*; Evan Heinrich*  
*; CE2801 sect. 011*  
*; 10/5/2021*  
*;*  
*; File:*  
*; main.S*  
*; Description of File:*  
*; Lab 4 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*  
  
 **LDR** R1, *=msg04* *; Load large number test text*  
 **BL** LCD\_PrintString *; Print string*  
  
 **MOV** R0, #1 *; Second row index*  
 **MOV** R1, #0 *; First column index*  
 **BL** LCD\_MoveCursor *; Move the cursor*  
  
 **LDR** R1, *=msg05* *; Load large number test text*  
 **BL** LCD\_PrintString *; Print string*  
  
 **LDR** R1, =0xBB8 *; Prep 3 second delay*  
 **BL** delay\_ms *; Execute delay*  
  
 **BL** LCD\_Clear *; Clear display*  
  
 **MOV** R1, #10000 *; Number larger than 4 digits*  
 **BL** LCD\_PrintNum *; Attempt to print, should display "Err."*  
  
 **LDR** R1, =0xBB8 *; Prep 3 second delay*  
 **BL** delay\_ms *; Execute Delay*  
  
 **BL** LCD\_Clear *; Clear display*  
  
 **LDR** R1, *=msg01* *; Load address for the countdown message*  
 **BL** LCD\_PrintString *; Print the string*

**MOV** R0, #1 *; Second row index*  
 **MOV** R1, #0 *; First column index*  
 **BL** LCD\_MoveCursor *; Move the cursor*  
  
 **LDR** R1, *=msg02* *; Load address for the second string*  
 **BL** LCD\_PrintString *; Print the second string*  
  
 **LDR** R1, =0xBB8 *; Prep 3 second delay*  
 **BL** delay\_ms *; Execute 3 second delay*  
  
 **BL** LCD\_Clear *; Clear display*  
  
 **MOV** R7, #100 *; Initial countdown value*  
  
**1:** **MOV** R0, #1 *; Second row index*  
 **MOV** R1, #0 *; First column index*  
 **BL** LCD\_MoveCursor *; Move the cursor*  
  
 **MOV** R1, R7 *; Load the countdown number*  
 **BL** LCD\_PrintNum *; Display the countdown number*  
  
 **BL** LCD\_Home *; Home the cursor*  
  
 **LDR** R1, =0x3E8 *; Prep 1 second delay*  
 **BL** delay\_ms *; Execute 1 second delay*  
  
 **SUBS** R7, R7, #1 *; Decrement countdown value*  
  
 **IT** MI *; If the next count is negative*  
 **BMI** done *; Print done*  
 **B** 1b *; Otherwise continue looping*  
***done:***  
 **BL** LCD\_Home *; Home the display*  
  
 **LDR** R1, *=msg03* *; Load address for "Done"*  
 **BL** LCD\_PrintString *; Print "Done"*  
  
 **B** end *; Infinite loop*  
***end:*** **B** end  
  
*; RAM starts at address 0x20000000*  
***.section* .rodata**  
*msg01*: .asciz **"It's the final"**  
*msg02*: .asciz **"countdown!"**  
*msg03*: .asciz **"Done"**  
*msg04*: .asciz **"Testing large"**  
*msg05*: .asciz **"number..."**

*; Evan Heinrich*  
*; CE2801 sect. 011*  
*; 10/5/2021*  
*;*  
*; File:*  
*; LCD\_Control.S*  
*; Description of File:*  
*; Lab 4 template provided by Dr. Livingston*  
*; (opt) Dependencies:*  
*; delay.S*  
*; ASCII.S*  
  
*.syntax* unified  
*.cpu* cortex-m4  
*.thumb*  
***.section* .text**

*; Constants*  
 .equ RCC\_BASE, 0x40023800 *; Base address for RCC*  
    .equ RCC\_AHB1ENR, 0x30 *; Offset from RCC to AHB1ENR*  
    .equ RCC\_GPIOAEN, 1 << 0 *; Location of the GPIOA Enabler*  
    .equ RCC\_GPIOCEN, 1 << 2 *; Location of the GPIOC Enabler*  
  
    .equ GPIOA\_BASE, 0x40020000 *; Base address for GPIOA*  
    .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\_BSRR, 0x18 *; Offset to the BSRR for all GPIO ports*

.equ RS, 1 << 8 *; RS location*  
 .equ RW, 1 << 9 *; RW location*  
 .equ E, 1 << 10 *; E location*  
  
*; Globally exposed functions*  
***.global*** LCD\_Init  
***.global*** LCD\_Clear  
***.global*** LCD\_Home  
***.global*** LCD\_MoveCursor  
***.global*** LCD\_PrintString  
***.global*** LCD\_PrintNum

*; Function: PortSetup*  
*; Register-safe!*   
*; Description:*  
*; Helper method to configure GPIO ports A & C for use with the LCD on our*

*; devboards*  
*; Args:*  
*; N/A*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Addresses*  
*; R2 - Scratch*  
*; R3 - Masks*  
***PortSetup*:**  
 *; Backup Registers*  
    **PUSH** {R1-R3, LR}  
  
 *; Enable GPIO Ports A & C*  
    **LDR** R1, *=RCC\_BASE* *; Load RCC base address*  
    **LDR** R2, [R1, *#RCC\_AHB1ENR*] *; Read from the AHB1 Enable Register*  
    **ORR** R2, R2, *#RCC\_GPIOAEN* *; Apply GPIOA Enable mask*  
    **ORR** R2, R2, *#RCC\_GPIOCEN* *; Apply GPIOC Enable mask*  
    **STR** R2, [R1, *#RCC\_AHB1ENR*] *; Write back to memory*  
  
    *; Set GPIOA Pins as output (PA4-PA11)*  
    **LDR** R1, *=GPIOA\_BASE* *; Load GPIOA base address*  
    **LDR** R3, =0x00555500 *; Load mode mask*  
    **LDR** R2, [R1, *#GPIO\_MODER*] *; Read*  
    **ORR** R2, R3 *; Apply mode mask*  
    **STR** R2, [R1, *#GPIO\_MODER*] *; Write*  
  
    *; Set GPIOC Pins as output (PC8-PC10)*  
    **LDR** R1, *=GPIOC\_BASE* *; Load GPIOC base address*  
    **LDR** R3, =0x00550000 *; Load mode mask*  
    **LDR** R2, [R1, *#GPIO\_MODER*] *; Read*  
    **ORR** R2, R3 *; Apply mode mask*  
    **STR** R2, [R1, *#GPIO\_MODER*] *; Write*  
  
 **POP** {R1-R3, LR} *; Restore*  
    **BX** LR *; Return*

*; Function: WriteInstruction*  
*; Register-safe!*   
*; Description:*  
*; Takes an instruction to send to the LCD stored in R1 and pushes it onto the*

*; data bus*  
*; (Helper method)*  
*; Args:*  
*; R1 - Instruction to be sent*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Instruction*  
*; R2 - Scratch*  
*; R3 - GPIOC Address*  
*; R4 - GPIOA Address*  
*; R7 - Masks*  
***WriteInstruction*:**  
 **PUSH** {R1-R4, R7, LR} *; Backup registers*

**LDR** R3, *=GPIOC\_BASE* *; Load GPIO port C address*  
 **LDR** R4, *=GPIOA\_BASE* *; Load GPIO port A address*  
  
 *; Clear RS, RW, E*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **BIC** R2, *#RS* *; Apply RS set mask*  
 **BIC** R2, *#RW* *; Apply RW set mask*  
 **BIC** R2, *#E* *; Apply E clear mask*  
 **STR** R2, [R3, *#GPIO\_ODR*] *; Write*  
  
 *; Set E, E => 1*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **ORR** R2, *#E* *; Apply E set mask*  
 **STR** R2, [R3, *#GPIO\_ODR*] *; Write*  
  
 *; Push the instruction onto the data bus*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **BFI** R2, R1, #4, #8 *; Insert instruction*  
 **STR** R2, [R4, *#GPIO\_ODR*] *; Write to BSRR*  
   
 *; Clear E, E => 0*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **BIC** R2, *#E* *; Apply E clear mask*  
 **STR** R2, [R3, *#GPIO\_ODR*] *; Write*  
  
 *; Wait for appropriate delay*  
 *; -> Listed delay for holding instructions on the bus after E falls*  
 *; is 10ns, when the next instruction takes more than 60ns*  
  
 **POP** {R1-R4, R7, PC} *; Restore & Return*

*; Function: WriteData*  
*; Register-safe!*   
*; Description:*  
*; Takes data provided in R1 and pushes it to the LCD*  
*; Args:*  
*; R1 - Data to be sent*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Instruction*  
*; R2 - Scratch*  
*; R3 - GPIOC Address*  
*; R4 - GPIOA Address*  
*; R7 - Masks*  
***WriteData*:**  
 **PUSH** {R1-R4, R7, LR} *; Backup*

**LDR** R3, *=GPIOC\_BASE* *; Load GPIOC address*  
 **LDR** R4, *=GPIOA\_BASE* *; Load GPIOA address*  
  
 *; Set RS=1,RW=0,E=0*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **BIC** R2, *#E* *; Apply E clear mask*  
 **ORR** R2, *#RS* *; Apply RS set mask*  
 **BIC** R2, *#RW* *; Apply RW clear mask*  
 **STR** R2, [R3, *#GPIO\_ODR*] *; Write*  
  
 *; Set E=1*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **ORR** R2, *#E* *; Apply E set mask*  
 **STR** R2, [R3, *#GPIO\_ODR*] *; Write to BSRR*  
  
 *; Set R1 -> DataBus (PA4-PA11)*  
 **LDR** R2, [R3, *#GPIO\_ODR*] *; Read*  
 **BFI** R2, R1, #4, #8 *; Insert data onto bus*  
 **STR** R2, [R4, *#GPIO\_ODR*] *; Write*  
  
 *; Set E=0*  
 **MOV** R2, #0 *; Clear scratch register*  
 **BIC** R2, *#E* *; Apply E clear mask*  
 **STR** R2, [R3, *#GPIO\_ODR*] *; Write to BSRR*  
  
 *; >37us delay*  
 **MOV** R1, #40  
 **BL** delay\_us

**POP** {R1-R4, R7, PC}

*; Function: LCD\_Init*  
*; Register-safe!*   
*; Description:*  
*; Initializes the LCD screen on our dev boards by writing the appropriate*  
*; sequence of instructions with the appropriate delay between instructions*  
*; Args:*  
*; N/A*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Instructions/Commands*  
***LCD\_Init*:**  
 **PUSH** {R1, LR} *; Backup registers*  
  
    **BL** PortSetup *; Configure GPIO ports*  
  
    *; Write Function Set (0x38)*  
    **MOV** R1, #0x38 *; Load instruction*  
    **BL** WriteInstruction *; Write instruction*  
  
    **MOV** R1, #40 *; >37us delay after prev. command*  
    **BL** delay\_us *; Execute delay*  
  
    *; Write Function Set (0x38)*  
    **MOV** R1, #0x38 *; Load instruction*  
    **BL** WriteInstruction *; Write instruction*  
  
    **MOV** R1, #40 *; >37us delay after prev. command*  
    **BL** delay\_us *; Execute delay*  
  
    *; Write Display On/Off(0x0F)*  
    **MOV** R1, #0x0F *; Load instruction*  
    **BL** WriteInstruction *; Write instruction*  
  
    **MOV** R1, #40 *; >37us delay after prev. command*  
    **BL** delay\_us *; Execute delay*  
  
    *; Write Display Clear (0x01)*  
    **MOV** R1, 0x01 *; Load instruction*  
    **BL** WriteInstruction *; Execute instruction*  
  
    **MOV** R1, #2 *; >1.52ms delay after prev. command*  
    **BL** delay\_ms *; Execute delay*  
  
    *#Write* Entry Mode Set (0x06)  
    **MOV** R1, #0x06 *; Load instruction*  
    **BL** WriteInstruction *; Execute instruction*  
  
 **MOV** R1, #40 *; >37us delay after prev. command*  
 **BL** delay\_us *; Execute delay*  
  
 **POP** {R1, PC} *; Restore & Return*  
  
  
  
*; Function: LCD\_Clear*  
*; Register-safe!*   
*; Description:*  
*; Clears the contents of the display and waits the appropriate >1.52ms delay*  
*; -> Clear display is instruction 0x01*  
*; Args:*  
*; N/A*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Instruction & Delay*  
***LCD\_Clear*:**  
 **PUSH** {R1, LR} *; Backup registers*  
  
 **MOV** R1, #0x01 *; Load instruction*  
 **BL** WriteInstruction *; Execute instruction*  
  
 **MOV** R1, #2 *; Load delay*  
 **BL** delay\_ms *; Execute delay*  
  
 **POP** {R1, PC} *; Restore & return*  
  
  
  
*; Function: LCD\_Home*  
*; Register-safe!*   
*; Description:*  
*; Returns the cursor of the LCD to its home position (top left) and waits the*  
*; appropriate >1.52ms delay*  
*; -> Return home is instruction 0x02*  
*; Args:*  
*; N/A*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Instructions & Delay*  
***LCD\_Home*:**  
 **PUSH** {R1, LR} *; Backup registers*  
  
 **MOV** R1, #0x02 *; Load instruction*  
 **BL** WriteInstruction *; Execute instruction*  
  
 **MOV** R1, #2 *; Load delay*  
 **BL** delay\_ms *; Execute delay*  
  
 **POP** {R1, PC} *; Restore & return*

*; Function: LCD\_MoveCursor*  
*; Register-safe! Pushes all general purpose registers (R0-R12 & LR) to the stack*  
*; Description:*  
*; Moves the cursor to a specified position on the LCD*  
*; Rows & Columns are ZERO INDEXED*  
*; Args:*  
*; R0 - Zero-indexed row, [0-1] for us*  
*; R1 - Zero-indexed column, [0-15] for the active display*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R0 - Argument*  
*; R1 - Argument*  
*; R7 - Scratch*  
*; R6 - Command mask*  
***LCD\_MoveCursor*:**  
 **PUSH** {R0-R1, R6-R7, LR}  
 **MOV** R7, #0 *; Clear scratch register*  
 **MOV** R6, #0 *; Command register*  
  
 **CMP** R0, #0 *; Determine if in top row*  
 **IT** NE  
 **MOVNE** R7, #0x40 *; Load second row mask if in bottom row*  
  
 **ORR** R7, R7, R1 *; Apply mask*  
 *; This gives us the desired address*  
  
 **MOV** R6, #1 << 8 *; Load command mask, 0b10000000*  
 **ORR** R1, R6, R7 *; Apply mask to desired address*  
 *; This should make the command be 0b1aaaaaaa where*  
 *; all of the a's represent the address of the desired*  
 *; location. Result is stored in R1, so we just call*  
 *; the method that pushes instructions*  
  
 **BL** WriteInstruction *; Push instruction to the LCD*  
  
 **MOV** R1, #40 *; >37us delay for moving cursor*  
 **BL** delay\_us *; Execute delay*  
  
 **POP** {R0-R1, R6-R7, PC}

*; Function: LCD\_PrintString*  
*; Register-safe! Pushes all general purpose registers (R0-R12 & LR) to the stack*  
*; Description:*  
*; Prints a string to the LCD & returns the number of characters written*  
*; -> String must be null-terminated*  
*; -> Memory address to string is provided in R1*  
*; Args:*  
*; R1 - Address to null-terminated string*  
*; Returns:*  
*; R0 - Number of characters printed*  
*; Register Use:*  
*; R0 - Return*  
*; R1 - Argument*  
*; R2 - Character currently being displayed*  
***LCD\_PrintString*:**  
 **PUSH** {R1-R2, LR} *; We don't need to back up R0 because it is a return*  
 **MOV** R0, #0 *; Iterator value*  
  
 *; Determine the length of the string*  
***loop*:**  
 **LDRB** R2, [R1, R0] *; Load character from the string with offset R0*  
 **CMP** R2, #0 *; Determine if the character is null*  
 **ITTTT** NE *; If the character isn't null*  
 **ADDNE** R0, #1 *; Increment the iterator*  
 **PUSHNE** {R1} *; Backup the address*  
 **MOVNE** R1, R2 *; Move the character into R1*  
 **BLNE** WriteData *; Write the character*  
  
 *; Because I built the delay for writing characters into WriteData,*  
 *; the condition flags get updated making the next IT block inaccurate*  
 *; so I need to redo the original comparisons to fix the PSR*  
  
 **CMP** R2, #0  
 **ITT** NE  
 **POPNE** {R1} *; Restore address*  
 **BNE** loop *; Loop until we hit a null char*  
  
 **POP** {R1-R2, PC} *; Restore & return*

*; Function: LCD\_PrintNum*  
*; Register-safe! Pushes all general purpose registers (R0-R12 & LR) to the stack*  
*; Description:*  
*; Prints a decimal number [0-9999] to the LCD display*  
*; -> If the number is greater than 4 digits, "Err." prints to the display*  
*; Args:*  
*; R1 - Decimal number to be printed*  
*; Returns:*  
*; N/A*  
*; Register Use:*  
*; R1 - Argument*  
*; R2 - Masks*  
***LCD\_PrintNum*:**  
 **PUSH** {R1-R2, LR}  
  
 **BL** num\_to\_ASCII *; Stores ASCII representing chars in R0*  
 **MOV** R1, #0 *; Clear R1 so we can use it for WriteData*  
 **MOV** R2, #0xFF000000 *; Base mask for characters*  
  
 **AND** R1, R0, R2 *; Mask off all but first char*  
 **LSR** R1, R1, #24 *; Move char into correct position*  
 **BL** WriteData *; Write char*  
  
 **LSR** R2, R2, #8 *; Shift mask right by one char*  
 **AND** R1, R0, R2 *; Apply mask*  
 **LSR** R1, R1, #16 *; Move char into correct position*  
 **BL** WriteData *; Write char*  
  
 **LSR** R2, R2, #8 *; Shift mask right by one char*  
 **AND** R1, R0, R2 *; Apply mask*  
 **LSR** R1, R1, #8 *; Move char into correct position*  
 **BL** WriteData *; Write char*  
  
 **LSR** R2, R2, #8 *; Shift mask left by one char*  
 **AND** R1, R0, R2 *; Apply mask*  
 **BL** WriteData *; Write char*  
  
 **POP** {R1-R2, PC}