8
188             OUTDX, AL
189             JMPEND_CHECK
190         _NINE:
191             MOVDX, PORTB
192             MOVAL, NUMB9
193             OUTDX, AL
194             JMPEND_CHECK
195         _DASH:
196             MOVDX, PORTB
197             MOVAL, NUMBN
198             OUTDX, AL
199             JMPEND_CHECK
200
201         END_CHECK:
202         POPDX; retrieve program context
203         POPAX
```

main.asm

```
204         POPF ; pop 16-bit operands
205         IRET ; return from interrupt
206         ISR2 ENDP ; end of procedure
207     PROCED2 ENDS
208
209     DATA SEGMENT
210         ORG 03000H
211         PORTA EQU 0F0H ; PORTA address
212         PORTB EQU 0F2H ; PORTB address
213         PORTC EQU 0F4H ; PORTC address
214         COM_REG EQU 0F6H ; Command Register Address
215         PIC1 EQU 0F8H ; A1 = 0
216         PIC2 EQU 0FAH ; A1 = 1
217         ICW1 EQU 13H
218         ICW2 EQU 80H
219         ICW4 EQU 03H
220         OCW1 EQU 0FCH
221         NUMB0 EQU 00111111B ;0
222         NUMB1 EQU 00000110B ;1
223         NUMB2 EQU 01011011B ;2
224         NUMB3 EQU 01001111B ;3
225         NUMB4 EQU 01100110B ;4
226         NUMB5 EQU 01101101B ;5
227         NUMB6 EQU 01111101B ;6
228         NUMB7 EQU 00000111B ;7
229         NUMB8 EQU 01111111B ;8
230         NUMB9 EQU 01101111B ;9
231         NUMBN EQU 01000000B-
232     DATA ENDS
233
234     STK SEGMENT STACK
235         BOS DW 64d DUP(?) ; stack depth (bottom of stack)
236         TOS LABEL WORD ; top of stack
237     STK ENDS
238
239     CODE SEGMENT PUBLIC 'CODE'
240     ASSUME CS:CODE, DS:DATA, SS:STK
241     ORG 03000H
242
243         START:
244             MOV AX, DATA
245             MOV DS, AX ; set the Data Segment address
246             MOV AX, STK
247             MOV SS, AX ; set the Stack Segment address
248             LEA SP, TOS ; set address of SP as top of stack
249             CLI ; clears IF flag
250
251             ;program the 8255
252             MOV DX, COM_REG
253             MOV AL, 81H
254             OUT DX, AL
```

main.asm

```
255
256             ;program the 8259
257             MOV DX, PIC1 ; set I/O address to access ICW1
258             MOV AL, ICW1
259             OUT DX, AL ; send command word
260             MOV DX, PIC2 ; set I/O address to access ICW2,ICW4 and OCW1
261             MOV AL, ICW2
262             OUT DX, AL ; send command word
263             MOV AL, ICW4
264             OUT DX, AL ; send command word
265             MOV AL, OCW1
266             OUT DX, AL ; send command word
267             STI ; enable INTR pin of 8086
268
269             ;storing interrupt vector to interrupt vector table in memory
270             MOV AX, OFFSET ISR1 ; get offset address of ISR1 (IP)
271             MOV [ES:200H], AX ; store offset address to memory at 200H
272             MOV AX, SEG ISR1 ; get segment address of ISR1 (CS)
273             MOV [ES:202H], AX ; store segment address to memory at 202H
274             MOV AX, OFFSET ISR2 ; get offset address of ISR2 (IP)
275             MOV [ES:204H], AX ; store offset address to memory at 204H
276             MOV AX, SEG ISR2 ; get segment address of ISR2 (CS)
277             MOV [ES:206H], AX ; store segment address to memory at 206H
278
279             ;foreground routine
280             MOV DX, PORTA ; set port address of PORTA
281             MOV AL, NUMB0
282             OUT DX, AL ;
283
284             MOV DX, PORTB ; set port address of PORTB
285             MOV AL, NUMB0
286             OUT DX, AL
287
288         HERE:
289             CALL DELAY_5MS
290             CALL DELAY_5MS
291             MOV DX, PORTC
292             MOV AL, 80H
293             OUT DX, AL
294             CALL DELAY_5MS
295             CALL DELAY_5MS
296             MOV AL, 00H
297             OUT DX, AL
298             JMP HERE
299
300         DELAY_5MS:
301             MOV BX, 0DF2H
302         L1:
303             DEC BX
304             NOP
305             JNZ L1
```

```
306             RET
307
308     CODE ENDS
309     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