|  |
| --- |
|  |
| CG2007 Project Report |
|  |
|  |
| **Song Yangyu**  **(A0077863N)** |
|  |

|  |
| --- |
|  |

Contents

[Introduction 3](#_Toc322062137)

[System Design and Development 3](#_Toc322062138)

[Functionality Achieved 3](#_Toc322062139)

[Major 1 3](#_Toc322062140)

[Major 2 3](#_Toc322062141)

[Hardware Design 3](#_Toc322062142)

[Major 1 Specific 3](#_Toc322062143)

[Major 2 Specific 4](#_Toc322062144)

[Code Logic 4](#_Toc322062145)

[Basic Configuration 4](#_Toc322062146)

[Major 1 -- Birthday Display 4](#_Toc322062147)

[Major 2 -- Elevator System 5](#_Toc322062148)

[Lesson & Things Learnt 6](#_Toc322062149)

[Soldering & Wire-wrapping Skills 6](#_Toc322062150)

[Read the User Manual and Datasheet 6](#_Toc322062151)

[Debugging Skills 7](#_Toc322062152)

[Conclusion 7](#_Toc322062153)

[Appendix: 8](#_Toc322062154)

[Photo of the board 8](#_Toc322062155)

[Code for Major Project 1 9](#_Toc322062156)

[Code for Major Project 2 11](#_Toc322062157)

# Introduction

The major project of 2007 includes two parts:

* Major 1: Display Birthday
* Major 2: Elevator System

This project would talking about the system design and development of the two projects, the problem encountered while doing the project and how I solved it, and the lesson learnt from this project.

The code for the two projects would be included in the end as appendix.

# System Design and Development

In this section, I'll first talk about the functionality achieved by my circuit + code, to give you a brief understanding of what my code can do; then I'll talk about the hardware design and software design for major1 and major2 separately.

## Functionality Achieved

### Major 1

Besides of what's required in the project manual (use LEDs in port A), I further used the 7 segment display to show the birthday information.

### Major 2

In this part, besides of what's required in the project manual, I further achieved these functionality:

* Support High Level -- user can set up to 64 level (instead of 16 levels as required)
* Support Two Modes of operation:
  + safe mode: same as what's required in the manual. When the lift is moving, user cannot change the target level unless they switch to fast mode.
  + fast mode: in this mode, if user specified a new level when the lift is moving, the lift would stop and immediately go to the new level specified by the user.

## Hardware Design

The hardware design follows the circuit schematic diagram as given in the IVLE. It mainly contains the following hardware:

* CPU -- Intel 80188 Microprocessor
* RAM -- UT6264B, 8K \* 8 Bit Low Power CMOS SRAM
* ROM -- AT28C64B, 8K \* 8 Bit EEPROM
* I/O -- uPD8255AC-2, Programmable Peripheral Interface

The ROM is mapped to the code segment, therefore its Enable Pin (E\) is connected to the UCS pin of the CPU; while the RAM is mapped to the data segment, and it contains the interrupt vector table (starting address 0), therefore its Chip Select Pin (CS\) is connected to the LCS pin of the CPU. We use PCS1 to select the 8255A chip, therefore it's Chip Select Pin(CS\) is connected to the PCS1.

### Major 1 Specific

In this part, I use PORTA LED to show the data in hexadecimal format, and use PORTC to show the data using two 7-segment displays.

***7 segment display***

Note that the 7 segment need 6 pin to display it's number properly. In order to save the total number of output pins used here, and also to simplify the code, I use a BCD to 7-segment decoder (DM74LS47) to do one level of decoding. After using this chip we only need 4 pints to display one number.

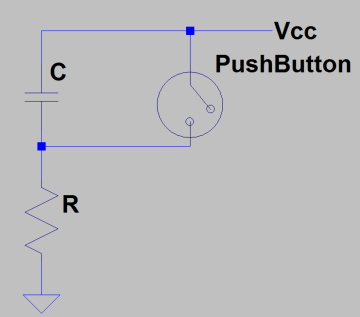
### Major 2 Specific

In the major 2 part, more hardware are added in. Here's the use of the ports:

PORTA: **output**. to show the move status using the left most two LEDs, and the target level using the rest of the 6 LEDs, in hexadecimal format.

PORTB: **input**. The leftmost bit is used for mode selection -- 1 for safe mode, 0 for fast mode; the right 6 LEDs are for selecting the target level.

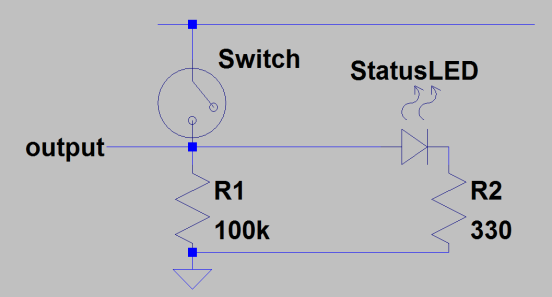
PORTC: **output**. to show the current level using two 7-segment displays

***Push Button De-bouncing Circuit***

If we simply connect the push button to the interrupt pin, when the push button is pressed, the bouncing would cause several level-change of in the interrupt pin, therefore would cause the interrupt several times.

Therefore I use the de-bouncing circuit on the right side to avoid this issue. The capacitor introduced would remove the AC components in the circuit.

***LEDs to show the status of Push Button***

I need an LED to display the current mode selected by the dip-switch. However, I cannot use any of the output port to show this information because the ports are fully occupied. Therefore, I use an LED to show the status of the Dip-Switch (circuit as shown on the).

Note that although I have an LED circuit to pull down the output, I'll still need the register R1 to further pull down the voltage when the switch is open, because the LED has some inner resistance which would possibly fail to pull down the voltage.

## Code Logic

I'll explain the basic logic of the code in this section. The basic configuration contains the configuration information common to the two projects -- what's the setting and why it's set like that; then I'll explain the project-specific setting in each subsection. In the end of each section, I'll give a "highlights" part to show what make my code different from others.

### Basic Configuration

Before the main logic, I'll do necessary configurations first. Besides the code given in the template, I need to set the CWR properly.

CLR is set to 82H, to set it in 0 mode, use Group A as output, Group B as input, and Group C as output.

***Highlights: The Use of Magic Variables***

For the ease of reading and writing the code, I use magic variables as much possible: the address of three ports, EOI register, register for different interrupt and timer, are all assigned with properly-named magic variable to store its address. This is a good practice of **Software Engineering**.

### Major 1 -- Birthday Display

The code for this part is very simple: just display the day/month/year information, and wait for one second.

***calculation of the delay***

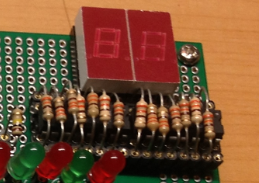
Notice that the CPU frequency is 8M. Therefore, I use two levels of for loop to do a delay of one second. The inner loop would run most of the time, the inner loop contains:

again: nop; loop again

the nop command would consume 3 clock cycle, and the loop command would consume 16 clock cycles when there's a jump (most of the time) , and 6 clock cycles when there's no jump (rarely happens). Therefore, the overall clock cycles consumed by the inner loop is expected to be 19.

Therefore, if we define the outer loop to be the number of 1MHz, then the inner loop is need to consume 1MHz. Therefore, loop number for the inner loop should be:

# loops = 1e6 / 19 = 52631

Therefore, the inner loop should execute 52631 times, and the outer loop should execute 8 times.

***Highlights: Compact Circuit Design***

In order to save some space (found unnecessary later), I placed the register on top of the chip. This is not necessary for this project because there's not much need for space. But this design would be useful for the space-sensitive projects.

### Major 2 -- Elevator System

The elevator contains 3 parts: read input, blink LED, and update display.

***Interrupt to Read Input***

I use the push button to trigger **int 1**.

When **initializing the int 1**, I need to do the following:

1. save the IP and CS of the int1's ISR information into the interrupt vector table, the address of IP is: 4\*I1TYP, and the address for CS is: 4\*I1TYP + 2
2. Initialize the int1, assign I1CON with 1, means priority 1,disable fully nested mode, disable cascade mode, set to edge-trigger, and enable interrupt from this source.

Inside the ISR (**btn\_isr**), I'll first disable interrupt and call a **read\_input** procedure to read PORTB. Afterwards, I'll send EOI the I1TYP value, to clear the in service status. In the end, call iret.

The **read\_input** proc:

* If it's in safe mode and it's moving, it would simply ignore this request (call ret).
* If request not ignored, it would do the following:
  1. store the level info
  2. update display of both PORTA and PORTC
  3. reset the timer count.

***Blink LED***

I use **timer 2** to blink LED.

Each timer count would take 4 clock cycle. The maximum value for timer compare register is 2^32 = 65536. Making the clock to blink 20 times every clock cycle would be a good option. Therefore, my timer count would be set to: 2M / 40 = 50000. This is within the range.

When initializing the timer (for all timers), I need to do the following:

1. update interrupt vector table
2. clear count
3. set compare value
4. set control register
5. In the end, set the Internal Interrupt Control Register (ICUCON)

In the case of timer 2, I need to set compare value to 50 000, control register to 1010000000000001b, means enable timer, disable write to enable pin, generate interrupt, clear max count, set to continuous mode..

The timer 2 ISR would call the "update\_pta\_led" proc, which would check move status, blink corresponding bit in PORTA if it's moving, and store this porta value for future check.

***Timer to update level display***

I use **timer 1** to update level display.

Since timer 2 would be triggered 40 times per second, I use timer 2 as a pre-scalar for timer 1, and set **T1CMPA** to be 40. The **T1CON** is set to be 1010000000001001b, which means: enable timer 1, disable write, enable interrupt, use CMPA, clear MC, clear RTG, user timer 2 as pre-scalar, use internal clock, do not alternate compare register, set to continuous mode.

The timer1\_isr would call procedure update\_disp, which would move the lift by 1 level, and update the 7-segment display to display current level. The logic for the this proc is:

1. compare the current\_level and target\_level
2. update the current level variable based on the result
3. update the move\_status variable based on the newly updated current\_level
4. display the current level information using the 7-segment display (a division is performed here to convert the integer to be binary-coded-decimal)

# Lesson & Things Learnt

I have to say that this is really a time-consuming project. The code is easy, the circuit is simply, but the debugging is hard. I spend **around 6 hours** to read the user manual, and only spend **1~2 hours** to write the code; but I spent **several days** to debug the board.

The good point is, I've really learnt something from doing this project, here's a summary.

## Soldering & Wire-wrapping Skills

Soldering the circuit was a very bad thing to me before I do this project, and I never heard of wire-wrapping before.

During this project, I did lots of soldering: push button, socket, register, capacitor. I can do fast and beautiful soldering now, and I like soldering a lot more than before.

For the wire-wrapping, I spent a whole day to do the wire-wrapping for major project 1, but now I'm quite fast with it.

I think the soldering and wire-wrapping skills are essential for a Computer Engineering student, for him to build complex circuits. I learnt that skill from this module.

### Read the User Manual and Datasheet

Teacher cannot teach you everything. Most of the things we used in life, are learnt by our own. This is especially true for this project.

In order to write correct code, I need to carefully read about the interrupt section of the very thick user manual; in order to use timer, I need to read though the chapter for timer/counter unit; in order to understand the example code, I need to read through the Chip-select unit; in order to know about the CPU frequency, I need to refer to "clock-generation and power management"... so on and so forth.

Also I need to refer to the datasheet on the expected behavior of the RAM, ROM, and 8255 chip.

I learnt more on both the topic included in the lecture, and not included in the lecture, via reading the data sheet.

And if I have a unknown chip with me now, I know how to know about it immediately: find the datasheet and read it.

### Debugging Skills

"What doesn't kill you make you stronger". This is true for all hard subjects, including this project.

For software programming, if it doesn't work, we know for sure it's the problem with the code. For electrical engineering, if it doesn't work, it can be the problem with anywhere: wrong connection of the circuit, non-working components, wrong code loaded, no power connected, bad soldering/wrapping lead to short circuit, etc.

Through trial and error, I've gain more experience with debugging (therefore I can debug a lot faster than before). Here're a few tips I learnt/used:

* When debugging the hardware, remove/replace one element each time
* When encounter problems, we need to trace back to our last success case, to confirm the working part
* Label the chips. So that when the we replace a certain, we can differentiate them
* Be more careful when building a circuit: the best way to fix a problem is to prevent it. A careful-built circuit is much more stronger than a massy one, and less likely to cause unknown bugs. This small amount of time you spent on paying more attention would finally give you a huge return

# Conclusion

I think here's the end of this report.

Throughout this project, I went through the whole process of circuit development: from the soldering the components onto the PCB board, to write code to control, to the end-prototype product.

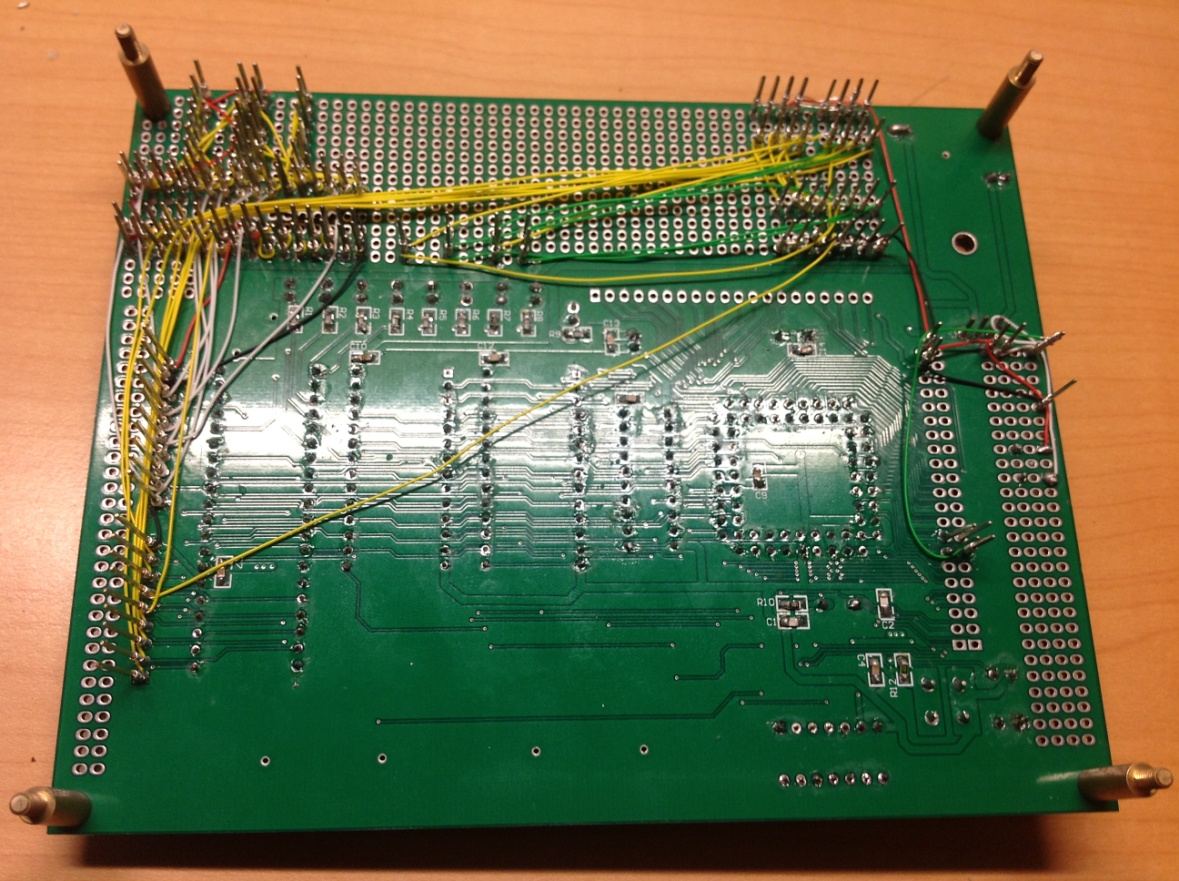
Balancing the huge amount of time I've put in (mostly on debugging) and the experiences I've gain and the things I've learnt, this is a meaningful(but time-consuming) project.

I think next time, I can build a board on my own, from sketch.

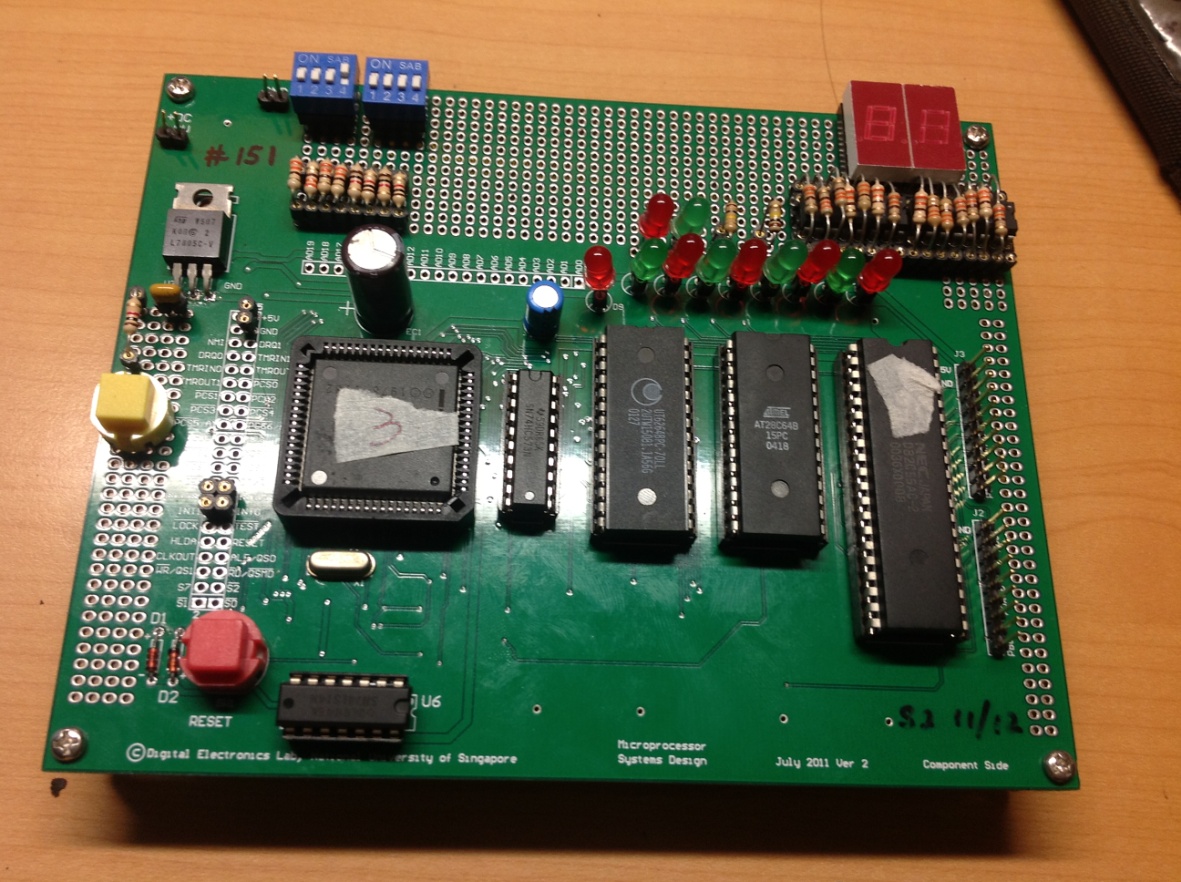
# Appendix:

## Photo of the board

***Front:***



***Back:***



## Code for Major Project 1

$mod186

NAME EG0\_COMP

;IO Setup for 80C188

UMCR EQU 0FFA0H ; Upper Memory Control Register

LMCR EQU 0FFA2H ; Lower Memory control Register

PCSBA EQU 0FFA4H ; Peripheral Chip Select Base Address

MPCS EQU 0FFA8H ; MMCS and PCS Alter Control Register

CWR EQU 0083H ; Address of Control Word Register

PORTA EQU 0080H ; Address of PORTA

PORTB EQU 0081H ; Address of PORTB

PORTC EQU 0082H ; Address of PORTC

; STACK SEGMENT

STACK\_SEG SEGMENT

STACK\_SEG ENDS

; DATA SEGMENT

DATA\_SEG SEGMENT

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, PCSBA

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

; YOUR CODE HERE ...

; Initialize 8255 with the right MCW

MOV DX, CWR

MOV AL, 80H ; IO mode, Group A - 0, Group B - 0

OUT DX, AL

HEAD:

; day

; Output to PORTC first, which connects to the 7-segment display

MOV DX, PORTC

MOV AL, 031H ; the day info

OUT DX, AL

;Output to PORTA then, whichi connects to the LEDs

;Since LEDs are common cathode, the output values should be reverted first

MOV DX, PORTA

NOT AL

OUT DX, AL

; Wait for 1 second

MOV BX, 0008H ; 8 iterations for the outer loop, accounts for 1 second

AGAIN5: MOV CX, 0BA03H ; 47619 iterations for the inner loop, accounts for 1M clock cycles

AGAIN6: NOP ; takes three clock cycles

LOOP AGAIN6 ; takes 18 clock cycles, accumulate to 21 clock cycles per loop

DEC BX

JNZ AGAIN5

; month

MOV DX, PORTC

MOV AL, 01H

OUT DX, AL

MOV DX, PORTA

NOT AL

OUT DX,AL

; Wait for 1 second

MOV BX, 0008H ; 8 iterations for the outer loop, accounts for 1 second

AGAIN3: MOV CX, 0BA03H ; 47619 iterations for the inner loop, accounts for 1M clock cycles

AGAIN4: NOP ; takes three clock cycles

LOOP AGAIN4 ; takes 18 clock cycles, accumulate to 21 clock cycles per loop

DEC BX

JNZ AGAIN3

; year

MOV DX, PORTC

MOV AL, 90H

OUT DX, AL

MOV DX, PORTA

NOT AL

OUT DX,AL

; Wait for 1 second

MOV BX, 0008H ; 8 iterations for the outer loop, accounts for 1 second

AGAIN5: MOV CX, 0BA03H ; 47619 iterations for the inner loop, accounts for 1M clock cycles

AGAIN6: NOP ; takes three clock cycles

LOOP AGAIN6 ; takes 18 clock cycles, accumulate to 21 clock cycles per loop

DEC BX

JNZ AGAIN5

; Loop back to the beginning of this application and restart the display

JMP HEAD

CODE\_SEG ENDS

END

## Code for Major Project 2

$mod186

NAME EG0\_COMP

;IO Setup for 80C188

PCB\_BASE EQU 0FF00H

UMCS EQU 0FFA0H ; Upper Memory Control Register

LMCS EQU 0FFA2H ; Lower Memory control Register

PACS EQU 0FFA4H ; PACS -- PCS Control Register.

MPCS EQU 0FFA8H ; MMCS and PCS Alter Control Register

PORTA EQU 0080H

PORTB EQU 0081H

PORTC EQU 0082H

CWR EQU 0083H ; Address of Control Word Register

EOI EQU PCB\_BASE + 22H

TCUCON EQU PCB\_BASE + 32H

I1CON EQU PCB\_BASE + 3AH

I1TYP EQU 0dh

T0CNT EQU PCB\_BASE + 50H

T0CMPA EQU PCB\_BASE + 52H

T0CMPB EQU PCB\_BASE + 54H

T0CON EQU PCB\_BASE + 56H

T0TYP EQU 8

T1CNT EQU PCB\_BASE + 58H

T1CMPA EQU PCB\_BASE + 5AH

T1CMPB EQU PCB\_BASE + 5CH

T1CON EQU PCB\_BASE + 5EH

T1TYP EQU 18

T2CNT EQU PCB\_BASE + 60H

T2CMPA EQU PCB\_BASE + 62H

T2CON EQU PCB\_BASE + 66H

T2TYP EQU 19

; magic variables for mov status

MS\_NO\_MOVE EQU 0 ; would only handle button click when there's no movement

; make sure the first bit is one

MS\_INC EQU 3 ; increasing

MS\_DEC EQU 1 ; decreasing

MS\_BTN\_CLK EQU 4 ; button clicked, need check

; DATA SEGMENT

DATA\_SEG SEGMENT

ORG 600H

db 128 dup(?)

tos label word

ORG 500H

to\_level db ? ; only the proper level for display

c\_level db ? ; current level

old\_pa\_val db ? ; the value of port A

mov\_stats db ? ;

safe\_mode db ? ; bool for safe/unsafe mode

; bit 7 of PORTB

; bit on: safe mode

; bit off: unsafe mode

multi\_mode db ? ; bool for multi/single mode

; bit 6 of PORTB

; bit on: multi mode

; bit off: single mode

DATA\_SEG ENDS

; RESET SEGMENT -- the first piece of code being executed.

Reset\_Seg SEGMENT

MOV DX, UMCS

MOV AX, 03E07H ; U17:10 = 11 1110 00 (0011 1111 1111 1100 0000, pos of 1 in the whole reg) = F8.

; Block Size 8, starting address 0FE000H (referring to Page 167 of the user manual)

; starting address FF, bus wait for 3 cycle

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:DATA\_SEG

START:

cli ; disable interrupt to do necessary configuration first.

; Initialize the stack segment and data segment

mov ax,DATA\_SEG

mov ds,ax

mov ss,ax

mov sp,offset tos

mov ax,0

mov es,ax

; Initialize MPCS to MAP peripheral to IO address

MOV DX, MPCS

MOV AX, 0083H ; block size unknown -- 00 is not specified in the table on the user manual

; Wait activate PCS6:0 for IO bus cycle

; Wait for 3 bus cycle -- for PCS6:4

OUT DX, AX

; PACS initial, set the parallel port start from 00H

MOV DX, PACS

MOV AX, 0003H ; Peripheral starting address 00H, Wait for 3 Bus Cycle

OUT DX, AX

; Initialize LMCS

MOV DX, LMCS

MOV AX, 01C4H ; ending address B01 1100 01 = 71H, wait for 3 wait states

OUT DX, AX

; YOUR CODE HERE ...

; init my variables

mov al,031h

mov to\_level,0 ; starting from 0 level

mov c\_level,0

mov mov\_stats,MS\_NO\_MOVE

mov old\_pa\_val,0ffh ; should be all off -- therefore ff

; Initialize 8255 with the right MCW

mov dx, CWR

mov al, 82h ; Binary 1000 0010 -- O mode, Group A - 0, Group B - 1(input), Group C - 0(output);

out dx, al

; Clear lights

mov al,0ffh

mov dx,PORTA

out dx,al

; clear level info

mov al,0

mov dx,PORTC

out dx,al

; \*\*\* PUSH BUTTON RELATED \*\*\*

; Initialize the int 1 in the interrupt table

mov si, 4 \* I1TYP

mov word ptr es:[si], offset btn\_isr

mov si, 4 \* I1TYP + 2

mov word ptr es:[si], seg btn\_isr

; Initialize the int 1

mov dx, I1CON ; location of the interrupt control register

mov ax, 1 ; enable interrupt, and priority 1

out dx, ax ; -- cuz handling this interrupt is supposed to be fast

; \*\*\* END OF PUSH BUTTON RELATED \*\*\*

; \*\*\* TIMER RELATED \*\*\*

; timer 1

; Initialize the interrupt table for timer 1 -- used for updating display

mov si, 4 \* T1TYP

mov word ptr es:[si], offset timer1\_isr

add si,2

mov word ptr es:[si], seg timer1\_isr

; Initialize the timer 1

mov dx,T1CNT ; clear the timer count

mov ax,0

; out dx,ax

mov dx,T1CMPA ; 40

mov ax,40

;out dx,ax

; control reg

mov dx,T1CON

mov ax,1010000000001001b ; Use CMPA, RTG0, P is 1 (use T2 to trigger) last bit set to enable

; timer 2

; initialize the interrupt vector table for timer 2 -- used for blinking LED

mov si, 4 \* T2TYP

mov word ptr es:[si], offset timer2\_isr

add si,2

mov word ptr es:[si], seg timer2\_isr

; Clear Timer 2

; use timer 2 to blink LED

mov dx,T2CNT

mov ax,0

out dx,ax

mov dx,T2CMPA ;50000 for T2

mov ax,50000

out dx,ax

; control reg

mov dx,T2CON

mov ax,1010000000000001b ; Use T2, enable, generate interrupt; also used to trigger T1)

out dx,ax

; in the end, the tcucon -- internal interrupt control register

mov dx,TCUCON

mov ax,0

out dx,ax

; \*\*\* END OF TIMER RELATED \*\*\*

sti

head\_st:

nop ; just wait here... all the things should be handled by different interrupt

jmp head\_st

ret ; just end here (but actually no need cuz here's a loop)

btn\_isr proc

cli

call read\_input

mov dx,EOI

mov ax,I1TYP

out dx,ax

sti

iret

btn\_isr endp

timer1\_isr proc

cli

call update\_disp

mov ax,T1TYP

mov dx,EOI

out dx,ax

sti

iret

timer1\_isr endp

timer2\_isr proc

cli

call update\_pta\_led

mov ax,T2TYP

mov dx,EOI

out dx,ax

sti

iret

timer2\_isr endp

read\_input proc

mov dx,PORTB

in al,dx

; test safe mode

test al,10000000b

jz ri\_safe\_test\_done

test mov\_stats,1 ; see moving or not

jz ri\_safe\_test\_done ; non-moving, allowed

ret ; in safemode, simply return

ri\_safe\_test\_done:

and al,00111111b

; save it to to\_level

mov to\_level, al

; save this value in old\_pa\_val, use update

xor al,0ffh

mov old\_pa\_val,al

call update\_disp

call update\_pta\_led

; and reset the timer 1, timer 2

mov dx,T1CNT

mov ax,0

out dx,ax

mov dx,T2CNT

mov ax,0

out dx,ax

ret

read\_input endp

; update the display of the move info -- caller should disable interrupt for it

update\_disp proc

; cmp to\_level, c\_level

; update level

; update status

; display level

; update current info

mov al,c\_level

cmp to\_level,al

ja update\_lvl\_above

jb update\_lvl\_below

; otherwise it's no move

mov mov\_stats,MS\_NO\_MOVE

; clear the first two bit of the move status

mov bl,old\_pa\_val

or bl,11000000b

mov old\_pa\_val,bl

; no move, no need to update display

jmp update\_sts

update\_lvl\_above:

inc al

jmp update\_lel\_var

update\_lvl\_below:

dec al

; and update the c\_level info

update\_lel\_var:

mov c\_level,al

; update status variable

update\_sts:

cmp to\_level,al

ja update\_sts\_above

jb update\_sts\_below

; status no move

mov mov\_stats,MS\_NO\_MOVE

jmp disp\_c\_level

update\_sts\_above:

jmp disp\_c\_level

mov mov\_stats,MS\_INC

update\_sts\_below:

mov mov\_stats,MS\_DEC

disp\_c\_level:

; al is still holding the current level info

mov bl,10

mov ah,0

div bl

mov cl,4

shl al,cl

or al,ah

mov dx,PORTC

out dx,al

ret

update\_disp endp

; caller should disable interrupt for it

update\_pta\_led proc

test mov\_stats,1 ; see it's moving or not

jz update\_pta\_led\_done ; no move, done the job

mov al,old\_pa\_val

test mov\_stats,10b ; see it's raising or falling

jz update\_pta\_led\_fall ; true when it's increaseing, false when falling

xor al,10000000b ; first bit for increasing

jmp update\_pta\_led\_update\_pta

update\_pta\_led\_fall:

xor al,01000000b ; second bit for decreasing

update\_pta\_led\_update\_pta:

mov dx,PORTA

out dx,al

mov al,old\_pa\_val ; and save the updated val

update\_pta\_led\_done:

ret

update\_pta\_led endp

code\_seg ends

end