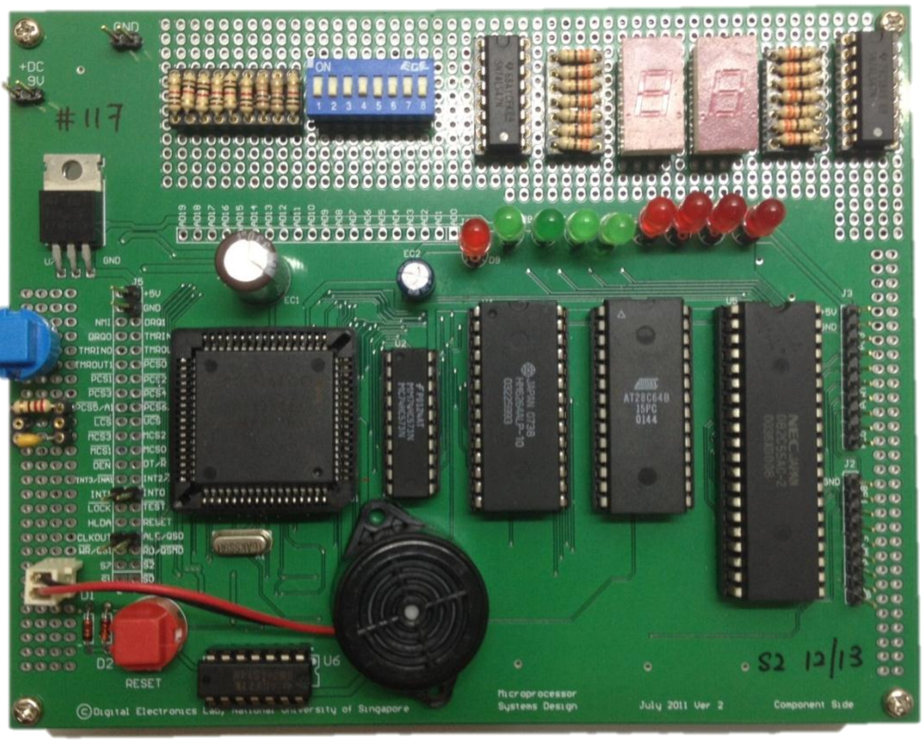


CG2007

Major Project 1 & 2

Yaadhav Raaj (A0081202Y)

Table of Contents

Table of Contents

Introduction

This is the project report for Major Project 1 and 2 for CG2007. It is essentially, the hardware section of the module, and introduces to a simplified computer system with a CPU, RAM, ROM and an IO chip. Concepts such as input, output, interrupts and delays will be used and tested in this system. Although the internals of the system have already been built for us on the circuit board, it was still necessary for us to understand the internals in order to select the high low states, and the correct address values for the various external components.

For this project, the first one involves a simple birthday display system, which displays a pre-set birthday value in order, with a 1s interval. The second one involves a lift control system, which allows the user to enter the floor he wishes to go to, and upon pressing a button, will trigger an interrupt which will cause the lift to move to that floor. Visual indication is also given through blinking LEDS, 7 segment displays and buzzers.

Problems and Solutions

Problem 1: Constant hardware failure

The board consistently had some form of instability or hardware failure. Sometimes, the CPU or the RAM chip might suddenly fail. Or, the solder from some connections becoming loose and hence causing errors. Hence, constant board failure would occur, and would require hardware level debugging

Solution 1: Hardware Debugging

The only solution was to do hardware debugging through a multimeter to check for disconnected connections, or to replace the faulty chips with a new one.

Problem 2: Delay value inaccurate

Assuming that the CLKOUT of the board is 8Mhz, and that we had calculated the clock cycles for various instructions, it would seem that the delay value should match the timing we see fit (1 second). However, the value was off by about 1M clock cycles.

Solution 2: Modify values

Hence, we had to choose a values such that an average of about 1 second was generated.

Problem 3: Button bouncing

The physical button tended to bounce, hence causing the interrupt to be triggered several times.

Solution 3: Capacitor and CLI

A capacitor based debouncing circuit was built, and once the interrupt routine was triggered, it disabled interrupts until the entire routine was complete.

Extra Features

Major Project 1:

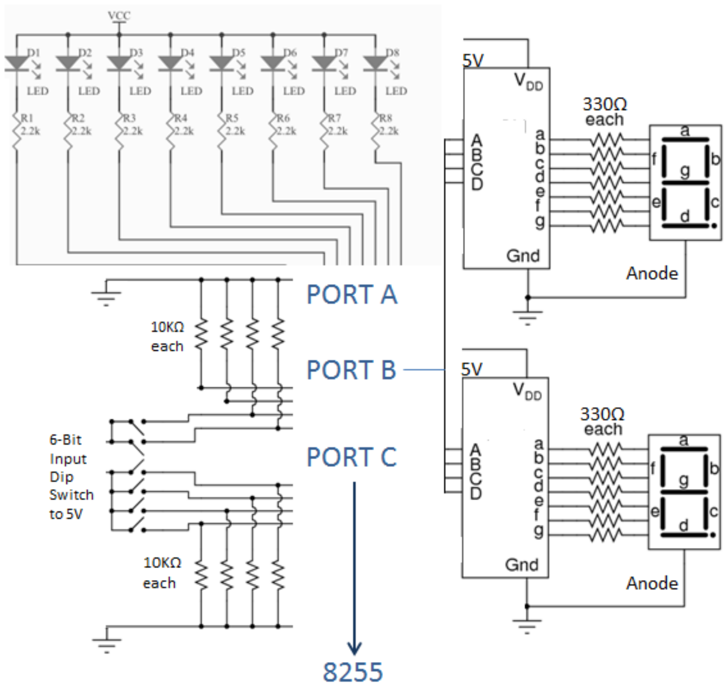
- Usage of 7 segment display

Major Project 2:

- Usage of buzzer when final story is reached
- Being able to count up to 64 story's and include a buzzer alert
- Instant updating of user selected input on the LEDS
- Debouncing of interrupt button and neat circuit layout

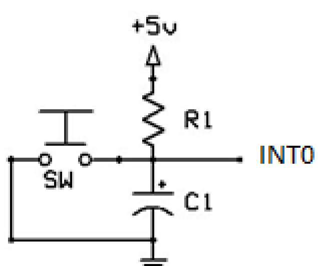
Circuit Diagram

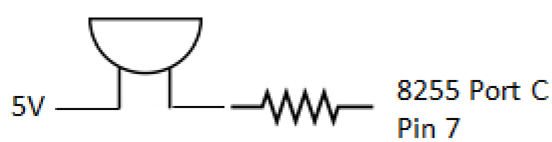
Connections to 8255 IO Chip



The above diagram shows the connections made to the 8255 IO Chip. The IO chip has the Control Word Register initially set-up as Port A (Output), Port B (Output) and Port C (Input). Port C input is later used for Major Project 2 Lift Control System. Port A drives the LED array which is output low as seen above, and Port B drives the 7-Segment driver, that drives the 7 segment anode chip. Port C supports up to a 6 bit input for 64 stories as later seen.

Additional Circuit Connections

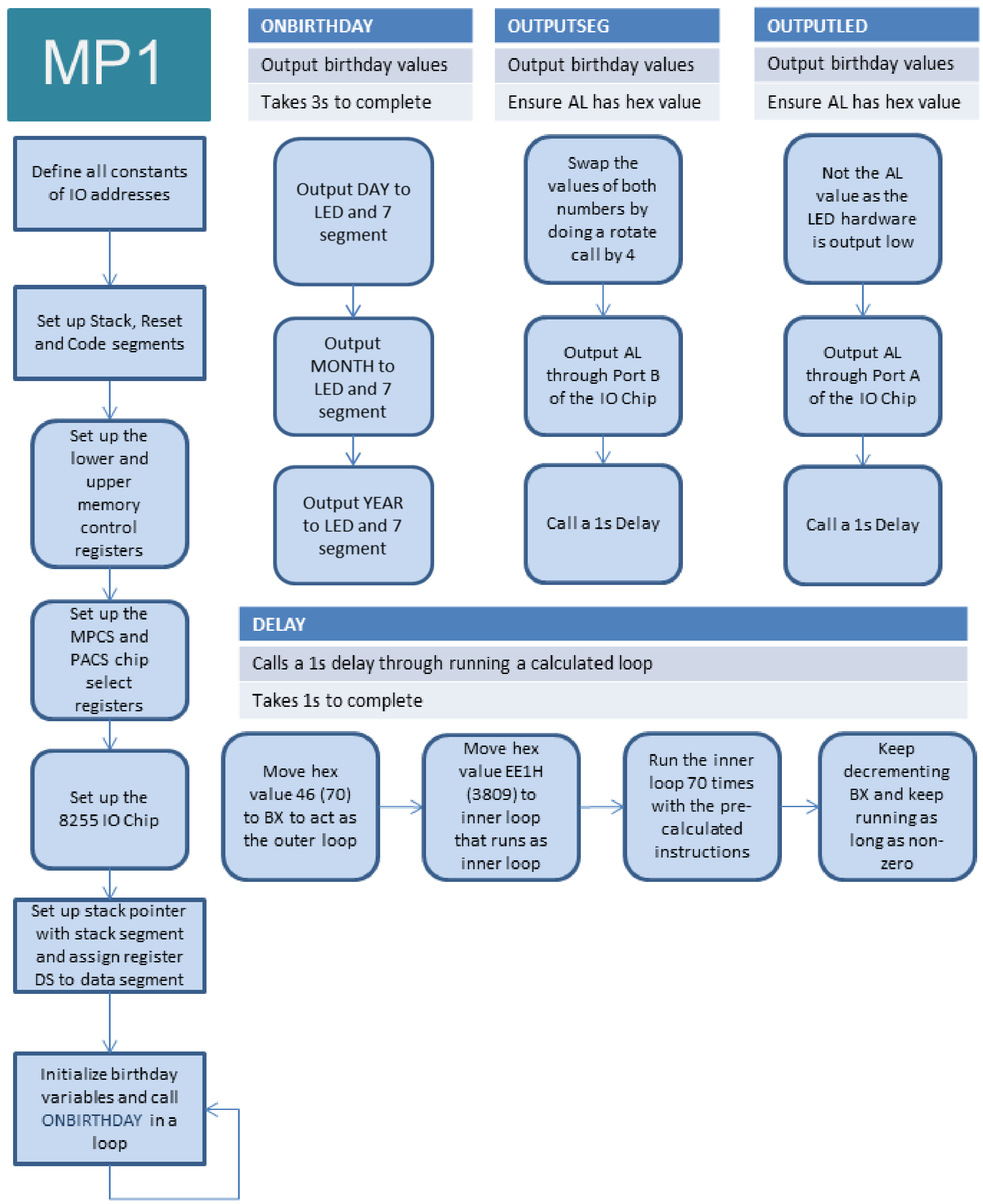




The first diagram shows the debouncing circuit for the Interrupt button used in Major Project 2. R1 is 1000 ohms and C1 is 0.1uF. The debouncing works by causing the capacitor to first charge up when pressed, and when released, causes the capacitor to discharge slowly rather than dropping to 0, hence preventing debouncing. The second diagram shows the buzzer, which is connected through a 330 ohm resistor, which is then connected to Pin 7 of Port C. The resistor limits the loudness of the buzzer.

Despite Port C being an input, when the buzzer is actually needed, it will be changed to output via assembly, so that 64 stories can still be counted to. This will be explained later. The buzzer rings when the user has reached the final story.

Flow Charts (MP1)



Basic MP1 Code

The code starts off with the definition of all constants which are used. These constants can be referred to at the Appendix. The constants refer to the various addresses that one can send byte data to through the OUT command. Then the stack, reset and code segments are set up, followed by the upper (ROM) and lower memory control (RAM) registers. Finally the IO chip is set up by setting the Control Word Register to Port A and B as output, and the stack pointer, stack segment and data segments are set up.

The code then enters a constant loop, where the procedure ONBIRTHDAY is called to display the birthday values with a 1s interval.

Variables

```
MOV DS:DAY,019H
MOV DS:MONTH,004H
MOV DS:YEAR,090H
```

As seen from the code in the appendix, the variables for day, month and year are initially reserved with a byte space in the Data Segment. Then, these values are initialized as 19, 4 and 90 in BCD Hex format. These are later to be displayed on both the LED and 7 segment displays.

Birthday Flashing (ONBIRTHDAY)

Simply move the variables into AL, then CALL OUTPUTSEG and OUTPUTLED.

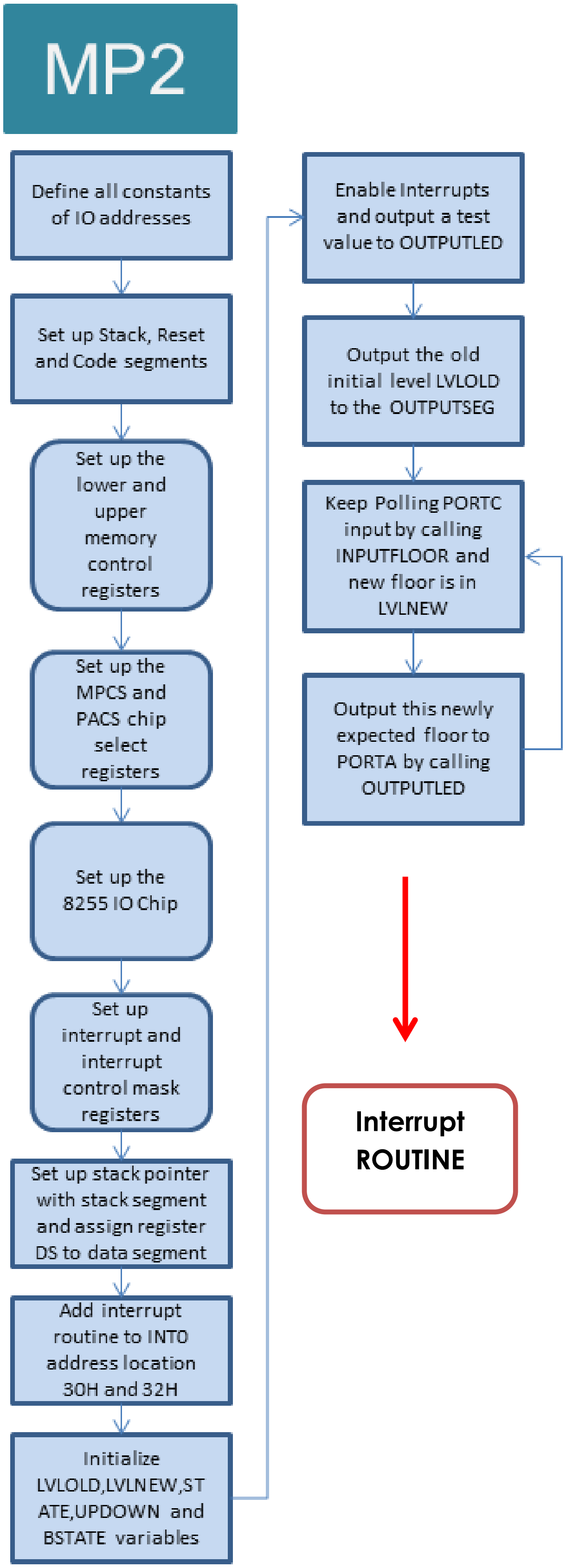
Output Functions

Simply output the BCD values to the LED and 7 segments. Due to the way the hardware was connected, it was required to NOT the AL value for the LED due to it being output low and rotate the AL value for the SEGMENT due to the connections.

DELAY

For DELAY an outer loop of count 70 was used with an inner loop of 3809 counts. The inner loop had instructions that amounted to about 21 clock cycles. This resulted in a 1 second delay.

Flow Charts (MP2)



Basic MP1 Code

As said earlier, all constants are defined, with the addition of Interrupt Control Registers (ICR), Interrupt Status Registers (ISR), Interrupt Priority and Interrupt Mask Registers (IMR), and End of Interrupt Registers (EOI).

Interrupt

In this case, the ICR for interrupt 0 is set up as I have wired the button that. It is set up with masking disabled, and a priority of 7 which doesn't matter, and is edge triggered. The bits can be seen in *Figure 1* in the appendix. IMR was set to disable INT0 mask as figured out from *Figure 2*.

The interrupt routine which is named "ROUTINE" BASE address is set to 30H and its OFFSET to 32H as based on the manual in *Figure 3*. Hence when INT0 is triggered, it will jump to the address specified at 30-32H. This is because INT0 is of type 12 where (12*4=30H) and (12*4+2=32H)

Variables

```
MOV DS:LVLOLD,1
MOV DS:LVLNEW,0
MOV DS:UPDOWN,0
MOV DS:BSTATE,0
```

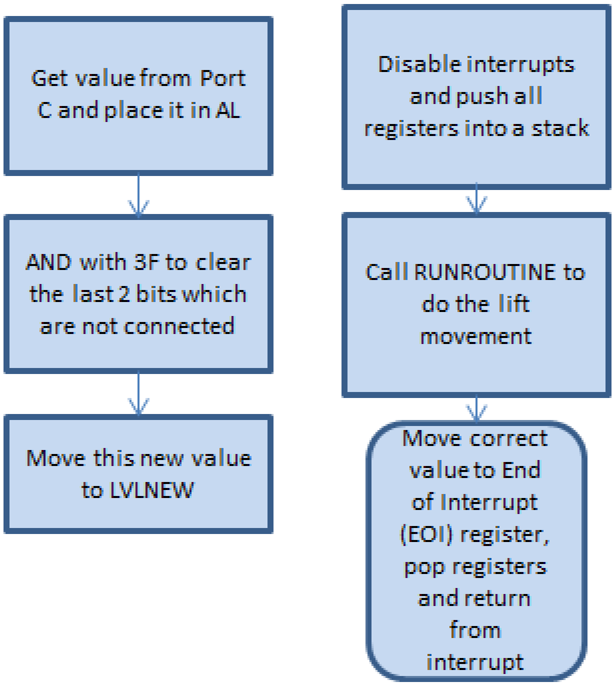
LVLOLD refers to the initial level set. LVLNEW refers to the level that the user wishes to go to. UPDOWN refers to whether the lift is currently going up or down, and BSTATE refers to the state of current blinking.

Polling of Input

When the lift is not moving, the level that the user wishes to go to is constantly polled (INPUTFLOOR), and then displayed on the LED port. It also updates the variable called LVLNEW. This happens until the user presses the button which starts the interrupt routine to move the lift.

INPUTFLOOR
Get user floor input
Update LVLNEW

ROUTINE
Interrupt Routine
NA



Input of Floor

When we are not running the lift movement routine, we are constantly polling to update the current floor. This procedure gets the 6-bit value from Port C and places it into the variable LVLNEW.

However, since bits 7 and 8 are not used, they are cleared anyway.

Routine

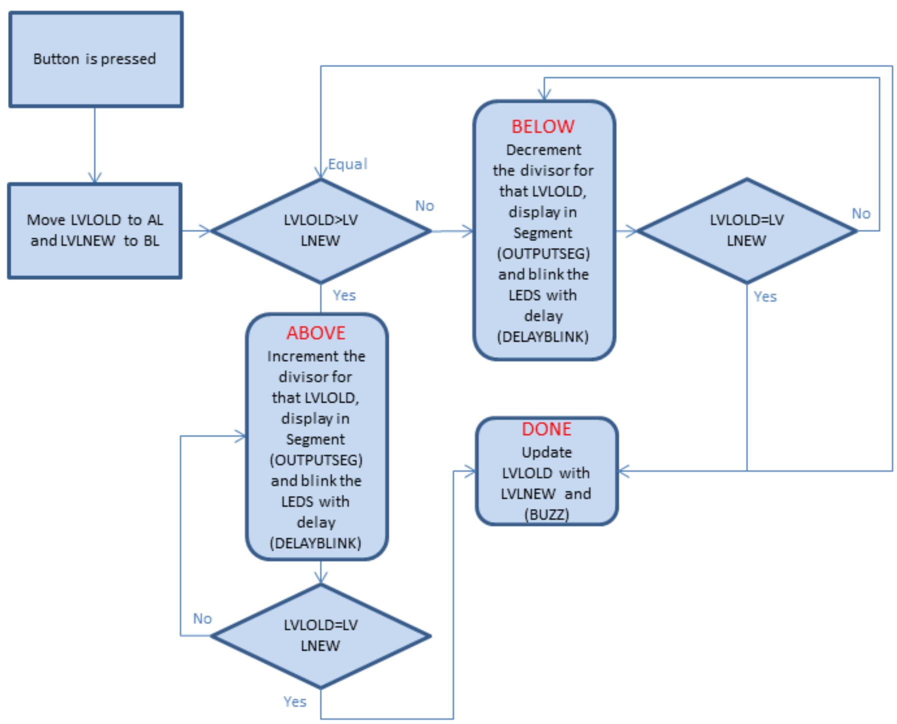
When the push button is pressed, the Instruction Pointer (IP) points to the address set at the Interrupt Vector, which points to "ROUTINE."

From here, a CLI command is used to disable interrupts temporarily, and (RUNROUTINE) is called to run the lift movement procedure.

Once this is complete, we can issue the bit 1 to INT0 whose in-Service bit is to be cleared. This can be seen from *Figure 4*.

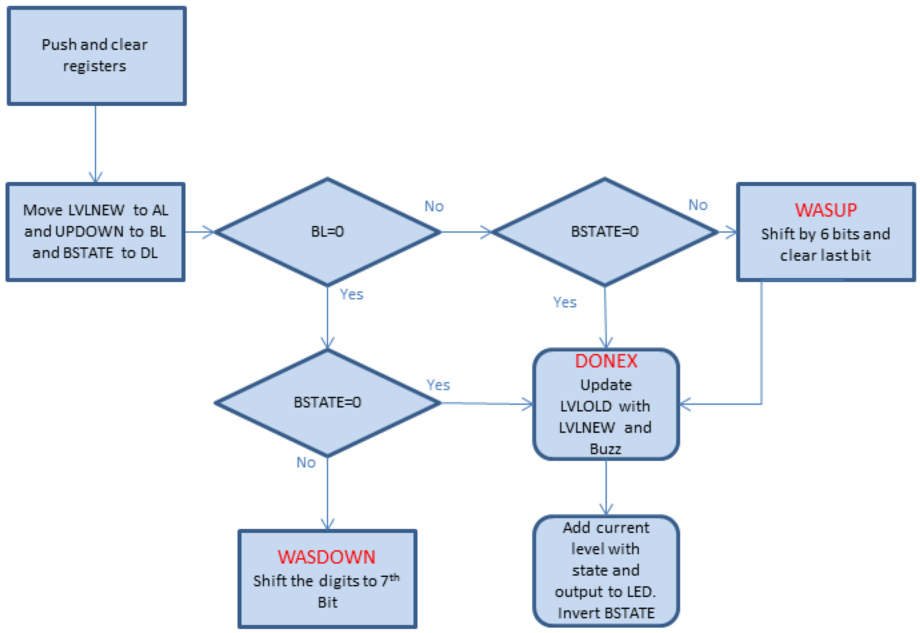
Finally, we return from the interrupt to the polling of a new user input again

Lift Routine (RUNROUTINE)



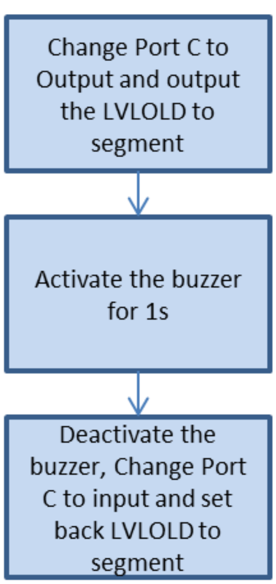
The above chart represents the lift movement routine (RUNROUTINE). Initially, LVLOLD and LVLNEW are compared, and if they are equal, it updates LVLOLD with the LVLNEW value and ends. If it is above or below, it runs the respective routine to decrement or increment, then display the correct floor. While it is rising or falling, (DELAYBLINK) is called to cause the 1s delay and the correct blinking of the Up or Down LED. When the final floor is reached, (BUZZ) is called to sound the buzzer.

LED Blinking (BLINKLED)

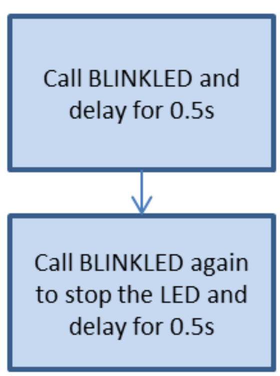


The above chart shows how the LED blinks in the delay function. Basically, the UPDOWN state is first checked, followed by BSTATE which indicates if the LED was blinking earlier. If it was not blinking earlier, make the correct LED turn on (7th for down and 6th for up). While the LED blinks as the lift moves, the user expected lift floor continues to be shown as it is added to the blinking value. A fair bit of bit shifting is used here.

BUZZ
Buzz output
NA



DELAYBLINK
1s Delay with LED Blink
NA



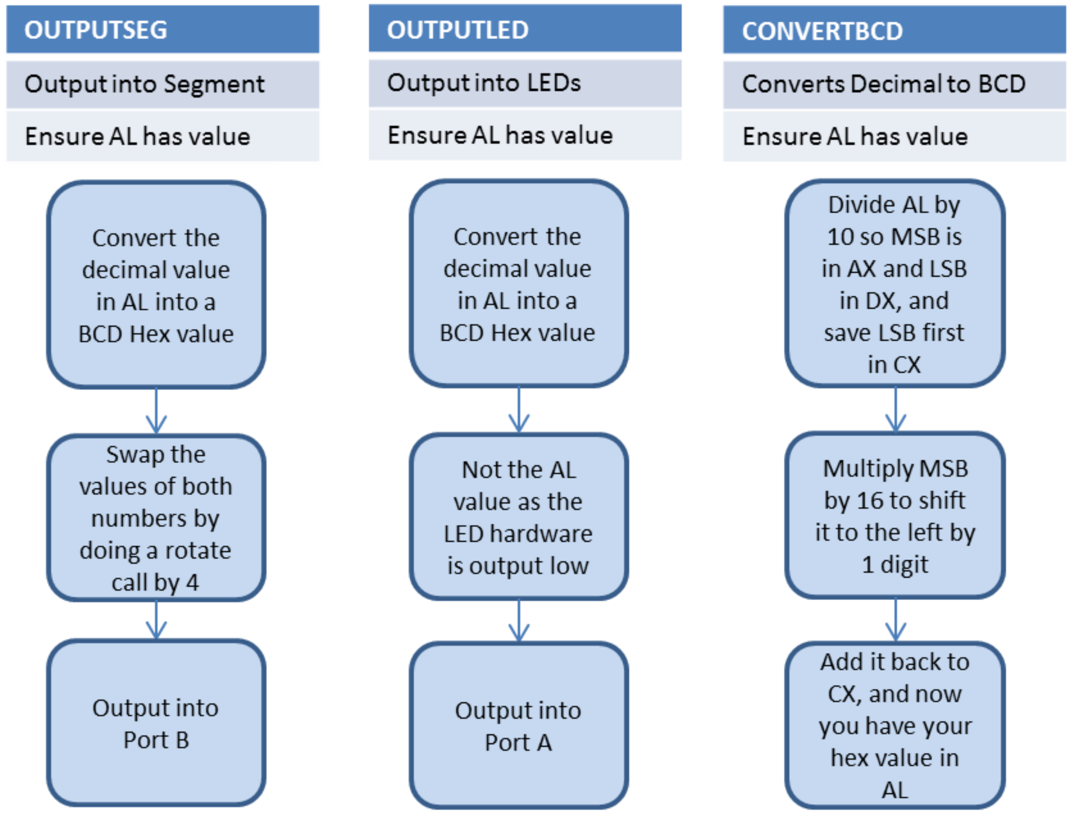
DELAYBLINK

This is the delay function that mixes (BLINKLED) to generate the 1s delay, and also to turn on and off the Up and Down LED while showing the user floor input.

BUZZ

Since we have no other output port to connect the buzzer to as we are using Port B for the bits to update the 7 segment display, we convert Port C temporarily into an output and sound the buzzer, then we convert it back to an input.

Output Procedures



Finally, we have some more procedures to easily control the outputs to the LED and 7 Segment. A decimal value of up to 63 can be passed into AL, then the procedure can be called to easily display it using the above calls.

Conclusion

In conclusion, we have learnt the basics of how an actual computer system works as a scaled-down model. We have learnt how the processor executes instructions stored in the ROM, places the variables it needs to modify in the RAM, and finally, interact with the 8255 IO Chip to create outputs to LEDS, 7 segment displays, input switches and buzzers.

I have learnt how all this interaction takes place on the address and data buses of the circuit, and how the various components in the CPU (Interrupt Control) and out of the CPU (8255 IO Chip) can be controlled by the address lines. I have learnt how the PACS register controls the chip select (PCS1) that in turn selects other components on the board (8255) to be controlled.

I have also learnt how to compile and run assembly code on the 8088 processor, and how the batch file actually automates and sets the various address locations of the various segments. I can now see how an operating system can actually run on this, by making use of both interrupts and timers, to constantly switch between tasks to run a system, and this is something I will be working on in future modules. This is something I look forward to.

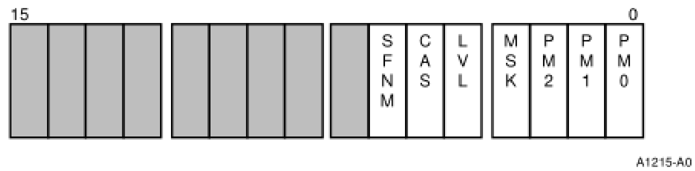
Appendix

Figure 1

Register Name: Interrupt Control Register (cascadable pins)

Register Mnemonic: I0CON, I1CON

Register Function: Control register for the cascadable external interrupt pins



Bit Mnemonic	Bit Name	Reset State	Function
SFNM	Special Fully Nested Mode	0	Set to enable special fully nested mode.
CAS	Cascade Mode	0	Set to enable cascade mode.
LVL	Level-trigger	0	Selects the interrupt triggering mode: 0 = edge triggering 1 = level triggering. The LVL bit **must be set** when external 8259As are cascaded into the Interrupt Control Unit.
MSK	Interrupt Mask	1	Clear to enable interrupts from this source.
PM2:0	Priority Level	111	Defines the priority level for this source.

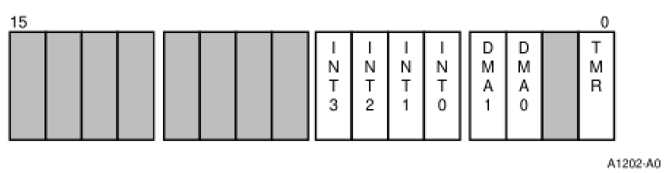
NOTE: Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 2

Register Name: Interrupt Mask Register

Register Mnemonic: IMASK

Register Function: Masks individual interrupt sources



Bit Mnemonic	Bit Name	Reset State	Function
INT3:0	External Interrupt Mask	0000 0	Set a bit to mask (disable) interrupt requests from the corresponding external interrupt pin.
DMA1:0	DMA Interrupt Mask	0	Set to mask (disable) interrupt requests from the corresponding DMA channel .
TMR	Timer Interrupt Mask	0	Set to mask (disable) interrupt requests from the timers.

NOTE: Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

Figure 3

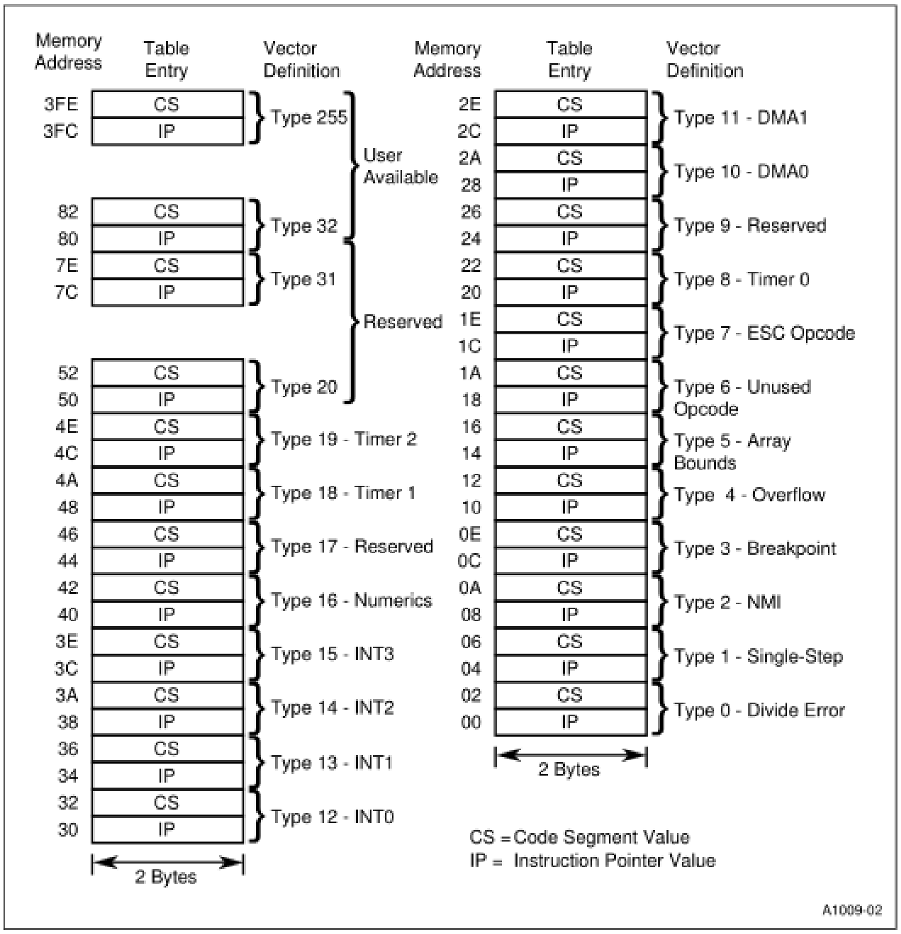
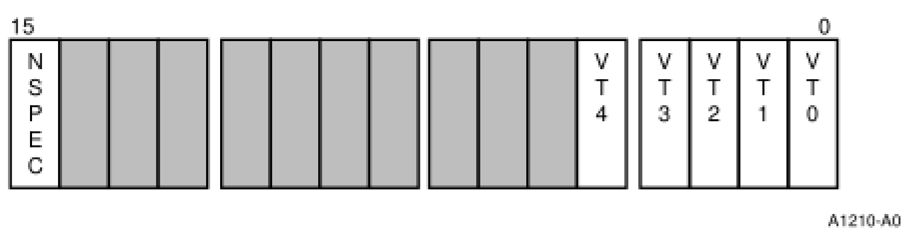


Figure 4

Register Name: End-of-Interrupt Register

Register Mnemonic: EOI

Register Function: Used to issue an EOI command



Bit Mnemonic	Bit Name	Reset State	Function
NSPEC	Nonspecific EOI	0	Set to issue a nonspecific EOI.
VT4:0	Interrupt Type	0 0000	Write with the interrupt type of the interrupt whose In-Service bit is to be cleared.

NOTE: Reserved register bits are shown with gray shading. Reserved bits must be written to a logic zero to ensure compatibility with future Intel products.

MP1 Code

```
$mod186
NAME EG0_COMP
; EE2007          Microprocessor Systems
; Sem 1                 AY2011-2012
;
; Author:        Mr. Niu Tianfang
; Address:       Department of Electrical Engineering
;                National University of Singapore
;                4 Engineering Dr 3
;                Singapore 117576.
; Date:          July 2011
;
; This file contains proprietory information and cannot be copied
; or distributed without prior permission from the author.
; --------------------------------------------------------------------


;IO Setup for 80C188
      UMCR              EQU 0FFA0H ; Upper Memory Control Register
      LMCR              EQU 0FFA2H ; Lower Memory control Register
      PACS              EQU 0FFA4H ; Peripheral Chip Select Base Address
      MPCS              EQU 0FFA8H ; MMCS and PCS Alter Control Register
      INT0              EQU 0FF38H ; INT0 Register
      PPI_8255_PORTA    EQU 080H ;port a
      PPI_8255_PORTB    EQU 081H ;port b
      PPI_8255_PORTC    EQU 082H ;port c
      PPI_8255_CWR      EQU 083H ;command word register

; STACK SEGMENT
STACK_SEG         SEGMENT
      db    128 DUP(?)
      tos   label word
STACK_SEG         ENDS


; DATA SEGMENT
DATA_SEG        SEGMENT
      DAY   DB ?
      MONTH       DB ?
      YEAR  DB ?
DATA_SEG        ENDS


; RESET SEGMENT
Reset_Seg   SEGMENT
    MOV DX, UMCR
    MOV AX, 03E07H
    OUT DX, AX
      JMP far PTR start
Reset_Seg  ends


; MESSAGE SEGMENT
MESSAGE_SEG     SEGMENT


MESSAGE_SEG     ENDS


;CODE SEGMENT
CODE_SEG        SEGMENT
```

```
PUBLIC      START
ASSUME  CS:CODE_SEG, DS:DATA_SEG, SS:STACK_SEG

START:

; Initialize MPCS to MAP peripheral to IO address
      MOV DX, MPCS
      MOV AX, 0083H
      OUT DX, AX

; PCSBA initial, set the parallel port start from 00H
      MOV DX, PACS
      MOV AX, 0003H ; Peripheral starting address 00H no READY, No Waits
      OUT DX, AX

; Initialize LMCS
    MOV DX, LMCR
    MOV AX, 01C4H  ; Starting address 1FFFH, 8K, No waits, last shoud be 5H for 1 waits
    OUT DX, AX

;initialize 8255 CWR (A to output)
      MOV DX, PPI_8255_CWR
      MOV AL, 080H
      OUT DX, AL

; Initialize Stack and data
      mov ax,STACK_SEG
      mov ss,ax
      mov sp,offset tos
      mov ax,DATA_SEG
      mov ds,ax

;initialize all general registers
      XOR AX,AX
      XOR BX,BX
      XOR CX,CX
      XOR DX,DX

;initialize variables
      MOV DS:DAY,019H
      MOV DS:MONTH,004H
      MOV DS:YEAR,090H

      ;mov ax,20h
;[RUN IN A LOOP]
RUNLOOP:
      CALL ONBIRTHDAY
      JMP RUNLOOP

;[STACKTEST]
STACKTEST PROC
      push ax
      xor ax,ax
      pop ax
      CALL OUTPUTLED
      RET
STACKTEST ENDP

;[BIRTHDAY FLASHING PROCEDURE]
ONBIRTHDAY PROC
      XOR AX,AX
      MOV AL, DAY
      CALL OUTPUTSEG
      CALL OUTPUTLED
```

```
        XOR AX,AX
        MOV AL, MONTH
        CALL OUTPUTSEG
        CALL OUTPUTLED

        XOR AX,AX
        MOV AL, YEAR
        CALL OUTPUTSEG
        CALL OUTPUTLED

        RET
ONBIRTHDAY ENDP

;[OUTPUT 7 SEGMENT TO PORTB]
OUTPUTSEG PROC
        ROR AL,4
        MOV DX, PPI_8255_PORTB
        OUT DX, AL
RET
OUTPUTSEG ENDP

;[OUTPUT LEDS TO PORTA]
OUTPUTLED PROC
        NOT AL
        MOV DX, PPI_8255_PORTA
        OUT DX, AL
        CALL DELAY
RET
OUTPUTLED ENDP


;[DELAY PROCEDURE FOR 1S]
DELAY PROC
        MOV BX, 0046H      ;delay 1 second for 8MHz processor
INNERLOOP:
        MOV CX, 0EE1H
        BACK:
        NOP                ;3 clocks
        LOOP BACK          ;18 clocks

        DEC BX                   ;3 clocks
        JNZ INNERLOOP      ;13 clocks
        RET
DELAY ENDP


CODE_SEG        ENDS
END
```

MP2 Code

```
$mod186
NAME EG0_COMP
; EE2007          Microprocessor Systems
; Sem 1                 AY2011-2012
;
; Author:       Mr. Niu Tianfang
; Address:      Department of Electrical Engineering
;               National University of Singapore
;               4 Engineering Dr 3
;               Singapore 117576.
; Date:         July 2011
```

```
;
; This file contains proprietory information and cannot be copied
; or distributed without prior permission from the author.
; -------------------------------------------------------------------


;IO Setup for 80C188
      UMCR               EQU 0FFA0H ; Upper Memory Control Register
      LMCR               EQU 0FFA2H ; Lower Memory control Register
      PACS               EQU 0FFA4H ; Peripheral Chip Select Base Address
      MPCS               EQU 0FFA8H ; MMCS and PCS Alter Control Register
      INT3_CTRL          EQU 0FF3EH ; Interrupt 3 Control Register
      INT2_CTRL          EQU 0FF3CH
      INT1_CTRL          EQU 0FF3AH
      INT0_CTRL          EQU 0FF38H
      TIMER_CTRL         EQU 0FF32H ; Timer Interrupt Control Register
      ISR                      EQU 0FF30H ; Interrupt Status Register
    EOI             EQU 0FF22H ; END OF INTERRUPT REGISTER
    IMKW            EQU 0FF28H ; Interrupt Mask
    IPMK            EQU 0FF2Ah ; Interrupt Priority Mask
      PPI_8255_PORTA     EQU 080H ;port a
      PPI_8255_PORTB     EQU 081H ;port b
      PPI_8255_PORTC     EQU 082H ;port c
      PPI_8255_CWR       EQU 083H ;command word register

; STACK SEGMENT
STACK_SEG         SEGMENT
      db    128 DUP(?)
      tos   label word
STACK_SEG         ENDS


; DATA SEGMENT
DATA_SEG        SEGMENT
      DAY   DB ?
      MONTH       DB ?
      YEAR  DB ?
      LVLOLD  DB ?
      LVLNEW      DB ?
      UPDOWN      DB ?
      BSTATE      DB ?
DATA_SEG        ENDS


; RESET SEGMENT
Reset_Seg   SEGMENT
    MOV DX, UMCR
    MOV AX, 03E07H
    OUT DX, AX
      JMP far PTR start
Reset_Seg  ends


; MESSAGE SEGMENT
MESSAGE_SEG     SEGMENT


MESSAGE_SEG     ENDS


;CODE SEGMENT
CODE_SEG        SEGMENT
PUBLIC      START
ASSUME  CS:CODE_SEG, DS:DATA_SEG, SS:STACK_SEG
```

```
START:

; Initialize MPCS to MAP peripheral to IO address
      MOV DX, MPCS
      MOV AX, 0083H
      OUT DX, AX

; PCSBA initial, set the parallel port start from 00H
      MOV DX, PACS
      MOV AX, 0003H ; Peripheral starting address 00H no READY, No Waits
      OUT DX, AX

; Initialize LMCS
    MOV DX, LMCR
    MOV AX, 01C4H  ; Starting address 1FFFH, 8K, No waits, last shoud be 5H for 1 waits
    OUT DX, AX

; Initialize Stack and data
      mov ax,STACK_SEG
      mov ss,ax
      mov sp,offset tos
      mov ax,DATA_SEG
      mov ds,ax

;initialize 8255 CWR (A to output)
      MOV DX, PPI_8255_CWR
      MOV AL, 089H
      OUT DX, AL

;initialize interrupts
      MOV DX, INT0_CTRL
      MOV AL, 00001111B
      OUT DX, AL
      XOR AX,AX
      MOV DX, IMKW
      MOV AL, 11101101B
      OUT DX, AL

; Initialize Extra segment to have interrupt vector
      MOV BX, 000H
      MOV ES, BX

;add to routine
      MOV BX, 30H
      MOV WORD PTR ES:[BX], OFFSET ROUTINE
      XOR BX,BX
      MOV BX, 32H
      MOV WORD PTR ES:[BX], SEG ROUTINE

;initialize all general registers
      XOR AX,AX
      XOR BX,BX
      XOR CX,CX
      XOR DX,DX

;initialize variables (including intial story)
      MOV DS:LVLOLD,1
      MOV DS:LVLNEW,0
      MOV DS:UPDOWN,0
      MOV DS:BSTATE,0
      STI

;*****************************************
;Polling Logic
;*****************************************
```

```
STARTING:
      STI
      XOR AX,AX
      MOV AL,10101010B
      CALL OUTPUTLED                        ;Output a generic value to LED
      XOR AX,AX
      MOV AL,LVLOLD                         ;Output initial level to Segment
      CALL OUTPUTSEG
POLLINPUT:
      STI
      CALL INPUTFLOOR                       ;Keep polling to get user input
      XOR AX,AX

      MOV AL,LVLNEW
      NOT AL                                      ;Display user input to LED
      MOV DX, PPI_8255_PORTA
      OUT DX, AL
      JMP POLLINPUT

;******************************************
;Interrupt Routine
;******************************************
ROUTINE:
      CLI                                         ;Disable interrupts and save registers
      PUSH AX
      PUSH BX
      PUSH CX
      PUSH DX

      CALL RUNROUTINE                       ;Call the Lift Routine

      MOV DX, EOI                           ;End of interrupt to Interrupt controller
    MOV AX, 12
    OUT DX, AL

      POP DX                                      ;Pop registers and return to polling
      POP CX
      POP BX
      POP AX
      IRET

;******************************************
;Lift Movement Procedure
;******************************************
RUNROUTINE PROC
      ;[LOGIC GOES HERE]
      ;press button
      ;current floor is stored in lvlold
      ;floor we want to go to is lvlnew
      ;check if lvlnew is greater than lvlold
      ;if greater, then lvlold needs to go up to reach lvlnew
      ;if smaller, then lvlold needs to go down to reach lvlnew
      ;once reached, updated lvlold to be lvlnew

      XOR AX,AX
      XOR BX,BX

      ;check if lvlnew is greater than lvlold
      MOV AL,DS:LVLOLD
      MOV BL,DS:LVLNEW
      CMP AL,BL
      JE DONE
      JA ABOVE
      JB BELOW
```

```
      ;if greater, then lvlold needs to go up to reach lvlnew
ABOVE:
      MOV DS:UPDOWN,1                          ;set updown flag to 1-UP
      CALL DELAYBLINK                          ;1s delay and up down blinking
      CALL OUTPUTSEG                           ;output lvlold
      DEC AL                                         ;decrement floor
      CMP BL,AL
      JNE ABOVE
      CMP BL,AL                                ;when floor reached, done
      JE DONE

      ;if smaller, then lvlold needs to go down to reach lvlnew
BELOW:
      MOV DS:UPDOWN,0                          ;set updown flag to 0-DOWN
      CALL DELAYBLINK                          ;1s delay and up down blinking
      CALL OUTPUTSEG                           ;output lvlold
      INC AL                                         ;increment floor
      CMP BL,AL
      JNE BELOW
      CMP BL,AL                                ;when floor reached, done
      JE DONE

      ;once reached, updated lvlold to be lvlnew
DONE:
      CALL DELAYBLINK                          ;delay band blink one more time
      CALL OUTPUTSEG                           ;output floor for the final time
      MOV DS:LVLOLD,AL                   ;update lvlold=lvlnew
      CALL BUZZ                                ;call the buzzer alert
      RET
RUNROUTINE ENDP

;*******************************************
;Update LVLNEW with user input
;*******************************************
INPUTFLOOR PROC
      XOR AX,AX
      MOV DX, PPI_8255_PORTC
      IN AL,DX
      AND AL,3FH
      MOV DS:LVLNEW,AL
      RET
INPUTFLOOR ENDP

;*******************************************
;Convert a decimal value in AX to BCD
;*******************************************
CONVERTBCD PROC
      PUSH DX                                        ;save and clear registers
      PUSH BX
      PUSH CX
      XOR DX,DX
      XOR BX,BX
      XOR CX,CX

      MOV BX,10
      DIV BX                                         ;now msb in ax, lsb in dx

      ADD CX,DX                                ;move in the lsb

      MOV BX,16
      MUL BX
      ADD CX,AX                                ;move in msb

      MOV AX,CX                                ;move back to ax the special value
```

```
       POP CX
       POP BX
       POP DX
       RET
CONVERTBCD ENDP

;*******************************************
;Output to 7 Segment from decimal value
;*******************************************
OUTPUTSEG PROC
       PUSH AX
       CALL CONVERTBCD
       ROR AL,4
       MOV DX, PPI_8255_PORTB
       OUT DX, AL
       POP AX
RET
OUTPUTSEG ENDP

;*******************************************
;Output to LED from decimal value
;*******************************************
OUTPUTLED PROC
       PUSH AX
       CALL CONVERTBCD
       NOT AL
       MOV DX, PPI_8255_PORTA
       OUT DX, AL
       POP AX
RET
OUTPUTLED ENDP

;*******************************************
;Mix Up or down blinking LED with input
;*******************************************
BLINKLED PROC
       ;[LOGIC GOES HERE]
       ;check UPDOWN state to see if going up or down
       ;if going up, then check the current blink state
       ;if blink state is 0, got to DONEX:
       ;if blink state is 1,then shift that bit to the 7th LED
       ;if going down, then check the current blink state
       ;if blink state is 0, got to DONEX:
       ;if blink state is 1,then shift that bit to the 6th LED
       ;now at DONEX: add the new user level with the bit and show

       PUSH AX
       PUSH BX
       PUSH CX
       PUSH DX
       XOR AX,AX
       XOR BX,BX
       XOR CX,CX
       XOR DX,DX

       ;AL-LVLNEW BL-UPDOWN DL-BSTATE
       MOV AL,DS:LVLNEW
       MOV BL,DS:UPDOWN
       MOV DL,DS:BSTATE
       CMP BL,0
       JZ WASDOWN
       JNZ WASUP

       ;In UP-mode
WASUP:
```

```
      CMP DL,0
      JZ DONEX
      MOV CL,DL
      SHL CL,6
      AND CL,01111111B
      JMP DONEX

      ;In DOWN-mode
WASDOWN:
      CMP DL,0
      JZ DONEX
      MOV CL,DL
      SHL CL,7
      JMP DONEX

      ;Add the user input to the blink state
DONEX:
      NOT DL
      MOV DS:BSTATE,DL

      ADD AL,CL
      NOT AL
      MOV DX, PPI_8255_PORTA
      OUT DX, AL

      POP DX
      POP CX
      POP BX
      POP AX
      RET
BLINKLED ENDP

;*******************************************
;Buzzer output on PORT C
;*******************************************
BUZZ PROC
      ;initialize 8255 CWR (C to output)
      MOV DX, PPI_8255_CWR
      MOV AL, 080H ;1000 0000 then 89H
      OUT DX, AL

      ;set back value to 7 seg
      MOV AL,DS:LVLOLD
      CALL OUTPUTSEG

      ;send 1 to buzz buzzer
      MOV AL,10000000B
      MOV DX,PPI_8255_PORTC
      OUT DX,AL

      ;allow a delay of buzzing for 1s
      MOV BX, 0023H
INNERLOOP3:
      MOV CX, 0EE1H
      BACK3:
      NOP
      LOOP BACK3
      DEC BX
      JNZ INNERLOOP3

      ;send 0 to stop buzzer
      MOV AL,00000000B
      MOV DX,PPI_8255_PORTC
      OUT DX,AL
```

```
      ;initialize 8255 CWR (C to input)
      MOV DX, PPI_8255_CWR
      MOV AL, 089H ;1000 0000 then 89H
      OUT DX, AL

      ;set back value to 7 seg
      MOV AL,DS:LVLOLD
      CALL OUTPUTSEG

      RET
BUZZ ENDP

;*******************************************
;DELAY + LED blinking
;*******************************************
DELAYBLINK PROC
      PUSH BX
      PUSH CX

      ;Turn on UPDOWN LED for 0.5s
      CALL BLINKLED
      MOV BX, 0023H
INNERLOOP:
      MOV CX, 0EE1H     ;70 loops
      BACK:
      NOP               ;3 clocks
      LOOP BACK         ;18 clocks

      DEC BX                  ;3 clocks
      JNZ INNERLOOP     ;13 clocks

      ;Turn off UPDOWN LED for 0.5s
      CALL BLINKLED
      MOV BX, 0023H
INNERLOOP2:
      MOV CX, 0EE1H     ;70 loops
      BACK2:
      NOP               ;3 clocks
      LOOP BACK2        ;18 clocks

      DEC BX                  ;3 clocks
      JNZ INNERLOOP2    ;13 clocks

      POP CX
      POP BX
      RET
DELAYBLINK ENDP

CODE_SEG        ENDS
END
```