Introducció:

L'objectiu d'aquesta pràctica és conèixer un sistema típic de desenvolupament basat en microcontrolador així com els procediments per desenvolupar i depurar aplicacions senzilles.

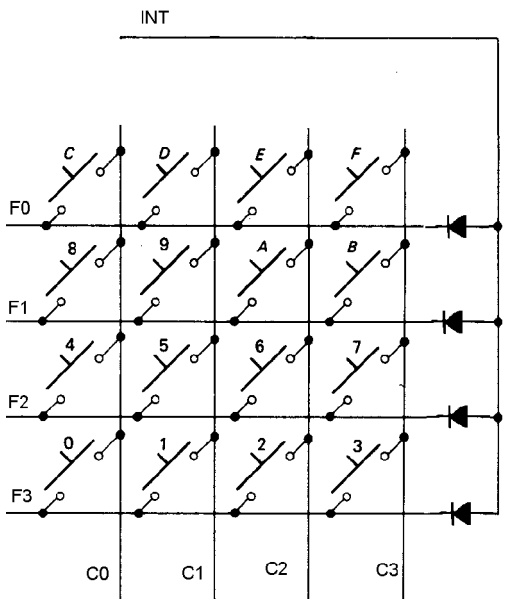
Les eines dels quals disposarem al laboratori són:

> ***Sistema de desenvolupament DS-552:*** Sistema microprocessat basat en el 80552 de Philips. Permet la càrrega de programes i la seva execució, via port sèrie, des d'un PC per tal "d'agilitzar" el desenvolupament basat en la estratègia "assaig i error" evitant el temps d'esborrat/programació d'EPROMS. Disposeu de la documentació redactada per l'autor d'aquest sistema (Josep Reixach). Annexat a aquest document (enunciat de la pràctica) hi ha un esquema auto-explicatiu dels perifèrics afegits i la interconnexió d'aquests amb la placa DS-552.
>
> ***Aplications Builder de Intel:*** Aplicació per tal de configurar els perifèrics integrats en un microcontrolador de Intel (entre ells la família MCS'51) així com efectuar consultes sobre l'arquitectura d'aquests (joc d'instruccions, registres, modes de treball...). Hi ha un petit fitxer de text per "iniciar-se" en el funcionament d'aquest programa (*ApBldr.txt*).
>
> Eines de compilació, enllaçat i simulació ja conegudes de la pràctica 1.

Enunciat de la pràctica.

A partir de les eines descrites anteriorment es crearan i s'executaran (simulant prèviament els blocs que creieu convenients) en el sistema DS-552 diferents programes que actuïn sobre les sortides "LED i brunzidor" a partir de les entrades de teclat.

- Per simplificar aquesta pràctica, només comptarem com entrades les tecles (quatre) d'una sola fila del teclat. L'estructura del ***teclat*** matricial és la que hi ha a la següent figura:



Amb aquest maquinari posarem zeros a totes les columnes (fent “CLR PX.i”) i llegirem les pulsacions de les tecles d’una fila. En cas de tecla “premuda” aquest valor serà zero (fent “JNB PX.i,tecla_n_premuda”). Per tal d’evitar lectures “falses” degut als rebots, la estratègia que seguirem és la següent: En cas de detectar una pulsació (llegit un zero), esperarem 10 ms i tornarem a llegir. Dues lectures idèntiques consecutives són “pulsació vàlida” i, només en aquest cas, actuarem en conseqüència. Es evident que la línia d’interrupció associada al teclat serà ignorada (deshabilitada al registre IE).

- ***Els LEDs*** de la placa s’encenen amb un 1 lògic a la sortida corresponent (recordem que al engegar tots els ports del micro són a “FF” i per tant els LEDs encesos). Per tal d’apagar un LED, només cal fer: “CLR Px.i”.

Es important habituar-se a estructurar el codi (subrutines i macros) per tal d’aconseguir una sèrie de funcions “de llibreria” que ens permetran, en successives pràctiques, abordar problemes més complexes (en quant a mida) recolzats en la feina feta amb anterioritat.

Annexat a aquesta pràctica hi ha el llistat de les directives de compilació del assemblador A8051 de IAR.

Un cop “domineu” les pulsacions de tecla, tingueu “controlades” rutines de retard (com les que hem comentat a classe) per tal de temporalitzar l’execució del codi a nivell “humà” (canvis en períodes de l’ordre de dècimes de segon) i governeu els LEDs a voluntat feu un programa que atenen a certes tecles, canvii seqüències de funcionament dels LEDs (Deixem la especificació al vostre criteri per tal d’afavorir la diversitat d’idees fresques i noves).

Ex. Si premem ‘1’, fa “cotxe fantàstic”. Si es prem ‘2’, s’entra en mode “seguiment”; això és, s’encendrà el LED corresponent a la tecla (de la ‘1’ a la ‘3’). Si es prem ‘*’, s’apaga tot i resta aturat fins nova comanda (de les anteriors).

Recomanacions:

Feu el plantejament del algorisme seguint una estratègia “top-down” (definiu el codi de dalt a baix, suposant resoltes certes funcions que posteriorment anireu resolent) i aneu-lo implementant en sentit “down-top” (de baix a dalt) en “porcions”. Proveu aquestes “porcions” amb programes “mono-tasques” (sobre el simulador o bé directament al maquinari). Un cop verificat el correcte funcionament d’una rutina (o macro) podeu passar a un nivell “superior”.

Ex. De “algorisme”

```
REPETIR
   SI (tecla_pitjada) LLAVORS
      SI (Mode=1)  LLAVORS
         Encen_led(tecla)
      ALTRAMENT
         Mode=2
      FSI
   FSI
 FINS (MAI)
```

Ex. Fem una macro per encendre qualsevol LED, la definim així (dins “MISMACROS.INC” o qualsevol nom que vulguem donar al fitxer)

```
MACRO %encen_led
```

```
    SETB   \0    ;aquest és el primer (i únic) paràmetre que passarem
    ENDMAC

LED1    EQU   T1     ;Pin P3.5
LED2    EQU   T0     ;Pin P3.4
LED3    EQU   INT1   ;en algun lloc hi ha: "INT1   EQU   P3.3"
LED4    EQU   P1.2   ;fixeu-vos en les possibilitats que ofereixen els símbols.
```

Dins el nostre programa:

```
$SFR8052.INC        ;això inclou definicions del 51, podeu examinar el fitxer
$MISMACROS.INC      ;això inclou el fitxer amb totes les definicions

Inici:   %encen_led   LED1    ;això expandeix encen_led passant com a paràmetre LED1
         %apaga_led   LED2    ;suposem definida
          CALL    retard      ;per tal d'esperar uns ms
         %encen_led   LED2
         %apaga_led   LED1
         AJMP         Inici
         END
```

En el procés d'assemblatge, les referències a les macros (%nom) s'expandiran "replicant" el codi expressat en la definició i substituint el/s paràmetres (de "\0" a "\9") per l'expressió indicada.

Examineu els llistats generats (per veure la expansió de les macros). Al laboratori hi ha un exemplar del manual del Assemblador i l'enllaçador. Consulteu-lo per ampliar i/o complementar aquesta informació.

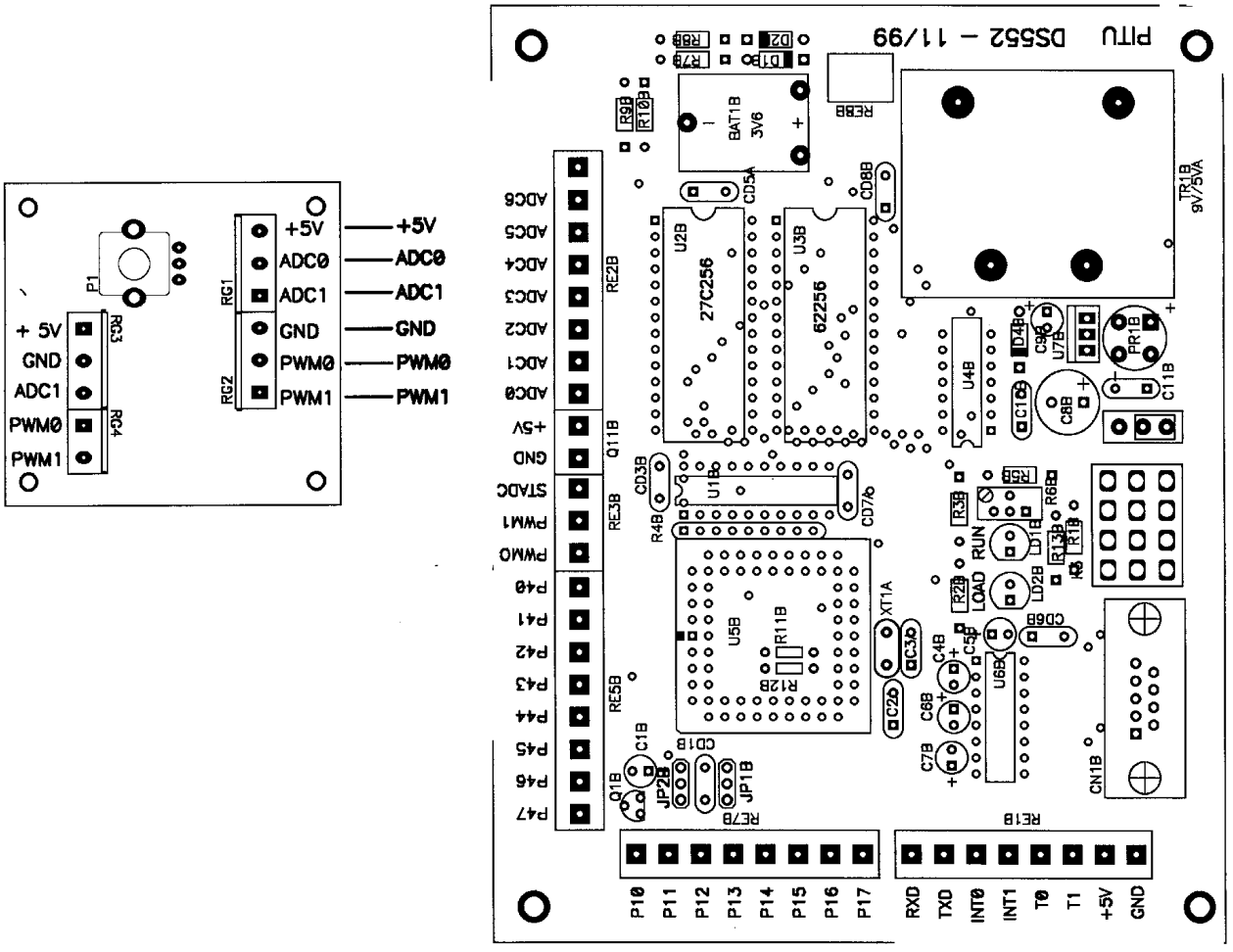
Conclusions:

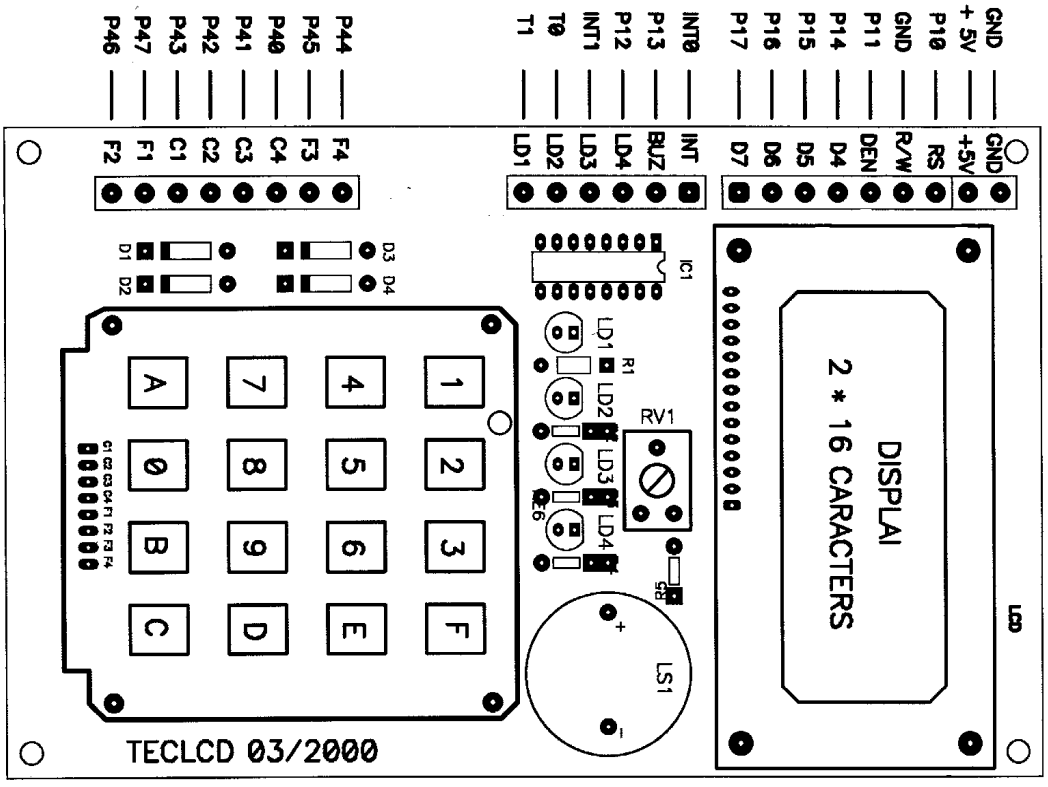
En acabar aquesta pràctica ja sou capaços d'actuar sobre "elements binaris" segons un "criteri" combinacional/seqüencial. No es requereixen càlculs més "sofisticats" que temporitzacions de l'ordre de segons.

Amb aquest "domini" podeu afrontar qualsevol automatització senzilla afegint els elements perifèrics necessaris (relés, sensors, pulsadors..). Aquests perifèrics són per adaptar els senyals que dona/rep el micro (TTL 0..5V) al nivell requerit (220V A.C. per motors...).

Dins aquesta categoria hi podem incloure: portes de garatge, ascensors, transport i control de matèries en cadenes de producció, control d'accessos (panys "intel·ligents"), domòtica (persianes que pugen i baixen "soles", calefaccions que s'encenen a una determinada hora, simuladors de presència...) i qualsevol aplicació que es pugui presentar amb aquestes característiques.

Annex I: Perifèrics i interconnexió dels mateixos a la placa DS-552.





Annex II: Directives de l'Assemblador A8051.

DE MODUL

```
NAME          <símbol>        ; comença un mòdul de codi
MODULE        <símbol>        ; comença un mòdul de llibreria
ENDMOD        <expressió>     ; fi d'un mòdul
END           <expressió>     ; fi d'un fitxer de codi font
```

DE SIMBOL

```
PUBLIC                <símbol>        ; els altres mòduls "sabran com/què" <símbol>
EXTERN        <símbol>        ; <símbol> definit en un altre mòdul
LOCKSYM+, LOCSYM-                     ; fer tots/cap síbols públics
```

DE SEGMENT

```
ASEG                                  ; inicia un segment absolut
RSEG          <nom> [(<alineació>)]   ; inicia in segment reubicable
STACK                 <nom> [(<alineació>)]   ; inicia in segment de pila
COMMON        <nom> [(<alineació>)]   ; inicia in segment comú
ORG           <expressió>             ; especifica el comptador de posició
```

DE VALOR

```
<etiq>  EQU     <expressió>     ; un valor "d'abast modular (en el que apareix)"
<etiq>  =       <expressió>     ; expressió alternativa d'EQU
<etiq>  SET     <expressió>     ; un valor redefinible
<etiq>  DEFINE          <expressió>     ; un valor "d'abast global (tot el ftxer font)"
```

DE DADES

```
DS      <expressió>             ; defineix bytes no-inicialitzats
DB      <valor(s) byte>         ; defineix bytes inicialitzats
DW      <valor(s) word>         ; defineix paraules (16bit) inicialitzades
DD      <valor(s) dword>        ; defineix paraulotes (32bit) inicialitzades
```

D'ASSEMBLATGE CONDICIONAL

```
IF      <condició>      ; inicia un bloc d'assemblatge condicional
ELSE                    ; polaritat condicional alternativa
ENDIF                   ; finalitza un bloc d'assemblatge condicional
```

D'EXPANSIO DE CODI FONT

```
$<fitxer d'inclusió>            ; insereix un fitxer d'inclusió
MACRO         %<nom>                    ; inicia la definició d'una macro
ENDMAC                          ; finalitza la definició d'una macro
%<nom> [<paràmetres>]           ; invoca l'expansió d'una macro
```

DE LLISTAT

```
...
PAGSIZ          <expressió>     ; insereix un canvi de plana després de <expressió> linies
TITL    '<cadena>'      ; titula les planes amb <cadena>
STITL   '<cadena>'      ; subtitula les planes amb <cadena>
...
```

Annex III: Format Intel Hex Standart (.A03 o bé .HEX)

Extret de l'Apèndix I del Datasheet d'un AT89C52 de Dallas.

Cada registre (línia) conté els següents camps:

<:><longitud de registre><adreça a la que es carregarà><tipus de registre><dades..><xecsum>

- Els dos punts (:) són la capçalera de registre.
- El camp "longitud de registre" consisteix en dos dígits hexadecimals i representa el nombre d'entrades del camp dades. Sortides del tipus 0H contenen 16 o menys entrades de dades.
- El camp "adreça de càrrega" consisteix en quatre dígits hexadecimals i indiquen l'adreça absoluta en la que els camp de dades s'ha de carregar.
- El camp "tipus de registre" són dos dígits hexadecimals que són sempre zero en registres de dades.
- El camp "dades" conté de un a setze parells de dígits hexadecimals.
- Els dos últims dígits són un xecsum sobre la longitud, adreça, tipus i dades. La suma dels equivalents binaris d'aquests camps i el xecsum és zero.
- Cada registre dins el fitxer, s'acaba amb un "retorn de carro" (0x0D) i un "avanç de línia" (0x0A).
- Un registre de tipus "1" marca el final de fitxer. Sempre conté el següent valor: ":0000001FF".

Table 2. A List of the MCS®-51 Arithmetic Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ADD A,<byte>	A = A + <byte>	X	X	X	X	1
ADDC A,<byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A,<byte>	A = A − <byte> − C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A − 1	Accumulator only				1
DEC <byte>	<byte> = <byte> − 1	X	X	X		1
MUL AB	B:A = B x A	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

Table 3. A List of the MCS®-51 Logical Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ANL A,<byte>	A = A .AND. <byte>	X	X	X	X	1
ANL <byte>,A	<byte> = <byte> .AND. A	X				1
ANL <byte>,#data	<byte> = <byte> .AND. #data	X				2
ORL A,<byte>	A = A .OR. <byte>	X	X	X	X	1
ORL <byte>,A	<byte> = <byte> .OR. A	X				1
ORL <byte>,#data	<byte> = <byte> .OR. #data	X				2
XRL A,<byte>	A = A .XOR. <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> .XOR. A	X				1
XRL <byte>,#data	<byte> = <byte> .XOR. #data	X				2
CRL A	A = 00H	Accumulator only				1
CPL A	A = .NOT. A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

Table 9. Conditional Jumps in MCS®-51 Devices

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A ≠ 0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A ≠ <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> ≠ #data		X	X		2

Table 4. A List of the MCS®-51 Data Transfer Instructions that Access Internal Data Memory Space

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
MOV A,<src>	A = <src>	X	X	X	X	1
MOV <dest>,A	<dest> = A	X	X	X		1
MOV <dest>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	DPTR = 16-bit immediate constant.				X	2
PUSH <src>	INC SP : MOV "@SP",<src>	X				2
POP <dest>	MOV <dest>, "@SP" : DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles		X			1

Table 5. A List of the MCS®-51 Data Transfer Instructions that Access External Data Memory Space

Address Width	Mnemonic	Operation	Execution Time (μs)
8 bits	MOVX A,@Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri,A	Write external RAM @Ri	2
16 bits	MOVX A,@DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @DPTR	2

Table 6. The MCS®-51 Lookup Table Read Instructions

Mnemonic	Operation	Execution Time (μs)
MOVC A,@A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A,@A+PC	Read Pgm Memory at (A+PC)	2

Table 8. Unconditional Jumps in MCS®-51 Devices

Mnemonic	Operation	Execution Time (μs)
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

Table 7. A List of the MCS®-51 Boolean Instructions

Mnemonic	Operation	Execution Time (μs)
ANL C,bit	C = C .AND. bit	2
ANL C,/bit	C = C .AND. .NOT. bit	2
ORL C,bit	C = C .OR. bit	2
ORL C,/bit	C = C .OR. .NOT. bit	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = .NOT. C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1; CLR bit	2