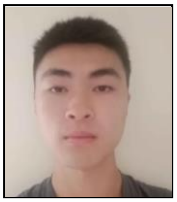


微算機實驗報告

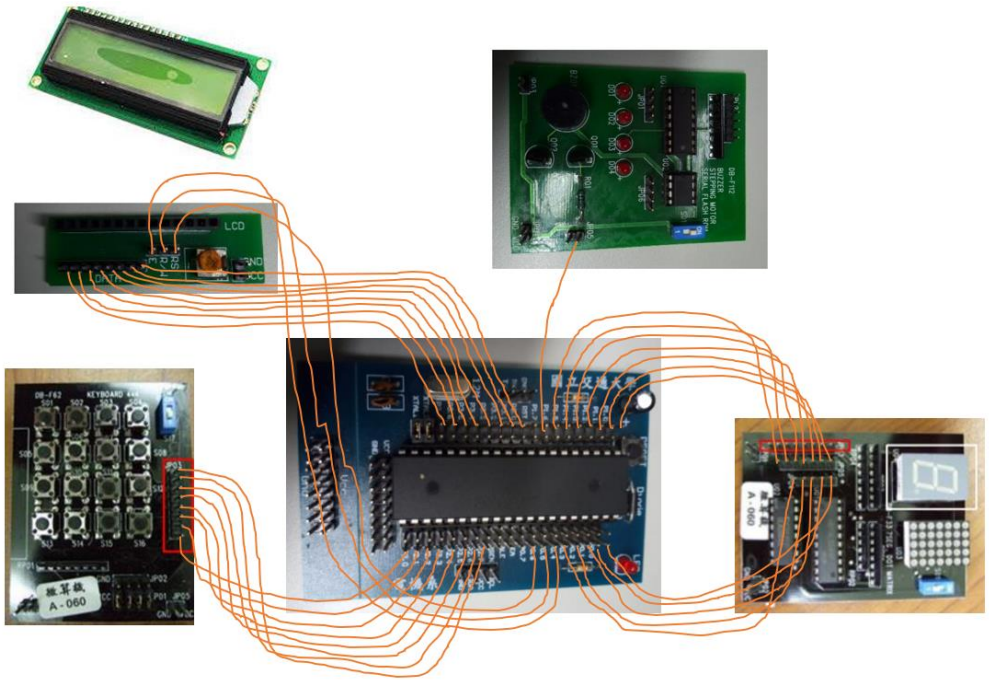
Lab # Final

姓名：朱沿道
系級：電機 15E
學號：
11511234
上課時間：T56AB

一、實驗目的：

如何在有限的程式記憶體空間，有限的資料記憶體空間，有限的指令，有限的 IO 接腳，沒有任何第三方函式庫的幫助，在全部只有 0 跟 1 的狀況下，結合各種實驗所學，東拼西湊，做出一個能觀賞，並且聲光俱全的專案。

二、硬體架構：



這是一個可以放在手掌上的生命遊戲模擬器，有限空間，無限可能 (343597383685 種組合)。

按鈕配置如下:

	編輯游標向上		音效 Debug
編輯游標向左	游標震盪	編輯游標向右	
	編輯游標向下		
Reset			開始遊戲

遊戲開始前，要先畫出活著細胞的配置，LCD 除了顯示 Edit Mode，在第二列有三個參數:

C 為 Column: 編輯游標的位置(行)

R 為 Row: 編輯游標的位置(列)

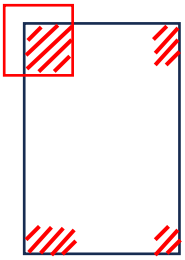
B 為 B: 當亮起時，游標會在螢幕上閃爍，同時更改此位置細胞的死活

透過編輯游標上下左右鍵去移動游標，按下游標震盪等 LED 燈改變來設定細胞的初始狀態。如果要一次性更改或是像小畫家一樣去畫東西，可以在 LED 燈由暗變亮時去移動游標。如果要跳到其他地方而不想更改移動途中的細胞，可以在按一次游標震盪鍵去暫停游標的震盪，透過 LCD R 與 C 的提示到玉移動的位置後，重新按游標震盪鍵(想要弄成小畫家的樣子結果這設計真的反人類)。設定完後，按下遊戲開始鍵即可看這個細胞組態繁衍的過程。

Reset 鍵隨時可按，將設備回到設備給電的瞬間。

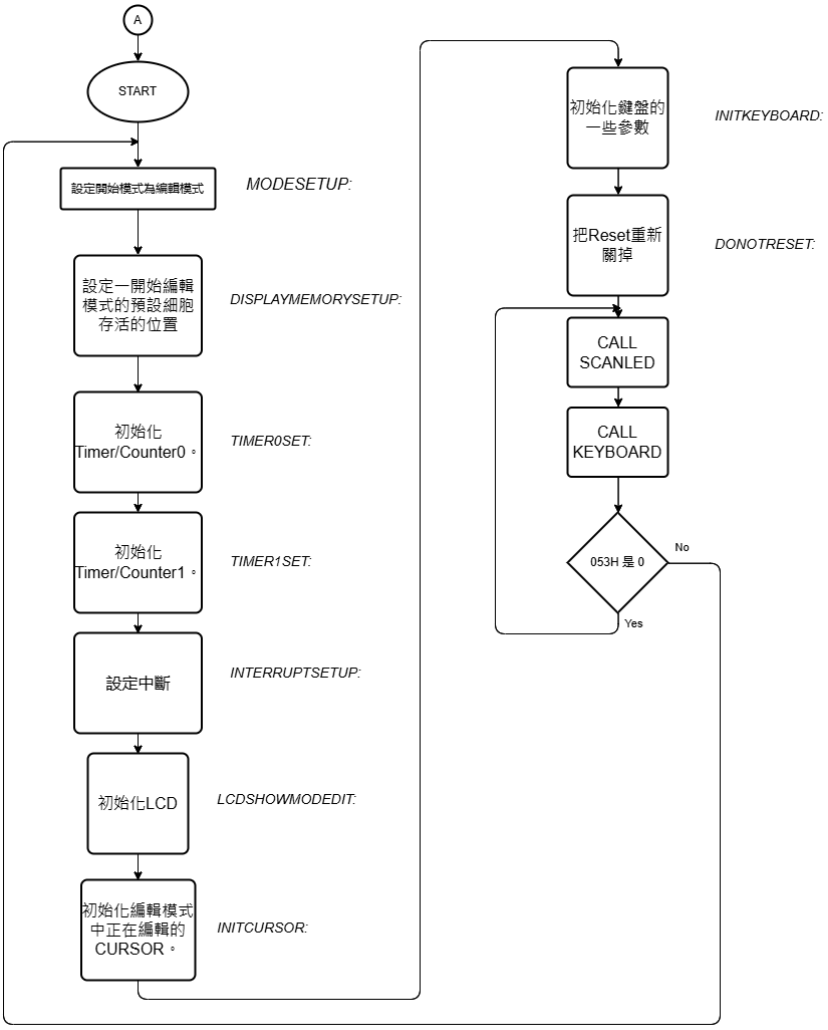
每一次細胞繁衍，規則如下:

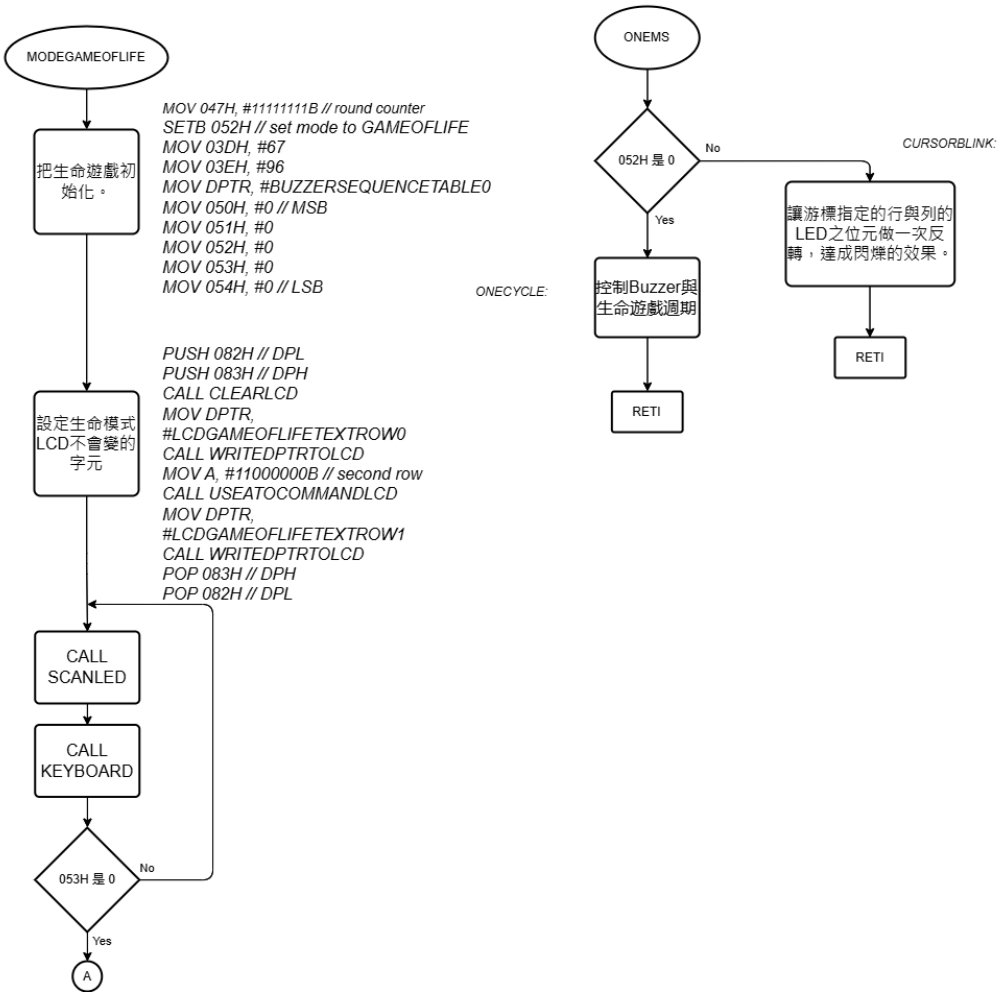
活著的細胞鄰居(九宮格)如果只有 1 個活著或更少，這個細胞會孤獨而死。如果有 3 個或更多活細胞，則會擁擠而死。死掉的細胞如果鄰居有剛剛好三個細胞是活的，那就會由死細胞變活細胞。因為只有 5*7 個空間，九宮格如果沒有細胞的，會將另一邊的細胞納入考量，增加無法控制性。



在生命模式中，LCD 會顯示五個參數:R, A, D, S, SD。
R 為 Round 的縮寫，代表這是第幾次繁衍。第一次繁衍為 0 不然太容易預測了。
A 為 Alives 的縮寫，代表目前有多少個細胞是活著的。
D 為 Delta 的縮寫，代表目前與上一次活著細胞的數量變化。
S 為 Score 的縮寫，為目前的總分。
SD 為 Score Delta，為這次繁衍對總分造成的變化。
SD 的公式為: 如果 D 大於 0，SD 為(R*D)%256；如果 D 小於等於 0，那 SD 為 0。
生命遊戲並不會停止，但是會有兩種狀況:
全部細胞死光與細胞數量永遠固定。
遊戲的目標為找到可以拿到最高分的初始細胞配置，同時 D 在不同的狀況下，繁衍與繁衍之間的音效會不同。共四個音效。
音效 credits: Quadpad Arpeggios (NES/Famicom 8bit 2A03) from Youtube

三、程式流程圖：


A
START
設定開始模式為編輯模式
MODESETUP:
設定一開始編輯模式的預設細胞存活的位置
DISPLAYMEMORYSETUP:
初始化Timer/Counter0。
TIMER0SET:
初始化Timer/Counter1。
TIMER1SET:
設定中斷
INTERRUPTSETUP:
初始化LCD
LCDSHOWMODEDIT:
初始化編輯模式中正在編輯的CURSOR。
INITCURSOR:
初始化鍵盤的一些參數
INITKEYBOARD:
把Reset重新關掉
DONOTRESET:
CALL SCANLED
CALL KEYBOARD
053H 是 0
No
Yes



四、程式碼與註解：

下面是每個記憶體位置的功能:

20H – 24H	**顯示記憶體。只要更改這裡的記憶體位置，7*5 點矩陣 LED 就會根據這裡的記憶體更改每個 LED 的亮暗** **因為每個 byte 為 8 個 bit，所以除了 MSB 之外，每個 bit 都有其配對的 LED 燈。** **每個 byte 為 5*7 顯示器的欄位，以底線表示。除了最上面那一行，剩下的為每個位置 LED 所對應的 bit。** **20H 21H 22H 23H 24H** **00H 08H 10H 18H 20H** **01H 09H 11H 19H 21H**

<table>
<tr><td></td><td>
<table>
<tr><td>02H</td><td>0AH</td><td>12H</td><td>1AH</td><td>22H</td></tr>
<tr><td>03H</td><td>0BH</td><td>13H</td><td>1BH</td><td>23H</td></tr>
<tr><td>04H</td><td>0CH</td><td>14H</td><td>1CH</td><td>24H</td></tr>
<tr><td>05H</td><td>0DH</td><td>15H</td><td>1DH</td><td>25H</td></tr>
<tr><td>06H</td><td>0EH</td><td>16H</td><td>1EH</td><td>26H</td></tr>
</table>
</td></tr>
<tr><td>25H</td><td>存目前掃描 LED 在掃描第幾行
如果是最上面的行: 00000001B
如果是最下面的行: 00010000B</td></tr>
<tr><td>26H</td><td>存目前掃描 LED 在掃描第幾列
如果是最左邊的列: 00000001B
如果是最右邊的列: 01000000B</td></tr>
<tr><td>27H</td><td>存目前掃描 LED 在掃描第幾個記憶體，搭配 MOV R0, 027H MOV A, @R0 使用。範圍為 020H – 024H</td></tr>
<tr><td>28H, 29H</td><td>Delay LED 程式使用，這樣 Delay 中中斷到另一個 Delay 的話，不會衝突到。</td></tr>
<tr><td>2AH</td><td>存一些可以用單個 bit 表示的資料
分別為:
050H: 是否將鍵盤鎖住不給使用者輸入
051H: 編輯模式時，是否要讓 Cursor 指向的 LED 燈閃爍,更改那個 LED 位置。
052H: 0 為編輯模式，1 為遊戲中模式
053H: 一輪後是否要 Reset</td></tr>
<tr><td>2BH - 2EH</td><td>每次計算下一輪細胞的存活暫時存起來的位置。跟 20H-2EH 格式相同</td></tr>
<tr><td>30H</td><td>目前 Cursor 所在的行
如果是最上面的行: 00000001B
如果是最下面的行 00010000B</td></tr>
<tr><td>31H</td><td>目前 Cursor 所在的列
如果是最左邊的列: 00000001B
如果是最右邊的列: 01000000B</td></tr>
<tr><td>32H</td><td>存上一次鍵盤按下哪個鍵，鍵盤上的鍵編號如下:
<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>12</td><td>13</td><td>14</td><td>15</td></tr>
</table></td></tr>
<tr><td>33H</td><td>Keyboard delay 程式區塊使用</td></tr>
<tr><td>34H, 35H</td><td>計算什麼時候 Cursor 位置的 Bit 要反轉</td></tr>
<tr><td>36H</td><td>給在執行 bit 反轉的 bit 所使用。</td></tr>
<tr><td>37H, 38H</td><td>標記目前計算 LED 是否在下一輪存活之位置的兩個記憶體位置。037H 為行，038H 為列</td></tr>
<tr><td>39H</td><td>存目前這個正在計算的 LED 位置旁邊九宮格有多少細胞是存活的</td></tr>
<tr><td>3AH, 3BH</td><td>標記目前計算 LED 之正在計算的鄰居細胞的位置是否存活之位置的兩個記憶體位置。03AH 為行，03BH 為列</td></tr>
<tr><td>3CH</td><td>標記目前計算 LED 之正在計算的鄰居細胞的位置是否存活之位置的 ID 號碼。範圍為 0-7:<table>
<tr><td>0</td><td>1</td><td>2</td></tr>
<tr><td>3</td><td>目標</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
</table></td></tr>
<tr><td>3DH</td><td>儲存 1/6 音符需要多少毫秒</td></tr>
<tr><td>3EH</td><td>儲存需要多少個 1/6 音符的時間才能計算下一輪每個細胞的生死。</td></tr>
<tr><td>40H, 41H</td><td>儲存代表聲音頻率，給 Timer 的初始值，40H 給 THn，41H 給 TLn。</td></tr>
<tr><td>43H</td><td>儲存每一輪的計算有多少細胞活著</td></tr>
<tr><td>44H, 45H</td><td>用於 LCD Delay 中</td></tr>
<tr><td>46H</td><td>用於 WRITEDPTRTOLCD 中，表示目前在顯示第幾個字</td></tr>
<tr><td>47H</td><td>目前模擬是在第幾次迭代</td></tr>
<tr><td>48H</td><td>上一輪迭代有多少細胞是活著的</td></tr>
<tr><td>49H</td><td>迭代前與迭代後相差多少個細胞(含正負)</td></tr>
<tr><td>4CH, 4DH, 4EH</td><td>存 CONVERTAINTODECIMAL 的輸出的百位十位與個位</td></tr>
</table>

50H – 54H	**儲存玩家的總得分，分別為萬位千位百位十位與個位**
55H	**儲存每輪迭代，總得分的變化**

程式部分:

ORG 0000H JMP START ORG 000BH JMP ONEMS ORG 001BH JMP BUZZERFLIP ORG 0050H	從程式記憶體 0000H 開始填程式 程式一開始時，會先執行程式記憶體位置 0000H 的地方，此時會執行 JMP START，跳到 START 所指示的記憶體位置。 000BH 為 Timer0 overflow 中斷時執行的記憶體位置，計畫 1ms 執行一次。用於編輯模式 Blink 與遊戲模式的音符撥放長度與每個迭代的時間間隔。 001BH 為 Timer1 overflow 中斷時執行的地方，用於控制 Buzzer 使用。
START: MOV SP, #007H	一開始的時候把 Stack Pointer 設為 #007H。#007H 這個位置是 8051 一開始的初始值。當需要 PUSH 的時候，Stack Pointer 會先加一後把東西放進去。反之 POP 時會先把東西拿出來後 Stack Pointer 減一。這麼做是因為以後要 Reset

	時，可以不管 Stack Pointer 指向哪裡。
MODESETUP: CLR 52H // Set mode to Edit Mode	將目前的模式設為編輯模式。

DISPLAYMEMORYSETUP:
// look from the right side of the led matrix
// the MSB bit is not in the scope
MOV 020H, #00010000B
MOV 021H, #01010000B
MOV 022H, #00110000B
MOV 023H, #00000000B
MOV 024H, #00000000B
	設定一開始編輯模式的預設細胞存活的位置。

TIMER0SET:
// MOD 1
// TMOD.1 0
// TMOD.0 1
ANL TMOD, #11111101B
ORL TMOD, #00000001B
// TMOD.2 0 Use Timer
ANL TMOD, #11111011B
// TR0 1
SETB TR0
// TMOD.3 Gate 0
ANL TMOD, #11110111B
// 1ms
MOV TH0, #252
MOV TL0, #23
	初始化 Timer/Counter0。使用 Mode1 16bit 數數，使用 Timer，只看軟體操控數數開關，將初始值設好讓他一毫秒跑一次。最後把 Timer0 Enable。

TIMER1SET:
// Mode 1
// TMOD.5 = 0
// TMOD.4 = 1
ANL TMOD, #11011111B
ORL TMOD, #00010000B
// TMOD.6 0 Timer Mode
ANL TMOD, #10111111B
// TMOD.7 0 Gate
ANL TMOD, #01111111B
CLR TF1
	初始化 Timer/Counter1。使用 Mode1 16bit 數數，使用 Timer，只看軟體操控數數開關，這裡初始值不重要。沒把 Timer0 Enable 因為編輯模式時不發聲。把 TF1 清掉怕不小心進入中斷。

// Low C
// Buzzer Frequency init value
MOV 040H, #241
MOV 041H, #242
MOV TH1, 040H
MOV TL1, 041H
// Disable Timer
CLR TR1	

INTERRUPTSETUP:
// TF0 Enable
SETB 0A9H
// TF0 Low Priority
CLR 0B9H
// TF1 Enable
SETB 0ABH
// TF1 High Priority
SETB 0BBH
// Enable All
SETB 0AFH
	把 Timer/Counter0 與 Timer/Counter1 的 Overflow 個別中斷開啟，並且把總體中斷開啟。

MODEEDIT:	單晶片啟動時會先執行編輯模式

LCDSHOWMODEDIT:
MOV A, #00111011B
CALL USEATOCOMMANDLCD
MOV A, #00001110B // DCB
CALL USEATOCOMMANDLCD
MOV A, #00000110B // AC setting and screen
CALL USEATOCOMMANDLCD
MOV A, #00000001B
CALL USEATOCOMMANDLCD
MOV A, #10000000B // set ddram memory pointer to 0
CALL USEATOCOMMANDLCD
 	
CALL CLEARLCD
MOV DPTR, #LCDINITTEXT
CALL WRITEDPTRTOLCD
	初始化 LCD，把 LCD 清空,把填入資料的指標指向 DDRAM 的第一個記憶體空間。並且把編輯模式中不會改的 LCD 字元先寫上去。

MOV A, #11000000B
CALL USEATOCOMMANDLCD
MOV DPTR, #LCDINITTEXTROW2
CALL WRITEDPTRTOLCD
 	
; JMP INITCURSOR	

INITCURSOR:
// cursor column #00000001B
MOV 030H, #00000001B
// cursor row
MOV 031H, #00000001B
// cursor timer
MOV 034H, #250
MOV 035H, #4
SETB 051H // blink enable
	初始化編輯模式中正在編輯的CURSOR。因為沒有游標的幫忙，在這裡使用閃爍的方式去更改細胞的死活。還有一個鍵是去停止閃爍，這樣可以移動並且不更改移動途中的細胞，但是缺點是看不到滑鼠游標的位置，只能重新閃爍才能看到游標的位置。LCD 上也會顯示滑鼠游標的位置，但是比較不容易去使用。

INITKEYBOARD: CLR 050H // keyboard trigger locker	初始化鍵盤的一些參數。在這裡把 Locker 關掉。Locker 發現有輸入時就會把鍵盤鎖上，只有在發現不再輸入時才會把 Locker 重新打開。
DONOTRESET: CLR 053H	如果有按下 Reset，這邊會把 Reset 重新關掉，避免不斷的 Reset。
LOOPEDIT: CALL SCANLED CALL KEYBOARD	在編輯模式，會一直不斷的去掃描 LED 與讀取鍵盤。

LOOPEDITCHECKRESET: JNB 053H, ENDOFLOOPEDIT // reset code JMP START	檢查 Reset 有沒有被按下。如果有，跳到一開始，也就是單晶片啟動時一開始執行的程式。
ENDOFLOOPEDIT: JMP LOOPEDIT	一輪結束後，跳到 LOOPEDIT
MODEGAMEOFLIFE:	生命遊戲模式

INITGAMEOFLIFE:
MOV 047H, #11111111B // round counter
SETB 052H // set mode to GAMEOFLIFE
MOV 03DH, #67
MOV 03EH, #96
 	
MOV DPTR, #BUZZERSEQUENCETABLE0
 	
MOV 050H, #0 // MSB
MOV 051H, #0
MOV 052H, #0
MOV 053H, #0
MOV 054H, #0 // LSB
	把生命遊戲初始化。將回合設為 0-1，模式設為生命模式，讓 Timer0 中斷知道，設定 1/6 音符的播放長度為 67 毫秒，設定四個節拍為 96 個音符。

PUSH 082H // DPL
PUSH 083H // DPH
 	
CALL CLEARLCD
MOV DPTR, #LCDGAMEOFLIFETEXTROW0
CALL WRITEDPTRTOLCD
 	
MOV A, #11000000B // second row
CALL USEATOCOMMANDLCD
MOV DPTR, #LCDGAMEOFLIFETEXTROW1
CALL WRITEDPTRTOLCD
 	
POP 083H // DPH
POP 082H // DPL
	設定生命模式 LCD 不會變的字元，因為會用到 DPTR，而在生命模式中 DPTR 需要存目前演奏的旋律，因此利用 Stack Memory 把這個資訊暫存起來。

LOOPGAMEOFLIFE: CALL SCANLED CALL KEYBOARD	跟編輯模式一樣，掃描 5*7 點矩陣，並且讀取鍵盤。
LOOPGAMEOFLIFECHECKRESET: JNB 053H, ENDOFLOOPGAMEOFLIFE // reset code JMP START	如果玩家按下 Reset 見的話，跳回 START。
ENDOFLOOPGAMEOFLIFE: JMP LOOPGAMEOFLIFE	一輪結束後，跳到 LOOPGAMEOFLIFE
// --- main calls ---	接下來是主程式使用的副程式
SCANLED:	掃描 5*7LED 副程式
INITSCAN: PUSH 0E0H // 0E0H: A PUSH 00H // init the led so that it will light up ANL 080H, #11100000B // column MSB right ANL 090H, #10000000B // row MSB down // P1 MOV 027H, #020H // column data pointer MOV 025H, #00000001B // column light up MOV 026H, #00000001B // row light up	初始化每次掃描需要的記憶體空間與腳位。因為 LED 如果一開始沒有把所有腳位設為 0 的話，LED 是完全點亮不起來的，非常奇怪。還有設定目前掃描的行與列為最左行與最上列。

CHECKCOLUMNLIGHTUP:
MOV A, 025H
CJNE A, #00100000B,
PASSCHECKCOLUMNLIGHTUP
// end of a scan session
POP 00H
POP 0E0H
RET
	檢查掃描的行數是否超出範圍，如果超出範圍代表掃描一輪 5*7 點矩陣了，可以 Return。

PASSCHECKCOLUMNLIGHTUP:
CHECKROWLIGHTUP:
MOV A, 026H
CJNE A, #10000000B,
PASSCHECKROWLIGHTUP
// check row failed
MOV 026H, #00000001B
	檢查掃描的列數是否超出範圍，如果超出範圍代表這一行已經掃完了。把行數加一，列重整成最上面的那一列。並且把讀取的記憶體的指

INC 027H // data pointer + 1
 	
MOV A, 025H
RL A
MOV 025H, A
 	
JMP CHECKCOLUMNLIGHTUP
	標加一。做完後檢查更新後的行是否有效。

PASSCHECKROWLIGHTUP: LITUP: MOV R0, 027H // pointer MOV A, @R0 ANL A, 026H // row light up MOV P0, 025H // column	如果行數與列數都通過的話，根據 data pointer 指向的 data，拿取其列數的位元，如果是一的話點亮目標的 LED 燈。

// MOV P1, A Filter P1.7
ANL A, #01111111B
MOV B, A
MOV A, P1
ANL A, #10000000B
ORL A, 0F0H // B
	因為 LED 用的 P0 跟 LCD 的 Control Pins 共用，P1 跟 Buzzer 共用，因此使用 ORL 與 ANL 去更改只有 LED 的接腳。

LITUPCLEAN: CLR 090H CLR 091H CLR 092H CLR 093H CLR 094H CLR 095H CLR 096H LITUPCHECK0: CJNE A, #00000001B, LITUPCHECK1 SETB 090H JMP ENDLITUPFILLIN LITUPCHECK1: CJNE A, #00000010B, LITUPCHECK2	至於因為 Buzzer 極為敏感，連 P1 這個都不給用，因此用調整 bit 的方式去調整接腳。 一開始先把每個接腳清為 0，接著根據欲調整的列數去將目標 LED 進行修改。

SETB 091H JMP ENDLITUPFILLIN LITUPCHECK2: CJNE A, #00000100B, LITUPCHECK3 SETB 092H JMP ENDLITUPFILLIN LITUPCHECK3: CJNE A, #00001000B, LITUPCHECK4 SETB 093H JMP ENDLITUPFILLIN LITUPCHECK4: CJNE A, #00010000B, LITUPCHECK5 SETB 094H JMP ENDLITUPFILLIN LITUPCHECK5: CJNE A, #00100000B, LITUPCHECK6 SETB 095H JMP ENDLITUPFILLIN LITUPCHECK6: CJNE A, #01000000B, ENDLITUPFILLIN SETB 096H ; JMP ENDLITUPFILLIN	

ENDLITUPFILLIN:
AFTERLIT:
MOV A, 026H
RL A // row + 1
MOV 026H, A
 	
CALL LEDDELAY
MOV P0, #00000000B
ANL 090H, #10000000B // P1
JMP CHECKROWLIGHTUP
	控制完 LED 後，把列數加一，等待一段時間讓人眼看的到後，把 LED 關掉並且去檢查列數的有效性。

LEDDELAY: MOV 028H, #030H LEDDELAY1: DJNZ 028H, LEDDELAY1 RET	一個普通的 delay 程式決定每顆 LED 亮燈的時間。

KEYBOARD:
ROW1:
MOV P2, #07FH // 01111111
CALL KEYBOARDDELAY
MOV A, P2
ANL A, #0FH // 00001111 ?桃蔗
MOV 032H, #0 // TABLE[0][...]
CJNE A, #0FH, COL1 // there is someone pressing this column
ROW2:
MOV P2, #0BFH // 10111111
CALL KEYBOARDDELAY
MOV A, P2
ANL A, #0FH
MOV 032H, #4 // TABLE[1][...]
CJNE A, #0FH, COL1
ROW3:
MOV P2, #0DFH // 11011111
CALL KEYBOARDDELAY
MOV A, P2
ANL A, #0FH
MOV 032H, #8 // TABLE[2][...]
CJNE A, #0FH, COL1
ROW4:
MOV P2, #0EFH
CALL KEYBOARDDELAY
MOV A, P2
ANL A, #0FH
MOV 032H, #12 // TABLE[3][...]
CJNE A, #0FH, COL1
NOTHINGPRESSED:
CLR 050H
RET
	鍵盤副程式。先根據鍵盤需要偵測的列打入 0，等一段時間後讀取鍵盤回傳的訊號。如果此列有按鍵被按下的話，後四個 bit 會有一個 bit 變成 0。如果有 bit 變成 0，會跳到偵測那一行的副程式。四個列都會進行這樣的動作。跟據那一行數，會決定跑到副程式時，032H 為 0,4,8, 還是 12。如果沒有發現任何鍵盤按下的話。會把 Locker 打開，並且 return。透過 Locker 可以讓按鍵變成 Trigger 開關，只會捕捉按下的一瞬間。

COL1: CJNE A, #0EH, COL2 // 00001110 MOV R0, #0 JMP KEYBOARDEVENT	接下來，看後四個位元是哪個位元為 0。根據位元的位置決定 R0 的值。值的範圍為 0-3。知道按下

COL2:
CJNE A, #0DH, COL3 // 00001101
MOV R0, #1
JMP KEYBOARDEVENT
COL3:
CJNE A, #0BH, COL4 // 00001011
MOV R0, #2
JMP KEYBOARDEVENT
COL4:
CJNE A, #07H, KEYBOARD // 00000111
MOV R0, #3
; JMP KEYBOARDEVENT
KEYBOARDEVENT:
MOV A, 032H
ADD A, R0
MOV 032H, A
	按鍵的值是哪個列那個行時，把兩個數字加起來便是按鍵的 ID。按鍵 ID 編號可以查看上面記憶體的說明。最後把 ID 存進 032H。

JNB 050H, KEYBOARDUNLOCKED // locked JMP AFTERKEYBOARD	檢查 Locker 是否有鎖上，如果有的話，把這個按鍵事件丟掉。
KEYBOARDUNLOCKED: SETB 050H // keyboard locked MOV A, 032H // last time keyboard index	如果沒有鎖上，把 Locker 鎖上，接著處理按鍵的事件。
CJNE A, #0, KEYBOARDNOT0 // left up // 0 JMP AFTERKEYBOARD	0 號鍵完全沒有功能。一偵測到直接跳到偵測後的程式。
KEYBOARDNOT0: CJNE A, #1, KEYBOARDNOT1 // up // 1 JNB 052H, KEYBOARDNOT0EDITMODE // game of life mode JMP AFTERKEYBOARD KEYBOARDNOT0EDITMODE: CALL CHECKANDMOVECURSORUP JMP AFTERKEYBOARD	如果是 1 號鍵，檢查是不是在 Edit Mode。如果是在編輯模式，執行 CHECKANDMOVECURSORUP 這個副程式，讓游標向上移動。做完後跳到偵測後的程式。
KEYBOARDNOT1:	如果是 2 號鍵被按

CJNE A, #2, KEYBOARDNOT2 // 2 JMP AFTERKEYBOARD	下，2 號鍵沒功能，直接跳到偵測後的程式。

KEYBOARDNOT2:
CJNE A, #3, KEYBOARDNOT3
// 3
JNB 052H, KEYBOARDNOT2EDITMODE
// game of life mode
JMP AFTERKEYBOARD
KEYBOARDNOT2EDITMODE:
// test - set C High
SETB TR1
MOV A, #7
CALL TURNAINTOFREQUENCY
JMP AFTERKEYBOARD
	如果是 3 號鍵被按下，檢查是否是在編輯模式被按下。如果是會讓蜂鳴器發出 So 的聲音，只是單純 Debug 使用。做完後會跳到偵測後的程式。

KEYBOARDNOT3: CJNE A, #4, KEYBOARDNOT4 // left // 4 JNB 052H, KEYBOARDNOT3EDITMODE // game of life mode JMP AFTERKEYBOARD KEYBOARDNOT3EDITMODE: CALL CHECKANDMOVECURSORLEFT JMP AFTERKEYBOARD	如果是 4 號鍵，檢查是不是在 Edit Mode。如果是在編輯模式，執行 CHECKANDMOVECURSORLEFT 這個副程式，讓游標向左移動。做完後跳到偵測後的程式。

KEYBOARDNOT4:
CJNE A, #5, KEYBOARDNOT5 // enter
// 5
JNB 052H, KEYBOARDNOT4EDITMODE
JMP AFTERKEYBOARD
KEYBOARDNOT4EDITMODE:
// dont blink
CPL 051H
CHECKBLINKUPDATELCDINFO:
MOV A, #11001000B
CALL USEATOCOMMANDLCD
JNB 051H, NOBLINKUPDATELCDINFO
BLINKUPDATELCDINFO:
	如果是 5 號鍵，檢查是不是在 Edit Mode。如果是在編輯模式，把 Blink 代表的位元翻轉，並且根據反轉後是否有閃爍更改 LCD 第二列的第九個字元是否要顯示空白還是 B。

MOV A, #66
JMP BLINKUPDATINGLCDINFO
NOBLINKUPDATELCDINFO:
MOV A, #32
BLINKUPDATINGLCDINFO:
CALL USEASETTINGDATA
JMP AFTERKEYBOARD	

KEYBOARDNOT5: CJNE A, #6, KEYBOARDNOT6 // right // 6 JNB 052H, KEYBOARDNOT5EDITMODE // game of life mode JMP AFTERKEYBOARD KEYBOARDNOT5EDITMODE: CALL CHECKANDMOVECURSORRIGHT JMP AFTERKEYBOARD	如果是 6 號鍵，檢查是不是在 Edit Mode。如果是在編輯模式，執行 CHECKANDMOVECURSORRIGHT 這個副程式，讓游標向右移動。做完後跳到偵測後的程式。
KEYBOARDNOT6: CJNE A, #7, KEYBOARDNOT7 // 7 JMP AFTERKEYBOARD KEYBOARDNOT7: CJNE A, #8, KEYBOARDNOT8 // 8 JMP AFTERKEYBOARD	7 號鍵和 8 號鍵都沒有功能。
KEYBOARDNOT8: CJNE A, #9, KEYBOARDNOT9 // down // 9 JNB 052H, KEYBOARDNOT8EDITMODE // game of life mode JMP AFTERKEYBOARD KEYBOARDNOT8EDITMODE: CALL CHECKANDMOVECURSORDOWN JMP AFTERKEYBOARD	如果是 9 號鍵，檢查是不是在 Edit Mode。如果是在編輯模式，執行 CHECKANDMOVECURSORDOWN 這個副程式，讓游標向下移動。做完後跳到偵測後的程式。
KEYBOARDNOT9: CJNE A, #10, KEYBOARDNOT10 // 10	10 號鍵和 11 號鍵都沒有功能。

JMP AFTERKEYBOARD KEYBOARDNOT10: CJNE A, #11, KEYBOARDNOT11 // 11 JMP AFTERKEYBOARD	
KEYBOARDNOT11: CJNE A, #12, KEYBOARDNOT12 // 12 // all modes full reset SETB 053H // reset!	12 號鍵會把 053H 設為 1，main 在一次循環結束後會看 053H 是否為 1，如果為 1 的話會跳至 START。
KEYBOARDNOT12: CJNE A, #13, KEYBOARDNOT13 // 13 JMP AFTERKEYBOARD KEYBOARDNOT13: CJNE A, #14, KEYBOARDNOT14 // 14 JMP AFTERKEYBOARD	13 號鍵和 14 號鍵都沒有功能。
KEYBOARDNOT14: CJNE A, #15, KEYBOARDNOT15 // run the sim // 15 JNB 052H, KEYBOARDNOT14EDITMODE JMP AFTERKEYBOARD KEYBOARDNOT14EDITMODE: JMP MODEGAMEOFLIFE	15 號鍵會先檢查是否在編輯模式，如果在編輯模式的話會直接跳至生命遊戲模式。
KEYBOARDNOT15: // it should not happen AFTERKEYBOARD: RET	如果 0-15 都不是，恩不應該發生。事件結束後回到執行副程式的程式位置，可能是編輯模式，可能是生命遊戲模式。
KEYBOARDDELAY: MOV 033H, #10H KEYBOARDDELAY1: DJNZ 033H, KEYBOARDDELAY1 RET	在選擇偵測按鈕行數與讀取間有個 Delay，這段 code 決定時間的長短。

BUZZERFLIP:
CPL 097H
MOV TH1, 040H
MOV TL1, 041H
RETI
	當 Timer1 Overflow 中斷時，把 097H 反轉。這樣可以製造一個方波。最後把初始值放進去就可以結束中斷了。

BUZZERTABLE: DB 241, 22 DB 241, 242 DB 242, 182 DB 243, 122 DB 244, 41 DB 244, 214 DB 245, 112 DB 246, 8 DB 246, 155 DB 247, 30 DB 247, 157 DB 248, 23 // ----- DB 248, 139 DB 248, 242 DB 249, 90 DB 249, 183 DB 250, 20 DB 250, 102 DB 250, 184 DB 251, 3 DB 251, 74 DB 251, 143 DB 251, 206 DB 252, 11 // ----- DB 252, 67 DB 252, 120 DB 252, 171 DB 252, 219 DB 253, 8	這是所有音符需要的 THn TLn 的初始值，跨過三個音階，包含黑鍵白鍵

DB 253, 51 DB 253, 91 DB 253, 129 DB 253, 165 DB 253, 199 DB 253, 231 DB 254, 5 // -----	
BUZZERSEQUENCETABLE0: DB 27, 27, 27, 36, 36, 36, 27, 27, 27, 36, 36, 36 DB 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36 DB 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36 DB 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36 DB 27, 27, 27, 36, 36, 36, 27, 27, 27, 36, 36, 36 DB 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36 DB 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36 DB 36, 36, 36, 36, 36, 36, 27, 36, 27, 36, 27, 36 ; BUZZERSEQUENCETABLE1: ; DB 30, 30, 30, 30, 30, 30, 29, 29, 29, 29, 29, 29 ; DB 27, 27, 27, 27, 27, 27, 25, 25, 25, 25, 25, 25 ; DB 27, 27, 27, 27, 27, 27, 29, 29, 29, 29, 29, 29 ; DB 30, 30, 30, 30, 30, 30, 32, 32, 32, 32, 32, 32 ; DB 30, 30, 30, 30, 30, 30, 29, 29, 29, 29, 29, 29 ; DB 27, 27, 27, 27, 27, 27, 25, 25, 25, 25, 25, 25 ; DB 27, 27, 27, 27, 27, 27, 29, 29, 29, 29, 29, 29 ; DB 30, 30, 30, 30, 30, 30, 32, 32, 32, 32, 32, 32 ; BUZZERSEQUENCETABLE2: DB 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15 DB 15, 15, 15, 15, 15, 15, 13, 13, 13, 13, 13, 13 DB 18, 18, 18, 18, 18, 18, 17, 17, 17, 17, 17, 17 DB 15, 15, 15, 15, 15, 15, 13, 13, 13, 13, 13, 13 DB 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15 DB 15, 15, 15, 15, 15, 15, 13, 13, 13, 13, 13, 13 DB 18, 18, 18, 18, 18, 18, 17, 17, 17, 17, 17, 17 DB 15, 15, 15, 15, 15, 15, 13, 13, 13, 13, 13, 13	這 7 個 TABLE 存入不同的四個節拍同音調的旋律。因為 program memory 有限制，所以只用了 4 個旋律。一個給生出來的細胞小於死掉的細胞數量所撥的旋律，一個給生出來等於死掉的，一個給生出來比死掉的多一個，一個給生出來大於死掉並且不只一個。

```
; BUZZERSEQUENCETABLE3:
;       DB 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20
;       DB 20, 20, 20, 20, 20, 20, 22, 22, 22, 22, 22, 22
;       DB 23, 23, 23, 23, 23, 23, 22, 22, 22, 22, 22, 22
;       DB 20, 20, 20, 20, 20, 20, 15, 15, 15, 15, 15, 15
;       DB 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18
;       DB 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18
;       DB 20, 20, 20, 20, 20, 20, 17, 17, 17, 17, 17, 17
;       DB 15, 15, 15, 15, 15, 15, 13, 13, 13, 13, 13, 13
;
BUZZERSEQUENCETABLE4:
    DB 15, 15, 10, 10, 15, 15, 18, 18, 10, 10, 18, 18
    DB 27, 27, 29, 29, 30, 30, 29, 29, 27, 27, 22, 22
    DB 15, 15, 10, 10, 15, 15, 18, 18, 10, 10, 18, 18
    DB 27, 27, 29, 29, 30, 30, 29, 29, 27, 27, 22, 22
    DB 15, 15, 10, 10, 15, 15, 18, 18, 10, 10, 18, 18
    DB 27, 27, 29, 29, 30, 30, 29, 29, 27, 27, 22, 22
    DB 15, 15, 10, 10, 15, 15, 18, 18, 10, 10, 18, 18
    DB 27, 27, 29, 29, 30, 30, 29, 29, 27, 27, 22, 22

BUZZERSEQUENCETABLE5:
    DB 11, 11, 8, 8, 15, 15, 8, 8, 11, 11, 8, 8
    DB 11, 11, 8, 8, 15, 15, 8, 8, 11, 11, 8, 8
    DB 18, 18, 8, 8, 17, 17, 8, 8, 15, 15, 8, 8
    DB 18, 18, 8, 8, 17, 17, 8, 8, 15, 15, 8, 8
    DB 22, 22, 18, 18, 22, 22, 18, 18, 15, 15, 18, 18
    DB 15, 15, 10, 10, 15, 15, 10, 10, 6, 6, 10, 10
    DB 6, 6, 3, 3, 6, 6, 10, 10, 6, 6, 10, 10
    DB 15, 15, 10, 10, 15, 15, 18, 18, 20, 20, 22, 22

; BUZZERSEQUENCETABLE6:
;       DB 11, 11, 3, 3, 10, 10, 3, 3, 8, 8, 3, 3
;       DB 11, 11, 3, 3, 10, 10, 3, 3, 8, 8, 3, 3
;       DB 18, 18, 8, 8, 17, 17, 8, 8, 15, 15, 8, 8
;       DB 18, 18, 8, 8, 17, 17, 8, 8, 15, 15, 8, 8
;       DB 18, 18, 8, 8, 17, 17, 8, 8, 15, 15, 8, 8
;       DB 18, 18, 8, 8, 17, 17, 8, 8, 15, 15, 8, 8
;       DB 18, 18, 22, 22, 18, 18, 15, 15, 18, 18, 15, 15
```

; DB 10, 10, 15, 15, 10, 10, 6, 6, 10, 10, 6, 6 ; BUZZERSEQUENCETABLE7: ; DB 22, 22, 18, 18, 22, 22, 18, 18, 15, 15, 18, 18 ; DB 15, 15, 10, 10, 15, 15, 10, 10, 6, 6, 10, 10 ; DB 6, 6, 3, 3, 6, 6, 10, 10, 6, 6, 10, 10 ; DB 15, 15, 10, 10, 15, 15, 18, 18, 15, 15, 18, 18 ; DB 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8 ; DB 36, 36, 36, 36, 36, 36, 15, 15, 13, 13, 15, 15 ; DB 15, 15, 15, 15, 15, 15, 36, 36, 36, 36, 36, 36 ; DB 11, 11, 11, 11, 13, 13, 11, 11, 8, 8, 6, 6	
// --- buzzer callables --	下面是蜂鳴器相關的副程式。

TURNAINTOFREQUENCY:
  PUSH 0E0H // A
  PUSH 0F0H // B
  PUSH 082H // DPL
  PUSH 083H // DPH
  CJNE A, #36, NOTNOSOUND
  CLR TR1
  CLR 097H // weird
  JMP RETTURNAINTOFREQUENCY
	這個副程式會將目前 A 存的數字設定蜂鳴器的頻率。A 的範圍為 0-36，三個音階分別為 0-11 12-23 24-35，36 為沒聲音。執行這副程式前會先判斷是否為 36。如果為 36，會先把 Timer1 關掉，並且把連接 Buzzer 腳位設為 0，不然會發生奇怪的事情，LED 亮不起來。最後準備 Return。

NOTNOSOUND:
  SETB TR1
  MOV B, #2
  MUL AB
  MOV B, A
  MOV DPTR, #BUZZERTABLE
  MOVC A, @A+DPTR
  MOV 040H, A
  MOV A, B
	如果不是 36，Timer1 會先開啟，並且因為 BUZZERTABLE 是以兩個 byte 代表一個頻率的初始值，前面為 THn 後面 TLn。因此，A 要先乘以二，取得 THn

INC A
MOVC A, @A+DPTR
MOV 041H, A
; JMP RETTURNAINTOFREQUENCY
	後再把值加一，取得 TLn 的初始值。

RETTURNAINTOFREQUENCY: POP 083H // DPH POP 082H // DPL POP 0F0H // B POP 0E0H // A RET	最後把有 PUSH 的東西 POP 回來就可以 Return，避免影響到原本的運算。

ONEMS:
// this will modify display memory
ONEMSCHECKMODE:
// edit mode or game of life mode
JB 052H, ONECYCLE
; JMP CURSORBLINK
	這個是 Timer0 的中斷。會透過 052H 知道是編輯模式還是生命遊戲模式。這是整個遊戲最重要的計時器。

CURSORBLINK: PUSH 000H // R0 PUSH 0E0H // A JNB 051H, CURSORBLINKRETI DJNZ 034H, CURSORBLINKRETI MOV 034H, #250 DJNZ 035H, CURSORBLINKRETI	如果是在編輯模式的話，會跑這裡的副程式。這是讓游標指定的行與列的 LED 之位元做一次反轉，達成閃爍的效果。 因為我們要一秒做一次閃爍，因此要用兩個 byte 數 1000。當 1000 到時，才能執行位元轉換的動作。 在這裡讓 034H 一開始存 236 這個值，035H 存 4。利用 DJNZ 去扣裡面存的數字。先扣 034H，當 250 扣完後，會把 034H 存回 250，再扣 035H 1。直到 035H 扣沒

	了才將位元反轉。 程式需要扣 250*4 次，也就是 1000 次，才能進入位元反轉，也就是每一秒做一次反轉。

// blink!
// target column
MOV R0, #01FH
MOV A, 030H
RL A
	準備反轉虛擬游標指向的 LED 燈前，先把 R0 設為存哪些 LED 亮那些暗的第一個計體位置的前一個位置。A 存虛擬游標所在的行。並且把行移向上一行的位置(如果做一次下一行的運算的話，會回到原本 A 的值)

CURSORBLINKPOINTERLOCATING: INC R0 RR A CJNE A, #00000001B, CURSORBLINKPOINTERLOCATING	一開始，先把指向行的指標指向 030H 提供的行之資料上。因為在設定 R0 與 A 時有先把值往前推一個，所以經過 INC R0 與 RR A 後，A 為原本 A 的資料與 R0 為資料第一行的位置。 接著，會檢查 A 是否推向最右邊(最右邊代表最左邊的那一行)。如果不是，INC R0 並且 RR A，直到 A 推向最右邊，此時代表說 R0 指向的記憶體體位置為從這指標開始數第一個的位

	置，也就是目標位元就在這的記憶體位置上。

// pointer ready
// set the target bit to zero
MOV A, @R0
MOV 036H, A // processing data
MOV A, 31H
CPL A
ANL A, 036H
MOV @R0, A
	接下來，只需要透過 031H，就可以知道欲更改之列。
	031H 透過 00000010B 表示游標在第二列。
	先把指標指向的資料送進 036H，把 031H 反轉後 跟資料做 AND 運算。做完之後得到的資料為欲翻轉的位元設為 0 其他保持不變。

// invert get the bit and paste the bit
MOV A, 036H
CPL A
ANL A, 031H
	把欲控制的位元變成 0 後，將原本資料反轉，與 031H 做 ORL 運算，這樣可以得到除了欲控制位元的翻轉，其他設定為 0

ORL A, @R0 MOV @R0, A	做完兩個運算之後，把這兩個 ORL 起來，就可以得到把游標指向的位元翻轉的動作了。

// success -> update timer
MOV 035H, #4
; JMP CURSORBLINKRETI
	翻轉完後，重新把 035H 放 4 回去，重新數 1000，並且進入 Return 環節。

CURSORBLINKRETI: POP 0E0H // A POP 000H // R0 JMP ENDONEMS	Return 把暫存在 Stack Memory 存回去有使用到的暫存器，並且跳到中斷所在的 Return 副程

	式。
// ----- cursor call ----- 	下面副程式執行後會更改 030H 與 031H。

CHECKANDMOVECURSORLEFT:
MOV A, 030H
CJNE A, #00000001B, CURSORMOVELEFT
RET
CURSORMOVELEFT:
RR A
MOV 030H, A
JMP UPDATECURSORLCDINFO
	先檢查這個虛擬游標是否到最左邊了，如果到最左邊了，直接 Return。如果沒有到最底。把虛擬游標向左移動。

CHECKANDMOVECURSORRIGHT:
MOV A, 030H
CJNE A, #00010000B, CURSORMOVERIGHT
RET
CURSORMOVERIGHT:
RL A
MOV 030H, A
JMP UPDATECURSORLCDINFO
	先檢查這個虛擬游標是否到最右邊了，如果到最右邊了，直接 Return。如果沒有到最底。把虛擬游標向右移動。

CHECKANDMOVECURSORUP:
MOV A, 031H
CJNE A, #00000001B, CURSORMOVEUP
RET
CURSORMOVEUP:
RR A
MOV 031H, A
JMP UPDATECURSORLCDINFO
	先檢查這個虛擬游標是否到最上面了，如果到最上面了，直接 Return。如果沒有到最上面。把虛擬游標向上移動。

CHECKANDMOVECURSORDOWN:
MOV A, 031H
CJNE A, #01000000B, CURSORMOVEDOWN
RET
CURSORMOVEDOWN:
RL A
MOV 031H, A
; JMP UPDATECURSORLCDINFO
	先檢查這個虛擬游標是否到最下面了，如果到最下面了，直接 Return。如果沒有到下面。把虛擬游標向下移動。

UPDATECURSORLCDINFO: // column	如果虛擬游標有移動的話，更新顯示

MOV A, #11000010B
CALL USEATOCOMMANDLCD
MOV A, 030H
CALL TURN1LOCINTONUMBER
ADD A, #48
CALL USEASETTINGDATA
// row
MOV A, #11000110B
CALL USEATOCOMMANDLCD
MOV A, 031H
CALL TURN1LOCINTONUMBER
ADD A, #48
CALL USEASETTINGDATA
RET
	在 LCD 上的位置。一開始先調整下一次送資料進去時，會填入 DDRAM 的哪一個位置。接著，把遮罩的格式變成數字，並且加上 48 變成 ASCII 格式。行與列都是用這種方法。

// NOTE: Check Mode and if GAMEOFLIFEMODE RUN ONECYCLE
ONECYCLE:
PUSH 0E0H // A
PUSH 000H // R0
PUSH 0F0H // B
 	
DJNZ 03DH, ENDCYCLENOUPDATETEMP
MOV 03DH, #67
 	
MOV A, 03EH // 96 init
CJNE A, #0, UPDATEFREQUENCY
JMP NOUPDATEFREQUENCY
	如果為生命遊戲模式，每一毫秒都會執行這個副程式。03DH 控制每個音符的長度，預設為 67 毫秒。如果 67 毫秒還沒到，會 Return。每 67 毫秒一到，會檢查 03EH 是否為 0。如果為 0 的話，需要更新細胞組態並且換新的旋律。不為零的話，會換相同旋律但是是新的音符，及新的頻率。

UPDATEFREQUENCY:
MOV A, #96
CLR C
SUBB A, 03EH
MOVC A, @A+DPTR
CALL TURNAINTOFREQUENCY
	到這裡會 03EH 是 1~96，因為第一個音符為 96，第二個音符為 95，一直到最後一個音符為 1，所以透過 03EH -96 可以把 96~1 變成 0~95。透過

	DPTR 得到音符的編號之後，可以透過 TURNAINTOFREQUENCY 用 A 提供的訊息設定 Timer1 的初始值。
NOUPDATEFREQUENCY: DJNZ 03EH, ENDCYCLENOUPDATETEMP JMP INITCYCLE	忽略 DJNZ 那 03EH 是零了，會直接跳到 INITCYCLE，開始計算下一輪生命遊戲的細胞配置。
ENDCYCLENOUPDATETEMP: JMP ENDCYCLENOUPDATE	因為 DJNZ CJNE 能跳的距離有限，因此這邊用 LJMP 作為跳板。
INITCYCLE: MOV 043H, #0 // alive counter MOV 048H, #0 // previous alives MOV 037H, #10000000B // column MOV 038H, #10000000B // row	生命遊戲先這定些初始值，一開始檢查的行與列先設為一開始檢查的行數的前一個，運算完後活著的數量，與上一輪活著的數量都先清零。

CYCLENEXTCOLUMN:
MOV A, 037H
RL A
CJNE A, #00100000B, CYCLEROWCHECKED
JMP ENDCYCLE
CYCLEROWCHECKED:
MOV 037H, A
	將行數設為下一行，如果沒有超出範圍，把沒問題的行存起來，並且換列。超出範圍的話，結束計算。

CYCLENEXTROW:
MOV A, 038H
RL A
CJNE A, #10000000B,
CYCLECOLUMNROWCHECKED
MOV 038H, #10000000B
JMP CYCLENEXTCOLUMN
	將列數設為下一列，如果沒有超出範圍，把沒問題的列存起來。超出範圍的話，把列存為一開始的樣子，並且行數加一。

CYCLECOLUMNROWCHECKED: MOV 038H, A	
CYCLECOLUMNROWREADY: CYCLEGETNEIGHBORALIVES:	欲計算的細胞已經設定好了，下一步是計算有多少鄰居細胞是活著的。
CYCLEINITNEIGHBORBIT: MOV 039H, #0 // alive counter MOV 03CH, #0 // neighbor ID 0 - 7	039H 為存有多少鄰居細胞活著，03CH 為目前檢查的鄰居 ID。ID 編號可以看上面描述記憶體位置的表。
CYCLECHECKVALIDATENEIGHBORID: MOV A, 03CH CJNE A, #8, TEMPCYCLEADJUSTNEIGHBORBIT JMP CYCLECHECKNEIGHBORIDINVALID	檢查鄰居 ID 是否為 0-8 之間，如果是，檢查鄰居信息，如果不是，計算正在計算的細胞下一輪的結果。
TEMPCYCLEADJUSTNEIGHBORBIT: JMP CYCLEADJUSTNEIGHBORBIT	如果鄰居 ID 正常，因為要跳到比較遠的距離所以借助 LJMP 的力量。

CYCLECHECKNEIGHBORIDINVALID:
// neighbor count is ready
CYCLESTORENEWITERATION:
// NOTE: 039H: neighbor alives 037H, 038H: column and row loc
// NOTE: OLD: 20H - 24H
// NOTE: NEW: 2BH - 2FH
CYCLEIDENTIFYBITALIVEORDEAD:
CYCLEINITIDENTIFYBITALIVEORDEAD:
MOV R0, #20H
MOV A, 037H
	當數好鄰居的狀況時，先檢查目前計算的細胞是死的還是活的。R0 為存取行的指標，037H 存目標行的位置。

CYCLEIDENTIFYBITALIVEORDEADCHECKCOLUMN: CJNE A, #00000001, CYCLEIDENTIFYBITALIVEORDEADCHECKCOLUMNWRONG	把存目標行位置之(A)，不斷的把 1 向右移。向右移同時把指標向下一個記憶體位置移動。如

CYCLEIDENTIFYBITALIVEORDEADCHECKCOLUMNRIGHT: JMP CYCLEIDENTIFYBITALIVEORDEADGETROW CYCLEIDENTIFYBITALIVEORDEADCHECKCOLUMNWRONG: RR A INC R0 JMP CYCLEIDENTIFYBITALIVEORDEADCHECKCOLUMN	果 1 在最右邊的話。代表指標對了，換列。

CYCLEIDENTIFYBITALIVEORDEADGETROW:
// find if alive or dead not neighbor
MOV A, @R0
ANL A, 038H
CJNE A, #00000000B, CYCLEIDENTIFYBITALIVEORDEADALIVE
JMP CYCLEIDENTIFYBITALIVEORDEADDEAD
	透過 038H 與 A AND 起來，如果等於零代表 A 指示的位元為零，也就是死掉，反之亦然。

CYCLEIDENTIFYBITALIVEORDEADALIVE: INC 048H MOV A, 039H CJNE A, #2, CYCLEIDENTIFYBITALIVEORDEADALIVENOT2 JMP CYCLENEXTITERATIONALIVE CYCLEIDENTIFYBITALIVEORDEADALIVENOT2: CJNE A, #3, CYCLEIDENTIFYBITALIVEORDEADALIVEOTHER JMP CYCLENEXTITERATIONALIVE CYCLEIDENTIFYBITALIVEORDEADALIVEOTHER: JMP CYCLENEXTITERATIONDEAD	如果計算的細胞是活著的，檢查鄰居細胞活著的數量是否為 2, 3，如果是的話下一輪會活著，否則會死。
CYCLEIDENTIFYBITALIVEORDEADDEAD: MOV A, 039H	如果計算的細胞是死的，檢查鄰居細

CJNE A, #3, CYCLEIDENTIFYBITALIVEORDEADALIVEOTHER JMP CYCLENEXTITERATIONALIVE CYCLEIDENTIFYBITALIVEORDEADDEADOTHER: JMP CYCLENEXTITERATIONDEAD	胞活著的數量是否為 2，如果是的話下一輪會由死變活，否則保持死的。

CYCLENEXTITERATIONALIVE:
MOV A, 037H
MOV R0, #2BH
INC 043H // sequence score + 1
CYCLENEXTITERATIONALIVECHECKCOLUMN:
CJNE A, #00000001B,
CYCLENEXTITERATIONALIVENEXTCOLUMN
JMP
CYCLENEXTITERATIONALIVECOLUMNCHECKED
CYCLENEXTITERATIONALIVENEXTCOLUMN:
RR A
INC R0
JMP
CYCLENEXTITERATIONALIVECHECKCOLUMN
CYCLENEXTITERATIONALIVECOLUMNCHECKED:
MOV A, @R0
ORL A, 038H
MOV @R0, A
JMP CYCLEENDSTORENEWITERATION
	如果下一輪是活著的話，透過前面的方法，在 2BH-2EH 範圍內找到相對應的位元，透過 ORL 把那個位元填入 1。

CYCLENEXTITERATIONDEAD:
MOV A, 037H
MOV R0, #2BH
CYCLENEXTITERATIONDEADCHECKCOLUMN:
CJNE A, #00000001B,
CYCLENEXTITERATIONDEADNEXTCOLUMN
JMP
CYCLENEXTITERATIONDEADCOLUMNCHECKED
	啊如果計算出來的結果是死的話，透過前面的方法，在 2BH-2EH 範圍內找到相對應的位元，透過 ANL 把那個位元填入 0。

CYCLENEXTITERATIONDEADNEXTCOLUMN:
RR A
INC R0
JMP CYCLENEXTITERATIONDEADCHECKCOLUMN
CYCLENEXTITERATIONDEADCOLUMNCHECKED:
MOV A, 038H
CPL A
ANL A, @R0
MOV @R0, A
 	
JMP CYCLEENDSTORENEWITERATION	

CYCLEENDSTORENEWITERATION: JMP CYCLENEXTROW	填完一個細胞下一輪的死活後，換下一個細胞。

// temp function for neighbor
CYCLEPUTAINTO03AHRL:
CJNE A, #00100000B, CYCLENORMALPUTAINTO03AHRL
MOV 03AH, #00000001B
RET
CYCLENORMALPUTAINTO03AHRL:
MOV 03AH, A
RET
CYCLEPUTAINTO03AHRR:
CJNE A, #10000000B, CYCLENORMALPUTAINTO03AHRR
MOV 03AH, #00010000B
RET
CYCLENORMALPUTAINTO03AHRR:
MOV 03AH, A
RET
 	
CYCLEPUTAINTO03BHRL:
CJNE A, #10000000B, CYCLENORMALPUTAINTO03BHRL
MOV 03BH, #00000001B
	一些下面會用到的CALLs，只要輸進去的範圍為非定義的行與列，會根據是透過 RR 還是 RL 自動調整。

RET
CYCLENORMALPUTAINTO03BHRL:
MOV 03BH, A
RET
CYCLEPUTAINTO03BHRR:
CJNE A, #10000000B,
CYCLENORMALPUTAINTO03BHRR
MOV 03BH, #01000000B
RET
CYCLENORMALPUTAINTO03BHRR:
MOV 03BH, A
RET	

CYCLEADJUSTNEIGHBORBIT:
// 03AH <- 037H // column
// 03BH <- 038H // row
CJNE A, #0, CYCLEBITNOT0
// Neighbor ID: 0
MOV A, 037H
RR A
CALL CYCLEPUTAINTO03AHRR
 	
MOV A, 038H
RR A
CALL CYCLEPUTAINTO03BHRR
 	
JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT0:
CJNE A, #1, CYCLEBITNOT1
// Neighbor ID: 1
MOV 03AH, 037H
 	
MOV A, 038H
RR A
CALL CYCLEPUTAINTO03BHRR
 	
JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT1:
CJNE A, #2, CYCLEBITNOT2
	透過鄰居的 ID 與目前計算的細胞位置，得到要探討的鄰居細胞位置，並把結果存在 03AH 與 03BH，以只有一個 1 的格式儲存。

```
   // Neighbor ID: 2
   MOV A, 037H
   RL A
   CALL CYCLEPUTAINTO03AHRL

   MOV A, 038H
   RR A
   CALL CYCLEPUTAINTO03BHRR

   JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT2:
   CJNE A, #3, CYCLEBITNOT3
   // Neighbor ID: 3
   MOV A, 037H
   RR A
   CALL CYCLEPUTAINTO03AHRR

   MOV 03BH, 038H

   JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT3:
   CJNE A, #4, CYCLEBITNOT4
   // Neighbor ID: 4
   MOV A, 037H
   RL A
   CALL CYCLEPUTAINTO03AHRL

   MOV 03BH, 038H

   JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT4:
   CJNE A, #5, CYCLEBITNOT5
   // Neighbor ID: 5
   MOV A, 037H
   RR A
   CALL CYCLEPUTAINTO03AHRR

   MOV A, 038H
```

RL A
CALL CYCLEPUTAINTO03BHRL
 	
JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT5:
CJNE A, #6, CYCLEBITNOT6
// Neighbor ID: 6
MOV 03AH, 037H
 	
MOV A, 038H
RL A
CALL CYCLEPUTAINTO03BHRL
 	
JMP CYCLEINITGETTINGPOINTER
CYCLEBITNOT6:
// Neighbor ID: 7
MOV A, 037H
RL A
CALL CYCLEPUTAINTO03AHRL
 	
MOV A, 038H
RL A
CALL CYCLEPUTAINTO03BHRL
; JMP CYCLEINITGETTINGPOINTER	

CYCLEINITGETTINGPOINTER:
// left up
MOV A, 03AH
MOV R0, #20H
CYCLECHECKPOINTER:
CJNE A, #00000001B,
CYCLEPOINTERNOTREACH
JMP CYCLEPOINTERREACH
CYCLEPOINTERNOTREACH:
RR A
INC R0
JMP CYCLECHECKPOINTER
CYCLEPOINTERREACH:
MOV A, @R0
	透過前面的方法，把存取資料的記憶體位置透過 03AH 的線索找到，把資料讀取出來後，透過和 03BH AND 起來，得到目標位元的採樣。

ANL A, 03BH	
CJNE A, #00000000B, CYCLECALCULATINGBITISONE JMP CYCLECALCULATINGBITNOTONE CYCLECALCULATINGBITISONE: INC 039H JMP CYCLEAFTERCALCULATINGBIT CYCLECALCULATINGBITNOTONE: ; JMP CYCLEAFTERCALCULATINGBIT	如果是 1，把 039H 加一。其他的話不做任何事情。
CYCLEAFTERCALCULATINGBIT: INC 03CH JMP CYCLECHECKVALIDATENEIGHBORID JMP CYCLENEXTROW	結束這個鄰居的採樣之後，透過對鄰居 ID 加一，換下一個鄰居，下面那個 YCLENEXTROW 應該是寫錯了。

ENDCYCLE:
MOV 020H, 02BH
MOV 021H, 02CH
MOV 022H, 02DH
MOV 023H, 02EH
MOV 024H, 02FH
	當全部的細胞都算完後，把暫存在 02BH-02EH 存回去 020H-024H

CYCLEUPDATELCDINFO: INC 047H	首先，把回合數加一。

CYCLECHANGELCDINFO:
PUSH 082H
PUSH 083H
// Round Number
MOV A, #10000010B
CALL USEATOCOMMANDLCD
MOV A, 047H
ORL A, #10000000B
CALL WRITEATOLCDNUMBER
// Alives Number
MOV A, #10001000B
CALL USEATOCOMMANDLCD
MOV A, 043H
CALL WRITEATOLCDNUMBER
	接著把回合數、存活數丟到 LCD 上。

// delta
MOV A, #10001101B
CALL USEATOCOMMANDLCD
MOV A, 043H
SUBB A, 048H
MOV 049H, A // delta
JC DELTAHASCARRY
	算運算完後存活數的差距，用 SUBB，因為可能會有負數，所以要檢查是否有 carry。

DELTANOTHAVECARRY:
MOV B, A
MOV A, #43 // +
CALL USEASETTINGDATA
MOV A, B
CALL WRITEATOLCDNUMBER
JMP CALCULATEDELTASCORE
DELTAHASCARRY:
MOV B, A
MOV A, #45 // -
CALL USEASETTINGDATA
MOV A, B
CPL A
INC A
CALL WRITEATOLCDNUMBER
MOV 055H, #0
JMP DISPLAYDELTASCORE
	如果沒有 carry，在 LCD 上放上”+”，並且把數字渲染上去。如果沒有 carry，在 LCD 上面放上”-”，並且取二的補數，把所有位元翻轉再加一，並且把數字渲染上去。

CALCULATEDELTASCORE:
// TODO: use convertAtoDecimal call
// 47H round 49H delta
MOV A, 047H
MOV B, 049H
MUL AB
MOV 055H, A
// 0 - 255 process
CALL CONVERTAINTODECIMAL
MOV A, 052H
ADD A, 04CH
MOV 052H, A
	最後，要計算得到的分數，每次計算得到的分數為回合數乘上這回合增加的細胞數量。(失去更多的話得分為 0 不到扣很佛)。將得到的乘積依序跟總分數相加。最後，因為每個位數都有機會超過十，透過 SCORECARRYCHECK 會將總分以十進

MOV A, 053H
ADD A, 04DH
MOV 053H, A
 	
MOV A, 054H
ADD A, 04EH
MOV 054H, A
 	
CALL SCORECARRYCHECK
	位的方法排除這個問題。

DISPLAYSCORE:
MOV A, #11000010B
CALL USEATOCOMMANDLCD
MOV R0, #04FH
DISPLAYSCORENEXTNUMBER:
INC R0
MOV A, @R0
ADD A, #48
CALL USEASETTINGDATA
MOV A, R0
CJNE A, #054H,
DISPLAYSCORENEXTNUMBER
JMP DISPLAYDELTASCORE
	得到目前的總分後，把 50H-54H，的值，依序放進 LCD 中。

DISPLAYDELTASCORE:
MOV A, #11001011B
CALL USEATOCOMMANDLCD
MOV A, 055H
ORL A, #10000000B
CALL WRITEATOLCDNUMBER
	總分後面放著這回合到底加了多少總分。

ENDDELTA: POP 083H POP 082H	暫時不用 DPTR 了，把 DPTR 還回去。
CHANGESEQUENCEINIT: MOV A, 049H	接下來是根據存活的變化，更改旋律。
CHANGESEQUENCEIS0: CJNE A, #0, CHANGESEQUENCENOT0 // 0 MOV DPTR, #BUZZERSEQUENCETABLE0	生出來的細胞小於死掉的細胞數量所撥 2 號旋律，生出來等於死掉的話撥

JMP ENDCHANGESEQUENCE
CHANGESEQUENCENOT0:
CJNE A, #1, CHANGESEQUENCENOT1
// 1
MOV DPTR, #BUZZERSEQUENCETABLE4
JMP ENDCHANGESEQUENCE
CHANGESEQUENCENOT1:
ANL A, #10000000B
CJNE A, #00000000B,
CHANGESEQUENCENOT2
// 1+
MOV DPTR, #BUZZERSEQUENCETABLE5
JMP ENDCHANGESEQUENCE
CHANGESEQUENCENOT2:
// -
MOV DPTR, #BUZZERSEQUENCETABLE2
; JMP ENDCHANGESEQUENCE
	0 號旋律，生出來比死掉的多一個撥
	4 號旋律，生出來大於死掉並且不只一個撥 5 號旋律。

ENDCHANGESEQUENCE:
// reset timer
MOV 03EH, #96
	更改完旋律後，從第一個音符(共 96 個)開始撥。

ENDCYCLENOUPDATE: POP 0F0H // B POP 00H // R0 POP 0E0H // A ; JMP ENDONEMS	生命遊戲一個 Cycle 結束

ENDONEMS:
// Reset Timer
// 1ms
MOV TH0, #252
MOV TL0, #23
RETI
	重新設定一微秒 overflow 的初始值。

// --- global function ---	下面是整個程式共用的一些副程式。

USEATOCOMMANDLCD:
MOV P3, #00000000B
MOV P3, A
// set mode
ANL 080H, #10011111B // P0 write command
	把指令寫在 A 裡，CALL 這個 function 可以設定 LCD。

ORL 080H, #10000000B // enable CALL LCDDELAY ANL 080H, #01111111B CALL LCDDELAY RET	

USEASETTINGDATA:
MOV P3, #00000000B
MOV P3, A
// set mode write into ram (ddram or cgram)
ORL 080H, #00100000B
ANL 080H, #10111111B
// enable
ORL 080H, #10000000B
CALL LCDDELAY
ANL 080H, #01111111B
CALL LCDDELAY
RET
	如果要設定要寫入 DDRAM 的位置或是 CGRAM 的位置，先用 USEATOCOMMANDLCD。設定完後把要寫進記憶體的資料放進 A 並且 CALL USEASETTINGDATA

LCDDELAY:
MOV 044H, #010H
LCDDELAY1:
MOV 045H, #0FFH
LCDDELAY2:
DJNZ 045H, LCDDELAY2
DJNZ 044H, LCDDELAY1
RET
	在給指令給 LCD 時，給 LCD 的反應時間。

CLEARLCD: MOV A, #00000001B CALL USEATOCOMMANDLCD MOV A, #10000000B // set ddram memory pointer to 0 CALL USEATOCOMMANDLCD RET	把 LCD 清空，DDRAM 與 AC。

WRITEDPTRTOLCD:
STARTWRITEDPTRLCD:
MOV 046H, #0
CHECKWRITEDPTRLCD:
MOV A, 046H
	把 DPTR 設定好後，CALL 這個 function 會把 DPTR 寫進 LCD 中，DPTR 指定的 Table

MOVC A, @A+DPTR CJNE A, #0, WRITEDPTRDATATOLCD RET WRITEDPTRDATATOLCD: CALL USEASETTINGDATA INC 046H JMP CHECKWRITEDPTRLCD	最後面要設為 0，以便讓程式知道什麼時候停止。
LCDINITTEXT: DB "Edit Mode", 0 LCDINITTEXTROW2: DB "C:0|R:0|B", 0 LCDGAMEOFLIFETEXTROW0: DB "R:000|A:00|D:+00", 0 LCDGAMEOFLIFETEXTROW1: DB "S:00000|SD:+00", 0	一些存有初始化 LCD 文字的 TABLE

// MSB #1: 3num #0: 2num
WRITEATOLCDNUMBER:
STARTWRITEATOLCDNUMBER:
MOV B, A
ANL A, #10000000B
CJNE A, #00000000B,
WRITINGATOLCD3NUMBER
MOV A, B
JMP WRITINGATOLCD2NUMBER
WRITINGATOLCD3NUMBER:
MOV A, B
ANL A, #01111111B
MOV B, #100
DIV AB
ADD A, #48
CALL USEASETTINGDATA
MOV A, B
WRITINGATOLCD2NUMBER:
ANL A, #01111111B
MOV B, #10
DIV AB
ADD A, #48
CALL USEASETTINGDATA
	把二進位的 A 轉成十進位並且放到 LCD 上，如果 MSB 是 1 會把數字變成三個十進位的數字，MSB 為 0 的話把數字變成兩個十進位的數字。

MOV A, B
ADD A, #48
CALL USEASETTINGDATA
RET	

// 4C, 4D, 4E, A will change
CONVERTAINTODECIMAL:
PUSH 0F0H // B
MOV B, #100
DIV AB
MOV 04CH, A
 	
MOV A, B
MOV B, #10
DIV AB
MOV 04DH, A
MOV 04EH, B
POP 0F0H // B
RET
	將 A 存的東西變成三個十進位的值並分別存進 04CH，04DH 與 04EH。

// 50 51 52 53 54 MSB to LSB carry check
SCORECARRYCHECK:
PUSH 0E0H // A
PUSH 0F0H // B
PUSH 000H // R0
PUSH 001H // R1
MOV R0, #054H
MOV R1, #053H
NEXTCARRYCHECK:
MOV A, @R0
SCORECARRYCHECKAMINUS:
MOV B, A
SUBB A, #10
JC ENDSCORECARRYCHECK // if cannot minus 10
INC @R1
JMP SCORECARRYCHECKAMINUS
ENDSCORECARRYCHECK:
MOV @R0, B
SCORECARRYCHECKNEXTITERATION:
	讀取 050H-054H 的值，如果有大於十的，進位。每位的值的範圍 0-255，沒有什麼限制，Overflow 的話會直接丟棄。

DEC R0 DEC R1 MOV A, R1 CJNE A, #049H, NEXTCARRYCHECK RETCARRYCHECK: POP 001H POP 000H POP 0F0H // B POP 0E0H // A RET	

// A in A out callable
TURN1LOCINTONUMBER:
PUSH 000H // R0
MOV R0, #0
TURN1LOCINTONUMBERCHECKA:
CJNE A, #00000001B,
TURN1LOCINTONUMBERNEXTITERATION
MOV A, R0
POP 000H // R0
RET
TURN1LOCINTONUMBERNEXTITERATION:
RR A
INC R0
JMP TURN1LOCINTONUMBERCHECKA
	讀取 A 的值，將 #00000001B 變成 0，#00000010B 變成 1，#10000000B 變成 7，etc。算完的資料會丟回 A。

END	40 頁的 code，好多啊

五、心得：

做期末專題真的好困難壓，沒有 function，沒有變數，沒有好用的 Google，y 資料好少，只有記憶體，非直接存取，邏輯運算，單個 bit 的運算，PSW，JMP，CJNE 與 DJNZ，ADD SUBB 與 MUL 和 DIV，最後是 TABLE，當程式很大的時候，真的不知道要從哪邊開始看，各種 JMP 連接程式的每一個區塊，腦子真的好痛。真的，有生之年不想再寫組合語言了。PUSH 跟 POP 真的是超好用的，當有想要用但目前正在使用的暫存器的話，PUSH 與 POP 就有與世隔絕的效果啦。當然壞處跟沒刪 C++動態陣列一樣，永遠會有機會 PUSH 完沒有 POP，導致 Stack Pointer 跑到非常奇怪的地方。

我倒是找到一個非常好用的東西，除了可以手動設定 Breakpoint 之外，還可以透過點擊執行旁邊的紅色叉叉手動停止程式。停止後，下面原本預設的 C:000H 只要改成 D:開頭(直接存取記憶體)或是 I:開頭(間接存取記憶體)，後面輸入要查看的記憶體位置，就會直接告訴你目前那個記憶體位置存了什麼東西歐，找錯真的超方便的啊。

當然，有了這個神器，還是有極度煩人的硬體 bug，像是當 ICE 接到電腦蜂鳴器會使的 LED 亮不起來，把蜂鳴器關掉 LED 就亮得起來，5*7 點陣 LED 沒有預先設為#00000000B 會導致之後 LED 完全亮不起來。LCD 預先一些設定不對導致字根本顯示不出來等等。不，不想在重新做這個東西了。

成效呢? 不敢說他是什麼曠世巨作，但是它看起來很酷。