Small addition to the LCD API

```
;     Function: LCD_PrintChar
;     Register-safe!
;     Description:
;           Basically a globally exposed WriteData. Used to push individual
;           characters to the display.
;     Args:
;           R1    -     Character to be displayed
;     Returns:
;           N/A
;     Register Use:
;           R1    -     Argument
LCD_PrintChar:
      PUSH {LR}

      BL WriteData

      POP {PC}
```

```
;     Evan Heinrich
;     CE2801 sect. 011
;     10/12/2021
;
;     File:
;           main.S
;     Description of File:
;           Lab 5 driver program
;     (opt) Dependencies:
;           delay.S
;           LCD_Control.S

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text
.global main

main:
      BL LCD_Init                   ; Initialize display

      BL Key_Init                   ; Initialize keypad

      MOV R6, #0                    ; Initialize line counter
      MOV R7, #0                    ; Initialize char counter
1:
      BL Key_GetChar                ; Get the key being pressed
      MOV R1, R0                    ; Move it into an argument register
      BL LCD_PrintChar              ; Print the character
      ADD R7, R7, #1                ; Increment the char counter
      CMP R7, #16                   ; If there are 16 characters
      BEQ newLine                   ; Move to a new line
      B 1b                          ; Otherwise loop

newLine:
      CMP R6, #1                    ; Determine if we are on line 0 or 1
      ITTT NE                       ; If we are on the first row
            MOVNE R0, #1            ; Second row index
            MOVNE R1, #0            ; First column index
            BLNE LCD_MoveCursor     ; Move the cursor

      CMP R6, #1                    ; Redo comparison just to be safe
      ITT NE                        ; Again if we are on the first row
            MOVNE R6, #1            ; Update row counter
            MOVNE R7, #0            ; Reset char counter
            BNE 1b                  ; Jump back to loop

      CMP R6, #1                    ; Again, redo the comparison
      ITTT EQ                       ; If we are on the second row
            MOVEQ R6, #0            ; Update row counter
            MOVEQ R7, #0            ; Update char counter
            BLEQ LCD_Home           ; Home the cursor
      B 1b                          ; Return to loop
```

```
;     Evan Heinrich
;     CE2801 sect. 011
;     10/12/2021
;
;     File:
;           keypad.S
;     Description of File:
;           Lab 5 Keypad API
;     (opt) Dependencies:
;           delay.S
;           LCD_Control.S
;           keypad.S

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text

; Global Functions
.global Key_Init
.global Key_GetKey_NoBlock
.global Key_GetKey
.global Key_GetChar

; Constants
.equ RCC_BASE,    0x40023800  ; Base address for RCC
.equ RCC_AHB1ENR, 0x30        ; Offset from RCC to AHB1ENR
.equ RCC_GPIOCEN, 1 << 2      ; Location of the GPIOC Enabler
.equ GPIOC_BASE,  0x40020800  ; Base address for GPIOC
.equ GPIO_MODER,  0x0         ; Offset to the mode register for all GPIO ports
.equ GPIO_ODR,    0x14        ; Offset to the ODR for all GPIO ports
.equ GPIO_IDR,    0x10        ; Offset to the IDR for all GPIO ports
.equ GPIO_PUPDR,  0x0C        ; Offset to the PUPDR for all GPIO ports
.equ ROW_INPUT,   0x55        ; Mask to set rows as inputs and columns as outputs
.equ COL_INPUT,   0x55 << 8   ; Mask to set columns as inputs and rows as outputs
```

```
;     Function: Key_Init
;     Register-safe!
;     Description:
;           Initializes the GPIO port for use with the keypad
;     Args:
;           N/A
;     Returns:
;           N/A
;     Register Use:
;           R1    -     Instructions/Commands
;           R2    -     Masks
;           R3    -     Masks
; Keypad lives on PC0-PC7
; Row[0] = PC4; Row[3] = PC7
; Col[0] = PC0; Col[3] = PC3
Key_Init:
      PUSH {R1-R3, LR}              ; Backup

      LDR R1, =RCC_BASE             ; Load RCC base address
      LDR R2, [R1, #RCC_AHB1ENR]    ; Read from the RCC AHB1 enable register
      ORR R2, #RCC_GPIOCEN          ; Apply mask to enable GPIOC
      STR R2, [R1, #RCC_AHB1ENR]    ; Write back to the RCC

      LDR R1, =GPIOC_BASE           ; Load GPIOC base address
      LDR R2, [R1, #GPIO_MODER]     ; Read from the current mode register
      MOV R3, #ROW_INPUT            ; Load mask to set rows as input
      BFI R2, R3, #0, #16           ; Insert mask where PC0-PC7 live
      STR R2, [R1, #GPIO_MODER]     ; Write back to the mode register

      ; R1 still contains GPIOC's base address, so now configure PUPDR

      LDR R2, [R1, #GPIO_PUPDR]     ; Read the current pull-up/down register
      LDR R3, =0xAAAA               ; Load the mask to set our pins to pull-up
      ORR R2, R3                    ; Apply mask
      STR R2, [R1, #GPIO_PUPDR]     ; Write back to pull-up/down register

      POP {R1-R3, PC}               ; Restore & Return
```

```
;     Function: Key_GetKey_NoBlock
;     Register-safe!
;     Description:
;           Returns a numerical value 0-16 whenever called based on what key
;           is being pressed. A return value of 0 means no keys are pressed.
;           Also returns zero if multiple keys are pressed.
;     Args:
;           N/A
;     Returns:
;           R0    -     Numerical representation of the key being pressed
;     Register Use:
;           R0    -     Return
;           R1    -     Addresses
;           R2    -     Masks
;           R3    -     Column index
;           R4    -     Row index
Key_GetKey_NoBlock:
      ; Comments regarding how the keypad was implemented are at
      ; the end of the file.
      PUSH {R1-R4, LR}  ; backup registers

      ; Clear used registers because some BFI's are used
      MOV R0, #0
      MOV R3, #0
      MOV R4, #0

      ; Configure rows as inputs, columns as outputs
      LDR R1, =GPIOC_BASE           ; Load GPIOC base address
      LDR R2, [R1, #GPIO_MODER]     ; Read from the current mode register
      MOV R3, #ROW_INPUT            ; Load mask to set rows as input
      BFI R2, R3, #0, #16           ; Insert mask where PC0-PC7 live
      STR R2, [R1, #GPIO_MODER]     ; Write back to the mode register

      ; Push '1111' onto columns
      LDR R2, [R1, #GPIO_ODR]       ; Read current ODR
      ORR R2, #0xF                  ; Push 1111
      STR R2, [R1, #GPIO_ODR]       ; Write

      ; Give the electricity time to propagate
      MOV R1, #5
      BL delay_us

      ; Read in rows IDR
      LDR R1, =GPIOC_BASE     ; Load GPIOC base address
      LDR R2, [R1, #GPIO_IDR] ; Read current IDR
      LSR R2, R2, #4          ; Rows are in the upper nibble, so shift right 4 times
      BFI R4, R2, #0, #4      ; Store value into R4

      ; Swap rows to outputs and columns as inputs
      LDR R1, =GPIOC_BASE           ; Load GPIOC base address
      LDR R2, [R1, #GPIO_MODER]     ; Read from the current mode register
      MOV R3, #COL_INPUT            ; Load mask to set rows as input
      BFI R2, R3, #0, #16           ; Insert mask where PC0-PC7 live
      STR R2, [R1, #GPIO_MODER]     ; Write back to the mode register
```

```
; Push the stored value that was on rows IDR to the ODR
LDR R1, =GPIOC_BASE           ; Load GPIOC base address
LDR R2, [R1, #GPIO_ODR]       ; Read from the current ODR
BFI R2, R4, #4, #4            ; Insert into the upper nibble, aka rows
STR R2, [R1, #GPIO_ODR]       ; Write back to the ODR


; Give the electricity time to propagate
MOV R1, #5
BL delay_us

; Clear R3 because it still has a mask
MOV R3, #0

; Read the column IDR
LDR R1, =GPIOC_BASE           ; Load GPIOC base address
LDR R2, [R1, #GPIO_IDR]       ; Read the current IDR
BFI R3, R2, #0, #4            ; Store the upper nibble

MOV R1, R3                    ; Move to argument register
MOV R2, R4                    ; Move to argument register
BL IndexToNum                 ; Convert the two indexes to a numerical value

POP {R1-R4, PC}
```

```
;     Function: Key_GetKey
;     Register-safe!
;     Description:
;           A blocking implementation of GetKey_NoBlock. Waits for a key
;           to be pressed and released, then returns the key that was pressed.
;     Args:
;           N/A
;     Returns:
;           Numerical value representing what key was pressed
;     Register Use:
;           R0    -     Return value
;           R1    -     Subroutine arguments
;           R2    -     Backup copy of the button code
Key_GetKey:
      PUSH {R1-R2, LR}
      1:
            ; Delay 10ms for debouncing
            MOV R1, #10
            BL delay_ms

            ; Check if there's a key being pressed
            BL Key_GetKey_NoBlock

            ; Compare to 0 as it means no buttons being pressed
            ; If there isn't a button being pressed, loop.
            CMP R0, #0
            BEQ 1b
            MOV R2, R0
      1:
            ; Delay 10ms for debouncing
            MOV R1, #10
            BL delay_ms

            ; Get the key being pressed
            BL Key_GetKey_NoBlock

            ; Compare to the code representing no buttons pressed
            ; and if a button is being pressed, loop until it isn’t
            CMP R0, #0
            BNE 1b

            ; Load backup value of the key that was pressed
            MOV R0, R2

      ; Return
      POP {R1-R2, PC}
```

```
;     Function: Key_GetChar
;     Register-safe!
;     Description:
;           Calls GetKey and interprets the returned key code
;           as an ASCII character.
;     ->    ASCII characters are stored in RODATA as an array
;     ->    Numerical keycode can be thought of as the array index
;     Args:
;           N/A
;     Returns:
;           ASCII character byte representing the pressed button
;     Register Use:
;           R0    -     Return value
;           R1    -     Subroutine arguments
;           R2    -     Array address
Key_GetChar:
      PUSH {R1-R2, LR}

      BL Key_GetKey

      MOV R1, R0

      LDR R2, =chars
      LDRB R0, [R2, R1]

      POP {R1-R2, PC}
```

```
;     Function: IndexToNum
;     Register-safe!
;     Description:
;     ->    Helper method
;           Decodes the indexes provided from the GetKey functions and
;           returns a numerical representation of the key being pressed.
;     ->    Basically just a case statement.
;     Args:
;           R1    -     Column index
;           R2    -     Row index
;     Returns:
;           R0    -     Numerical representation of the key at col,row
;     Register Use:
;           R0    -     Return
;           R1    -     Argument
;           R2    -     Argument
IndexToNum:
      PUSH {LR}

      CMP R1, #0b0001         ; First column case
      BEQ column1

      CMP R1, #0b0010         ; Second column case
      BEQ column2

      CMP R1, #0b0100         ; Third column case
      BEQ column3

      CMP R1, #0b1000         ; Fourth column case
      BEQ column4

      ; Default case; only 16 buttons on our keypad.
      MOV R0, #0
      B return

      column1:
            CMP R2, #0b0001         ; First row case
            IT EQ
                  MOVEQ R0, #1            ; Column 1, Row 1
                  BEQ return
            CMP R2, #0b0010         ; Second row case
            IT EQ
                  MOVEQ R0, #4            ; Column 1, Row 2
                  BEQ return
            CMP R2, #0b0100         ; Third row case
            IT EQ
                  MOVEQ R0, #7            ; Column 1, Row 3
                  BEQ return
            CMP R2, #0b1000         ; Fourth row case
            IT EQ
                  MOVEQ R0, #0xF          ; Column 1, Row 4
                  BEQ return
            ; Default case; only 16 buttons on our keypad.
            MOV R0, #0
            B return

      column2:
            CMP R2, #0b0001         ; First row case
            IT EQ
                  MOVEQ R0, #2            ; Column 2, Row 1
                  BEQ return
            CMP R2, #0b0010         ; Second row case
            IT EQ
                  MOVEQ R0, #5            ; Column 2, Row 2
                  BEQ return
            CMP R2, #0b0100         ; Third row case
            IT EQ
                  MOVEQ R0, #8            ; Column 2, Row 3
                  BEQ return
            CMP R2, #0b1000         ; Fourth row case
            IT EQ
                  MOVEQ R0, #16           ; Column 2, Row 4
                  BEQ return

            ; Default case; only 16 buttons on our keypad.
            MOV R0, #0
            B return

      column3:
            CMP R2, #0b0001         ; First row case
            IT EQ
                  MOVEQ R0, #3            ; Column 3, Row 1
                  BEQ return
            CMP R2, #0b0010         ; Second row case
            IT EQ
                  MOVEQ R0, #6            ; Column 3, Row 2
                  BEQ return
            CMP R2, #0b0100         ; Third row case
            IT EQ
                  MOVEQ R0, #9            ; Column 3, Row 3
                  BEQ return
            CMP R2, #0b1000         ; Fourth row case
            IT EQ
                  MOVEQ R0, #0xE          ; Column 3, Row 4
                  BEQ return

            ; Default case; only 16 buttons on our keypad.
            MOV R0, #0
            B return
            column4:
                  CMP R2, #0b0001         ; First row case
                  IT EQ
                        MOVEQ R0, #0xA          ; Column 4, Row 1
                        BEQ return
                  CMP R2, #0b0010         ; Second row case
                  IT EQ
                        MOVEQ R0, #0xB          ; Column 4, Row 2
                        BEQ return
                  CMP R2, #0b0100         ; Third row case
                  IT EQ
                        MOVEQ R0, #0xC          ; Column 4, Row 3
                        BEQ return
                  CMP R2, #0b1000         ; Fourth row case
                  IT EQ
                        MOVEQ R0, #0xD          ; Column 4, Row 4
                        BEQ return

                  ; Default case; only 16 buttons on our keypad.
                  MOV R0, #0
                  B return

            return:
                  POP {PC}

      .section .rodata
      chars:      .ascii      "0123456789ABCD#*0"


      ; Implement using Keypad scanning
      ; Rows are stored in upper nibble (PC4-PC7)
      ; Cols are stored in lower nibble (PC0-PC3)
      ; 1.  Columns -> Outputs
      ;           Rows  -> Inputs
      ; 2.  '0000' -> Rows
      ; 3.  Wait small us delay
      ; 4.  Read rows IDR, example '1110' (Row 0 has a switch active)
      ; 5.  Backup row IDR
      ; 6.  Swap Columns to inputs and rows to outputs
      ; 7.  Store the backup of row IDR back on the row ODR
      ; 8.  Read column IDR, example '1101' (Row 0 was active, Column 1 is active)
      ; 9.  Insert row backup into top nibble, column into lower
```