Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

O L*CD* implementa uma comunicação com o *software* e com o módulo *IOS* como se verifica na figura 1.

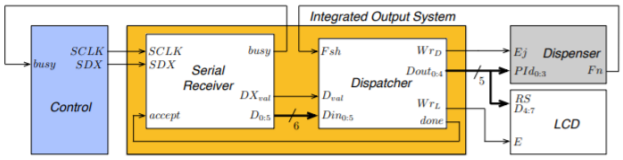
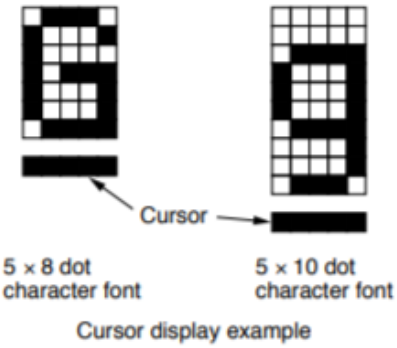


Figura 1 – Diagrama de blocos do módulo *IOS* que demonstra a sua ligação com o LCD

1 Estrutura Interna do LCD

O *LCD (Liquid Crystal Display)* é um visor constituído por inúmeras matrizes de pontos, na qual cada matriz é usada para gerar um caractere. Neste caso foi-nos disponibilizado um *LCD* que apresenta 32 matrizes (sendo 16 na linha de cima e 16 na linha de baixo) com 5 colunas por 8 linhas o que permite a escrita máxima de 16 caracteres por linha (32 no total).
Na sua parte interna o *LCD* contém um *chip* que tem as funções de *displayRAM*, *refresh* e gerar os caracteres. Neste caso o *chip* permite a sua utilização numa interface simples de 4 ou 8 *bits*. Como se verifica na primeira figura o *control* irá definir quais os *bits* a ser entregues apesar de antes de chegar ao *LCD* passar pelo *IOS*. Neste caso utilizámos a interface a 4 *bits* sendo estes os de maior peso (*bits D4* a *D7*).



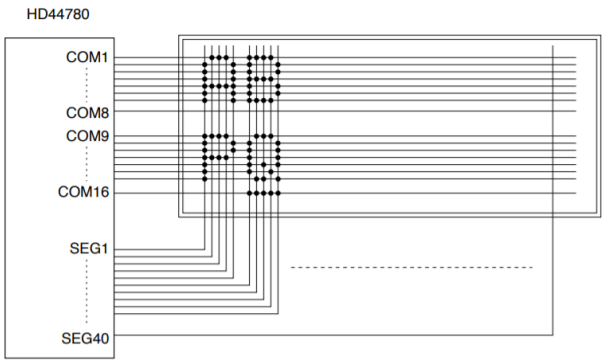


Figura 2 – Exemplo do display de 8 por 5 em comparação com um de 10 por 5 e da sua ligação interna no *LCD*

Já com os *bits* de informação necessários identificados, também é necessário a utilização de mais 2 *bits* de modo a controlar o *LCD* sendo estes o *Register Select* e o *Enable* (pins 4 e 6 respetivamente). Devido a só se querer escrever no *LCD*, este fica permanentemente a 0 de modo a deixar *Write* sempre ativo.

O *LCD* que estamos a utilizar tem como seu controlador o HD44780U, sendo por isso necessário respeitar as instruções de acordo com este processador de modo a funcionem como o nosso *LCD*.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

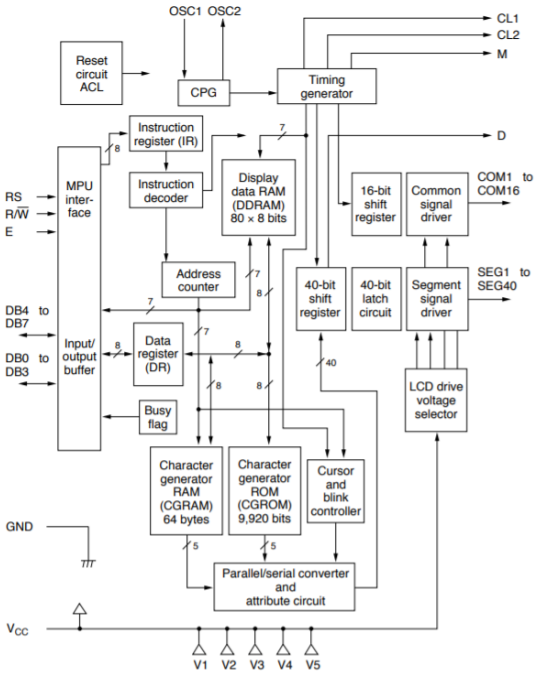


Figura 3 - Diagrama de blocos do controlador HD44780U

De acordo com a figura 3 para se escrever no LCD, numa posição previamente escolhida, o controlador precisa de realizar uma instrução que o permita colocar o *Address counter* na *Display Data RAM*, na posição designada. Sendo que o *Display Data RAM* está contido a informação em *ASCII* necessária para um caractere, é necessário mudar o valor do *Register Select* de forma a escrever no *Data Register*, ligar o *Enable e* colocar o código ASCII nos bits de dados correspondentes.

Terminada a interação entre o controlador e o LCD, os caracteres só irão aparecer no *display* depois da CGROM gera a matriz que corresponde ao código ASCII do caracter com endereço na DDRAM. Também é possível definir símbolos utilizando a CGRAM (caso se queira criar um símbolo específico).

2 Forma de utilização do LCD

Como anteriormente referenciado, o controlador do *LCD* permite uma interface com 4 ou 8 *bits*. Sendo que no nosso projeto temos implementado uma interface de quatro *bits* é impossível enviar 8 *bits* de uma só vez tendo por isso de se fazer de 4 em 4 enviando assim dois *Nibbles* (um seguido do outro)

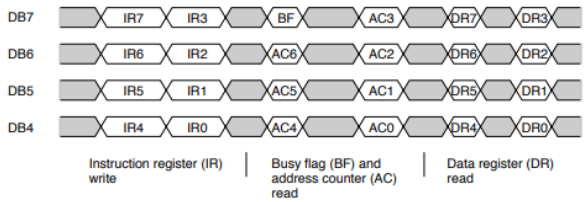
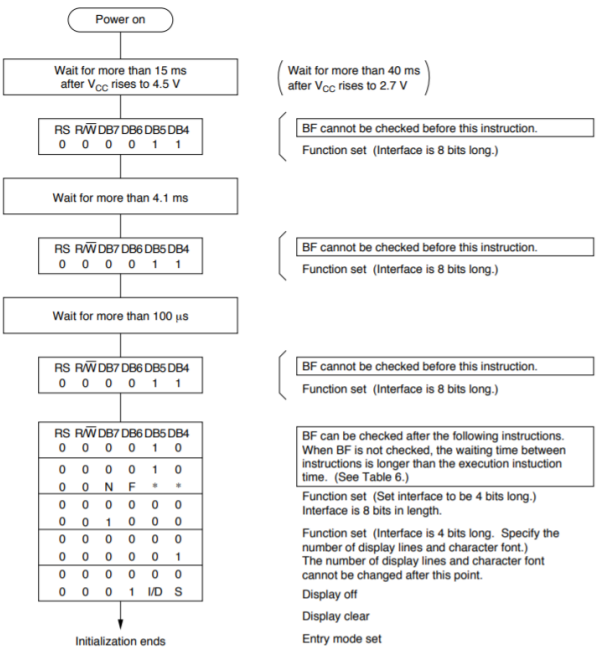


Figura 4–Exemplo de transferência a 4 *bits*

Como se verifica na Figura 4 o primeiro *Nibble* tem de ser o com os *bits* mais significativos e é na transição descendente do *Enable* que o LCD regista os dados sendo assim é necessário garantir a estabilidade dos dados quando *Enable* está a 1 de modo a evitar informações erradas ou incompletas no *display*. Para o envio do Nibble dos bits menso significativos o Enable é novamente ativo, quando ativo modifica os bits e de seguida destiva de novo de modo a registar os mesmos no LCD transmitindo assim um comando ao LCD com suceso.

Sendo que o LCD permite duas interfaces distintas é necessário primeiro saber em qual delas se está a trabalhar (em 4 bits ou em 8 bits) sendo assim existe uma sequência de iniciação especifica para cada interface.
Neste caso foi escolhida a iniciação a 4 bits.



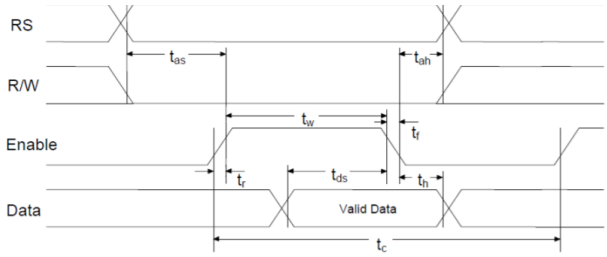
Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Figura 5 – Sequencia de iniciação da interface a 4 *bits*

Recorrendo às informações da figura 5 verificamos que para se enviar um comando é necessário ter o bit RS desativo (com valor 0).

Para iniciar o LCD com a interface a 4 bits é necessário enviar 3 comandos com o número 3 e de seguida um com o número 2 definindo a interface a 4 bits. Com este último comando o LCD já identifica a interface de 4 bits fazendo com que os próximos4 comandos sejam constituídos por 2 *Nibbles*. Estes comandos têm parâmetros que permitem ao utilizador definir o *display* que pretender. Verifica-se também que no fim da iniciação o display não estará ligado.

Sendo que o LCD é iniciado com a escrita de vários *Nibbles*, é importante entender que estes não podem ser escritos aleatoriamente o que levou à necessidade de entender o ciclo de escrita do *Nibble*.



Write cycle

Parameter	Symbol	Min [1]	Typ [1]	Max [1]	Unit
Enable Cycle Time	t_c	500	-	-	ns
Enable Pulse Width (High)	t_w	230	-	-	ns
Enable Rise/Fall Time	t_r,t_f	-	-	20	ns
Address Setup Time	t_{as}	40	-	-	ns
Address Hold Time	t_{ah}	10	-	-	ns
Data Setup Time	t_{ds}	80	-		ns
Data Hold Time	t_h	10	-	-	ns

Note [1] The above specifications are a indication only. Timing will vary from manufacturer to manufacturer.

Figura 6- Ciclo de escrita de um *Nibble* e tempos especificados.

Quando se observa a figura 6 é possível verificar o tempo em que a informação é válida para a sua escrita e confirmamos mais uma vez que é indispensável manter os bits estáveis (sem sofrerem alterações) durante certos intervalos de tempo.

Durante a implementação de software é necessário ter cuidado com o tempo de *Hold*, o tempo de Setup e o tempo necessário para realizar uma ação e o tempo de espera depois da realização da mesma.

3 Implementação

De modo a implementar o *Software* recorremos às instruções do LCD demonstradas na figura 7.

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, 37 µs × $\frac{270}{250}$ = 40 µs

Note: — indicates no effect.
* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

Figura 7 – Instruções do *LCD*

3.1 *E scrita de um Nibble*

```
// Escreve um nibble de comando/dados no LCDprivate
private fun writeNibble(rs: Boolean, data: Int){
    when(SERIAL_MODE){
        true-> writeNibbleSerial(rs,data)
        else-> writeNibbleParallel(rs,data)
    }
}

// Faz o efeito do writeNibble, mas em serie
private fun writeNibbleSerial(rs: Boolean, data: Int){
    val rsAndData = data shl 1 or if(rs) 1 else 0
    SerialEmitter.send(SerialEmitter.Destination.LCD,rsAndData)
}

// Faz o efeito do writeNibble, mas em paralelo
private fun writeNibbleParallel(rs: Boolean, data: Int){
    //e,rs,outro
    val rsBit = if(rs) RS_MASK else 0
    HAL.writeBits(RS_MASK,rsBit)
    Time.sleep(1)
    HAL.writeBits(RS_MASK+E_MASK,rsBit + E_MASK)
    Time.sleep(1)
    HAL.writeBits(RS_MASK+E_MASK+DATA_BITS_MASK,rsBit+E_MASK+data)
    Time.sleep(1)
    HAL.clrBits(E_MASK)
}
```

Figura 8- Funções de escrita de *Nibble* (*writeNibbleSerial*, *writeNibbleParallel*, *writeNibble*)

Neste projeto como é passado o limite de bits disponíveis é necessário enviar alguns os dados em série e outros será possível de enviar em paralelo.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Sendo assim a função *writeNibble* irá fazer a distinção do qual o tipo de escrita selecionando assim a função *writeNibbleSerial* ou *writeNibbleParallel* dependendo dos parâmetros fornecidos.

3.2 Escrita de um *Byte*

```
// Escreve um byte de comando/dados no LCDprivate
private fun writeByte(rs: Boolean, data: Int){
    writeNibble(rs,data shr HALF_BYTE)
    writeNibble(rs,(data and DATA_BITS_MASK))
}
```

Figura 9 – Função writeByte

Tendo em conta a Figura 7, verificamos que apesar de estarmos a utilizar a interface de 4 bits, os comandos do LCD continuam a ser de 8 bits (1 byte).

Como se observou no esquema da figura 4, verificamos que o primeiro *Nibble* a ser enviado é o mais significativo.

De modo a seguir esta regra tem de se escrever primeiro o *Nibble* mais significativo, chamando assim a função *writeNibble* e aplicando ao valor de Data um Logical Shift Right de modo a este ocupar a parte alta do byte, sendo por isso de 4 bits, chamando assim de seguida a função writeNibble para escrever os 4 bits de menor peso.

3.3 Escrita de um *Comand/Data*

```
// Escreve um comando no LCD
private fun writeCMD(data: Int) = writeByte(false,data)

// Escreve um dado no LCD
private fun writeDATA(data: Int) = writeByte(true,data)
```

Figura 10- Funções writeCMD e writeDATA.

De modo a escrever um comando no LCD é necessário enviar um *Byte* com o *Register Select* a 0 (desativado). Para escrever *Data* no LCD é preciso enviar um *byte* com o valor do *Register Select* a 1 (ativo).

3.4 Sequência de iniciação

```
fun init(){
    Time.sleep(15)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_8BITS)
    Time.sleep(5)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_8BITS)
    Time.sleep(1)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_8BITS)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_4BITS)
    writeCMD(FUNCTION_SET_INTERFACE_4BITS_2LINE_DISPLAY_FONT_5X7_DOTS)
    writeCMD(DISPLAY_OFF)
    writeCMD(CLEAR_DISPLAY)
    writeCMD(ENTRY_MODE_INCREMENT_CURSOR_NO_DISPLAY_SHIFT)
    writeCMD(DISPLAY_ON_CURSOR_ON_BLINK_CURSOR_ON)
}
```

Figura 11-Função *init*

Começando a sequência de iniciação é escrito um tempo de espera de 15ms e a escrita de 3 *Nibbles* com valor 3 e com os tempos de espera de 5ms e 1ms entre eles respetivamente.

Seguidamente é escrito um *Nibble* com valor 2 para o LCD reconhece que a interface é a 4 *bits*.

Com o LCD a reconhecer a interface enviamos o comando com o valor de 0x28, dado que este LCD apresenta 2 linhas, é necessário colocar o bit do parâmetro N a 1 (ativo) e como se quer uma fonte de 5 por 8 é necessário colocar o parâmetro F a 0 (desativo).

Seguidamente enviamos outros 2 comandos, o primeiro a 0x08 para desligar o *display* e o segundo a 0x01 para o limpar.

É depois enviado mais um comando com valor 0x06 de modo a definir o *Entry Mode*, é colocado o I/D a 1 de modo a incrementar o cursor sempre que se escrever e S é posto a 0.

Por fim tendo já o LCD inciado enviamos um comando final com o valor 0x0F de modo a ligar o *display*.

Os valores referidos acima forma todos substituídos por constantes que os representam de modo a aumentar a legibilidade do código (podemos observar estas constantes na figura 12).

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

3.5 Escrita de uma String/Char no LCD

```
// Escreve um caráter na posição corrente.
fun write(c: Char) = writeDATA(c.code)

// Escreve uma string na posição corrente.
fun write(text: String){
    for(element in text) { write(element) }
}
```

Figura 13-Funções *write* para o *char* e a *string* respetivamente

Como já referido anteriormente para se escrever no *display* do LCD é necessário converter para o código ASCII e passar este como parâmetro na função de *writeData*, o que fará com que esta envie 8bits para o LCD com o valor do *Register Select* a 1 (ativo), que irá assim colocar o código ASCII pretendido no respetivo endereço. Para escrever uma *String* é necessário chamar a função *write* para cada *char* da *String*.

3.6 Cursor/Clear LCD

```
// Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
fun cursor(line: Int,column: Int) = writeCMD(SET_DDRAM_ADDRESS+column+(if(line!=0)line*DDRAM_ADDRESS_NEXT_L

// 1.52ms (return home) + 37 microseconds * 0x80 (total cells), worst case (for the others instructions +1m
fun clear() {writeCMD(CLEAR_DISPLAY)
Time.sleep(6)}
```

A função cursor posiciona o cursor na linha e na coluna do LCD pretendidas, sendo estas indicadas nos parâmetros. De modo a mudar o cursor de posição foi usado o comando SET_DDRAM_ADDRESS, como na Figura 7, avançando-se a posição onde se encontra o DDRAM para a posição do caracter onde se deseja ter o cursor. Para se limpar o que está escrito no LCD é só necessário enviar o comando 0x01 (CLEAR_DISPLAY) fazendo assim com que este fique sem caracteres escritos e com o cursor na posição inicial.

4 Conclusões

Concluindo o LCD é necessário referir que os valores referidos acima forma todos substituídos por constantes que os representam de modo a aumentar a legibilidade do código (podemos observar estas constantes na figura 12).

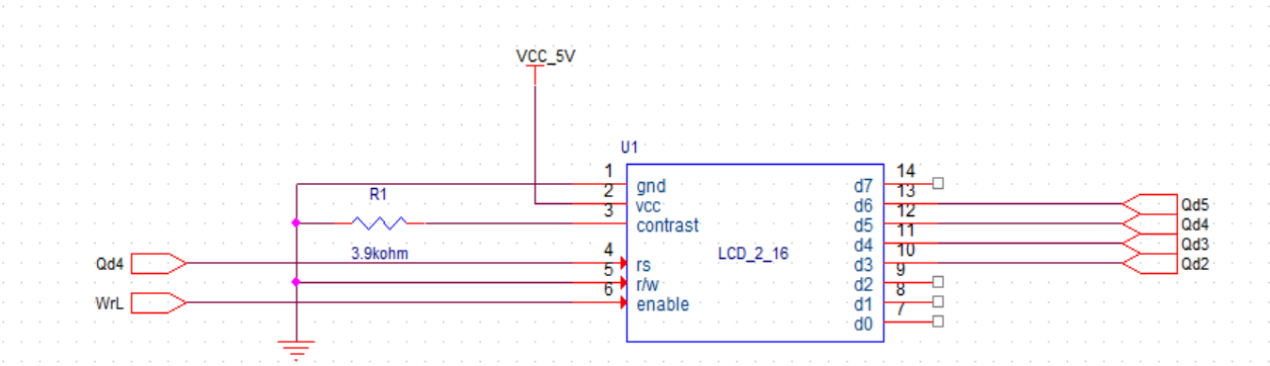
```
import isel.leic.UsbPort
import isel.leic.utils.Time

object LCD {
    // Dimensão do display.
    const val LINES = 2
    const val COLS = 16
    private const val E_MASK = 0x20 // 00100000
    private const val RS_MASK = 0x10 //00010000
    private const val DATA_BITS_MASK = 0x0F //LCD data bits
    private const val SERIAL_MODE = true //modo de envio de informacao do LCD
    private const val FUNCTION_SET_SPECIAL4_INTERFACE_8BITS = 0x3
    private const val FUNCTION_SET_SPECIAL4_INTERFACE_4BITS = 0x2
    private const val FUNCTION_SET_INTERFACE_4BITS_2LINE_DISPLAY_FONT_5X7_DOTS = 0x28
    private const val DISPLAY_OFF = 0x8
    private const val ENTRY_MODE_INCREMENT_CURSOR_NO_DISPLAY_SHIFT = 0X6
    private const val DISPLAY_ON_CURSOR_ON_BLINK_CURSOR_ON = 0X0F
    private const val SET_DDRAM_ADDRESS = 0x80
    private const val CLEAR_DISPLAY = 0x01
    private const val DDRAM_ADDRESS_NEXT_LINE = 0x40
    private const val HALF_BYTE = 4
```

Figura 12 -Constantes utilizadas no *software* do LCD.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

A. Esquema elétrico LCD



Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

B. Código do *software* do LCD

```
import isel.leic.UsbPort
import isel.leic.utils.Time

object LCD {
    // Dimensão do display.
    const val LINES = 2
    const val COLS = 16
    private const val E_MASK = 0x20 // 00100000
    private const val RS_MASK = 0x10 //00010000
    private const val DATA_BITS_MASK = 0x0F //LCD data bits
    private const val SERIAL_MODE = true //modo de envio de informacao do LCD
    private const val FUNCTION_SET_SPECIAL4_INTERFACE_8BITS = 0x3
    private const val FUNCTION_SET_SPECIAL4_INTERFACE_4BITS = 0x2
    private const val FUNCTION_SET_INTERFACE_4BITS_2LINE_DISPLAY_FONT_5X7_DOTS = 0x28
    private const val DISPLAY_OFF = 0x8
    private const val ENTRY_MODE_INCREMENT_CURSOR_NO_DISPLAY_SHIFT = 0X6
    private const val DISPLAY_ON_CURSOR_ON_BLINK_CURSOR_ON = 0X0F
    private const val SET_DDRAM_ADDRESS = 0x80
    private const val CLEAR_DISPLAY = 0x01
    private const val DDRAM_ADDRESS_NEXT_LINE = 0x40
    private const val HALF_BYTE = 4
```

C.

```
fun writeCostumCharToCGRAM(array: Array<Int>,posiValue: Int){
    writeCMD(0x40+posiValue*8)
    repeat(8){
        writeDATA(array[it])
    }
    cursor(0,0)
}
```

D.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
// Envia a sequência de iniciação para comunicação a 4bits.
fun init(){
    Time.sleep(15)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_8BITS)
    Time.sleep(5)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_8BITS)
    Time.sleep(1)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_8BITS)
    writeNibble(false,FUNCTION_SET_SPECIAL4_INTERFACE_4BITS)
    writeCMD(FUNCTION_SET_INTERFACE_4BITS_2LINE_DISPLAY_FONT_5X7_DOTS)
    writeCMD(DISPLAY_OFF)
    writeCMD(CLEAR_DISPLAY)
    writeCMD(ENTRY_MODE_INCREMENT_CURSOR_NO_DISPLAY_SHIFT)
    writeCMD(DISPLAY_ON_CURSOR_ON_BLINK_CURSOR_ON)
}
```

E.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
// Escreve um nibble de comando/dados no LCDprivate
private fun writeNibble(rs: Boolean, data: Int){
    when(SERIAL_MODE){
        true-> writeNibbleSerial(rs,data)
        else-> writeNibbleParallel(rs,data)
    }
}

private fun writeNibbleSerial(rs: Boolean, data: Int){
    val rsAndData = if(rs) data or RS_MASK else data
    SerialEmitter.send(SerialEmitter.Destination.LCD,rsAndData)
}

private fun writeNibbleParallel(rs: Boolean, data: Int){
    //e,rs,outro
    val rsBit = if(rs) RS_MASK else 0
    HAL.writeBits(RS_MASK,rsBit)
    Time.sleep(1)
    HAL.writeBits(RS_MASK+E_MASK,rsBit + E_MASK)
    Time.sleep(1)
    HAL.writeBits(RS_MASK+E_MASK+DATA_BITS_MASK,rsBit+E_MASK+data)
    Time.sleep(1)
    HAL.clrBits(E_MASK)
}

// Escreve um byte de comando/dados no LCDprivate
private fun writeByte(rs: Boolean, data: Int){
    writeNibble(rs,data shr HALF_BYTE)
    writeNibble(rs,(data and DATA_BITS_MASK))
}
```

F.

G.

H.

I.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
    // Escreve um comando no LCD
    private fun writeCMD(data: Int) = writeByte(false,data)

    // Escreve um dado no LCD
    private fun writeDATA(data: Int) = writeByte(true,data)

    // Escreve um caráter na posição corrente. Pode rebentar, .toInt() -> .code
    fun write(c: Char) = writeDATA(c.code)

    // Escreve uma string na posição corrente.
    fun write(text: String){
        for(element in text) { write(element) }
    }

    // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
    fun cursor(line: Int,column: Int) = writeCMD(SET_DDRAM_ADDRESS+column+(if(line!=0)line*DDRAM_ADDRESS_NEXT_LINE else 0))

    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() = writeCMD(CLEAR_DISPLAY)
}
```

J.