// Evan Heinrich

// CE2801 sect. 011

// 10/26/2021

//

// File:

// main.S

// Description of File:

// Main program of Lab 7

// Hexadecimal conversion game

// (opt) Dependencies:

// timer\_delay.S

// LCD\_Control.S

// keypad.S

// ASCII.S

// LED.S

// tone.S

// Assembler Directives

.syntax unified

.cpu cortex-m4

.thumb

**.section** **.text**

.equ MSG\_DELAY, 3 // Time in seconds to display each message

.equ RESET\_TIME, 2 // Delay between showing time and restarting

.equ MAX\_LOOPS, 4 // Total number of conversion loops

.equ TIM5\_BASE, 0x40000C00 // Timer 5 base address

.equ RCC\_BASE, 0x40023800 // RCC base address

.equ APB1ENR, 0x40 // Offset from RCC base to APB1ENR

.equ TIM5EN, 1 << 3 // Location in APB1ENR of TIM5EN

.equ TIM\_CR1, 0x00 // Offset from TIMx base to control reg. 1

.equ TIM\_PSC, 0x28 // Offset from TIMx base to prescale register

.equ TIM\_CNT, 0x24 // Offset from TIMx base to count register

.equ TIM\_EGR, 0x14

.equ TIM\_SR, 0x10

.equ TIM\_CEN, 1 << 0 // Location of CEN in CR1

.equ TIM\_UG, 1 << 0

.equ TIM\_UIF, 1 << 0

**.global** main

**main:**

// First time setup

BL delay\_Init

BL LCD\_Init

BL key\_Init

BL rnd\_Init

BL LED\_Init

BL tone\_Init

// Configure conversion timer

LDR R1, =RCC\_BASE // RCC base address

LDR R2, [R1, #APB1ENR] // Read

ORR R2, #TIM5EN // Enable TIM5

STR R2, [R1, #APB1ENR] // Write

LDR R1, =TIM5\_BASE // Timer 5 base address

LDR R2, =16000-1 // 16MHz / 16kHz = 1kHz aka 1ms

STR R2, [R1, #TIM\_PSC] // Apply prescale to 1ms per count

// This is the prescaler fix Dr. Livingston provided

// From what I understand, it forces an update event

// on the timer, which somehow forces the prescale into effect.

MOV R2, #TIM\_UG

LDR R1, =TIM5\_BASE

STR R2, [r1, #TIM\_EGR]

LDR R1, =TIM5\_BASE

LDR R2, [R1, #TIM\_SR]

BIC R2, #TIM\_UIF

STR R2, [R1, #TIM\_SR]

// DO NOT start timer counting yet

BL FirstText

// Check the memory address that reports if

// the keypad interrupt was triggered

**loop:**

LDR R1, =press

LDRB R2, [R1]

CMP R2, #0

IT NE

BLNE onPress

B loop

// Code to be executed when a button is pressed on the keypad

// R0 = returns/args

// R1 = Addr/Arg

// R2 = Scratch

// R3 = Conversion time

// R4 = Random number

**onPress:**

PUSH {R0, R1, R2, LR}

// Stop conversion timer

BL StopTimer

// Get keycode

LDR R1, =button

LDRB R2, [R1]

// Convert keycode to char

MOV R1, R2

BL key\_ToHexChar

// Print char to screen

// Even with debouncing, occasionally a

// null char makes it through to here

// so this just filters out nulls

MOV R1, R0

CMP R1, #0

IT NE

BLNE LCD\_PrintChar

// Convert to a number with nulls filtered out

CMP R1, #0

IT NE

BLNE key\_ToNum

// Compare entered number to RNG number

LDR R1, =random // random number address

LDR R4, [R1] // load random number

CMP R0, R4 // compare key to random number

// Display "correct" if correct

IT EQ

BLEQ DisplayCorrect

// Redo Comparison

LDR R1, =random // random number address

LDR R4, [R1] // load random number

CMP R0, R4 // compare key to random number

// Display "incorrect" if incorrect

IT NE

BLNE DisplayIncorrect

MOV R1, #MSG\_DELAY

BL delay\_sec

BL LCD\_Clear

// Display time it took to convert

BL DisplayTime

// Reset timer

BL ResetTimer

// Reset interrupt status

LDR R1, =press

MOV R2, #0

STRB R2, [R1]

MOV R1, #RESET\_TIME

BL delay\_sec

// Load loop iterator

LDR R1, =loops

LDRB R2, [R1]

// Compare to maximum loops

MOV R3, #MAX\_LOOPS

CMP R2, R3

// Increment iterator if less than max loops

// and store new value.

// Loop program if less than, show smallest time otherwise

ITTE LT

ADDLT R2, #1

STRBLT R2, [R1]

BGE DisplayMinTime // Go to end of progam

// Soft Restart

BL SoftRestart

POP {R0, R1, R2, PC}

**FirstText:**

PUSH {R1, LR}

// Startup tone

BL tone\_Success

// My hex conversion game

LDR R1, =msg01

BL LCD\_PrintString

// Delay between messages

MOV R1, #MSG\_DELAY

BL delay\_sec

// Clear display and print first info page

BL LCD\_Clear

LDR R1, =msg02

BL LCD\_PrintString

// Delay between messages

MOV R1, #MSG\_DELAY

BL delay\_sec

// Clear display and print second info page

BL LCD\_Clear

LDR R1, =msg03

BL LCD\_PrintString

// Delay between messages

MOV R1, #MSG\_DELAY

BL delay\_sec

// “# = 0xE, \* = 0xF”

BL LCD\_Clear

LDR R1, =msg04

BL LCD\_PrintString

// Delay between messages

MOV R1, #MSG\_DELAY

BL delay\_sec

// Clear and print ready set

BL LCD\_Clear

LDR R1, =ReadySet

BL LCD\_PrintString

// Slightly shorter delay between messages

MOV R1, #MSG\_DELAY-1

BL delay\_sec

// Print “go”

LDR R1, =go

BL LCD\_PrintString

// Newline

MOV R0, #1

MOV R1, #0

BL LCD\_MoveCursor

// Print “0x”

LDR R1, =hex

BL LCD\_PrintString

// Generate random number

BL rnd\_Sample4

// Store random number

LDR R1, =random

STR R0, [R1]

// Display random number

MOV R1, R0

BL num\_to\_LED

// Start conversion timer

BL StartTimer

POP {R1, PC}

**StartTimer:**

PUSH {R1, R2, LR}

LDR R1, =TIM5\_BASE

LDR R2, [R1, #TIM\_CR1]

ORR R2, #TIM\_CEN

STR R2, [R1, #TIM\_CR1]

POP {R1, R2, PC}

// Stores last time in memory

**StopTimer:**

PUSH {R0-R2, LR}

// Stop the timer

LDR R1, =TIM5\_BASE

LDR R2, [R1, #TIM\_CR1]

BIC R2, #TIM\_CEN

STR R2, [R1, #TIM\_CR1]

// Load R0 with the count

LDR R0, [R1, #TIM\_CNT]

// Store the last count to memory

LDR R1, =time

STR R0, [R1]

POP {R0-R2, PC}

**ResetTimer:**

PUSH {R1, R2, LR}

LDR R1, =TIM5\_BASE

MOV R2, #0

STR R2, [R1, #TIM\_CNT]

POP {R1, R2, PC}

// Basically just does “ready set go” and starts another conversion

**SoftRestart:**

PUSH {R0, R1, LR}

// Clear display and print ready set

BL LCD\_Clear

LDR R1, =ReadySet

BL LCD\_PrintString

// Slightly shorter delay between messages

MOV R1, #MSG\_DELAY-1

BL delay\_sec

// Print go

LDR R1, =go

BL LCD\_PrintString

// Newline

MOV R0, #1

MOV R1, #0

BL LCD\_MoveCursor

// Print “0x”

LDR R1, =hex

BL LCD\_PrintString

// Generate random number

BL rnd\_Sample4

// Store random number

LDR R1, =random

STR R0, [R1]

// Display random number

MOV R1, R0

BL num\_to\_LED

// Start conversion timer

BL StartTimer

POP {R0, R1, PC}

**DisplayCorrect:**

PUSH {R0-R2, LR}

// Load and display correct text

LDR R1, =correct

BL LCD\_PrintString

// Play success tone

BL tone\_Success

// Store the last count to memory

LDR R1, =time

LDR R0, [R1]

// Load previous minimum time

LDR R1, =minTime

LDR R2, [R1]

// Compare to this time

CMP R0, R2

// Store new time if it is less

// than the previous minimum time

IT LT

STRLT R0, [R1]

// Increment correct counter

LDR R1, =correctCNT

LDRB R2, [R1]

ADD R2, #1

STRB R2, [R1]

POP {R0-R2, PC}

**DisplayIncorrect:**

PUSH {R1, LR}

// Load and display incorrect text

LDR R1, =incorrect

BL LCD\_PrintString

// Play failure tone

BL tone\_Failure

POP {R1, PC}

**DisplayTime:**

PUSH {R0, R1, LR}

// Display time message

LDR R1, =timeMSG

BL LCD\_PrintString

// Print time in milliseconds

LDR R2, =time

LDR R1, [R2]

BL LCD\_PrintNum

// Append "ms"

LDR R1, =milliseconds

BL LCD\_PrintString

POP {R0, R1, PC}

// This is the last funct called, so I don't

// need to back anything up

**DisplayMinTime:**

// Congrats you made it

BL LCD\_Clear

LDR R1, =msg05

BL LCD\_PrintString

// Delay between text

MOV R1, #MSG\_DELAY

BL delay\_sec

// You got

BL LCD\_Clear

LDR R1, =msg06

BL LCD\_PrintString

// Newline

MOV R0, #1

MOV R1, #0

BL LCD\_MoveCursor

// (some number)

LDR R1, =correctCNT

LDR R2, [R1]

MOV R1, R2

BL LCD\_PrintNum

// Correct!

LDR R1, =msg07

BL LCD\_PrintString

// Delay between text

MOV R1, #MSG\_DELAY

BL delay\_sec

// Fastest time was

BL LCD\_Clear

LDR R1, =msg08

BL LCD\_PrintString

// newline

MOV R0, #1

MOV R1, #0

BL LCD\_MoveCursor

// (some time)

LDR R1, =minTime

LDR R2, [R1]

MOV R1, R2

BL LCD\_PrintNum

// millisecond suffix

LDR R1, =milliseconds

BL LCD\_PrintString

// Infinite loop until reset

B end

**end:** B end

**.section** **.data**

**time:**

**.word** 0

**minTime:**

**.word** 9999

**random:**

**.word** 0

**loops:**

**.byte** 0

**correctCNT:**

**.byte** 0

// Text prompts

**.section** **.rodata**

// msg01-04 are displayed on first startup

**msg01:**

**.asciz** "Evan's Hex conversion game!"

**msg02:**

**.asciz** "Number will display on LEDs"

**msg03:**

**.asciz** "Convert that number to hex!"

**msg04:**

**.asciz** "# = 0xE \* = 0xF"

// Ready set go & 0x is used every soft restart

**ReadySet:**

**.asciz** "Ready? Set! "

**go:**

**.asciz** "GO!"

**hex:**

**.asciz** "0x"

// Used when displaying if the answer was correct or not

**correct:**

**.asciz** " Correct!"

**incorrect:**

**.asciz** " Incorrect!"

// Time for last conversion

**timeMSG:**

**.asciz** "That took "

// Milliseconds suffix

**milliseconds:**

**.asciz** "ms"

// End of game text

**msg05:**

**.asciz** "Congrats! You made it!"

**msg06:**

**.asciz** "You got "

**msg07:**

**.asciz** " Correct!"

**msg08:**

**.asciz** "Fastest time was"

# Slight modification to the LCD driver to allow automatic newlines

// 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 - Current character

// R3 - Backup

// R4 - Backup

**LCD\_PrintString:**

PUSH {R1-R2, LR} // We don't need to back up R0 because it is a return

MOV R0, #0 // Iterator value

**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

// Newline Logic

MOV R3, R0 // Backup iterator

MOV R4, R1 // Backup address

CMP R0, #16 // Length of one line

ITTT EQ

MOVEQ R0, #1

MOVEQ R1, #0

BLEQ LCD\_MoveCursor

MOV R0, R3 // Restore iterator

MOV R1, R4 // Restore address

CMP R2, #0 // Redo comparison

ITT NE

POPNE {R1} // Restore address

BNE loop // Loop until we hit a null char

POP {R1-R2, PC} // Restore & return

// Evan Heinrich

// CE2801 sect. 011

// 10/29/2021

//

// File:

// random.S

// Description of File:

// Configures a timer to use as a random

// number generator

// -> Uses a 16-bit timer, TIM3

// (opt) Dependencies:

// N/A

// Assembler directives

.syntax unified

.cpu cortex-m4

.thumb

**.section** **.text**

.equ TIM3\_BASE, 0x40000400 // TIM2 base address

.equ RCC\_BASE, 0x40023800 // RCC base address

.equ APB1ENR, 0x40 // Offset from RCC base to APB1ENR

.equ TIM3EN, 1 << 1 // Location of TIM3 enabler is bit 1

.equ TIM\_CR1, 0x00 // Offset from TIM base to control reg.

.equ TIM\_CNT, 0x24 // Offset from TIM base to count reg.

.equ TIM\_CEN, 1 << 0 // Location for count enable

**.global** rnd\_Init

**.global** rnd\_Sample4

// Function: rnd\_Init

// Register-safe! Pushes all used registers to the stack

// Description:

// Configures TIM3 as a pseudo-random number generator

// Literally just starts TIM3 upcounting constantly

// Args:

// Void

// Returns:

// Void

// Register Usage:

// R1 - Addresses

// R2 - Scratch

**rnd\_Init:**

PUSH {R1, R2, LR}

// Enable TIM3 in RCC

LDR R1, =RCC\_BASE

LDR R2, [R1, #APB1ENR]

ORR R2, #TIM3EN

STR R2, [R1, #APB1ENR]

// Start TIM3 counting

LDR R1, =TIM3\_BASE

LDR R2, [R1, #TIM\_CR1]

ORR R2, #TIM\_CEN

STR R2, [R1, #TIM\_CR1]

POP {R1, R2, PC}

// Function: rnd\_Sample4

// Register-safe! Pushes all used registers to the stack

// Description:

// Samples the random number generator and

// returns the lower 4 bits of the sample

// Args:

// Void

// Returns:

// R0 - Pseudo-random number

// Register Usage:

// R0 - Return

// R1 - Addresses

// R2 - Scratch

**rnd\_Sample4:**

PUSH {R1, R2, LR}

// Sample counter

LDR R1, =TIM3\_BASE

LDR R2, [R1, #TIM\_CNT]

// Mask off all but lower 4 bits

MOV R1, #0xF

AND R0, R1, R2

POP {R1, R2, PC}

// Evan Heinrich

// CE2801 sect. 011

// 10/12/2021

//

// File:

// keypad.S

// Description of File:

// Lab 5 Keypad API

// Modified 10/26/2021 for Lab 7

// -> Converted from polling to interrupts

// -> Scan keypad on button press instead of manually polling

// (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\_ToChar

**.global** key\_ToHexChar

**.global** key\_ToNum

// Constants

.equ RCC\_BASE, 0x40023800 // Base address for RCC

.equ RCC\_AHB1ENR, 0x30 // Offset from RCC to AHB1ENR

.equ RCC\_APB2ENR, 0x44 // Offset from RCC to APB2ENR

.equ RCC\_GPIOCEN, 1 << 2 // Location of the GPIOC Enabler

.equ RCC\_SYSCFGEN, 1 << 14 // Location of the SYSCFG 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

.equ PIN\_INPUT, 0b00 // Mask to set a pin as an input

.equ PIN\_OUTPUT, 0b01 // Mask to set a pin as an output

.equ SYSCFG\_BASE, 0x40013800 // Base address of SYSCFG to configure EXTIx

.equ EXTI\_CR2, 0x0C // Offset to EXTICR2 for configuring EXTI4-EXTI7

.equ EXTI\_PC, 0b10 // Mask to set EXTIx to Port C

.equ EXTI\_BASE, 0x40013C00 // Base address for EXTI registers

.equ EXTI\_RTSR, 0x08 // Offset to rising trigger select

.equ EXTI\_PR, 0x14 // Offset to the external interrupt pending register

.equ EXTI\_IMR, 0x0 // Offset to interrupt mask register

.equ RISING\_MASK, 0b1111 // Mask to set rows as rising edge trigger

.equ EXTI\_UNMASK, 0b1111 // Mask to enable EXTI

.equ NVIC\_BASE, 0xE000E100 // Base address of the NVIC

.equ NVIC\_ISER, 0x00 // Offset to interrupt set enable register

.equ NVIC\_ENABLE, 1<<10|1<<23 // Mask to enable interrupts in the NVIC

.equ DEBOUNCE, 30

.equ ROW\_MASK, 0x0F // Masks off all but rows in a keycode

.equ COL\_MASK, 0xF0 // Masks off all but columns in a keycode

// 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

// Enable GPIOC

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

// Enable SYSCFG

LDR R2, [R1, #RCC\_APB2ENR]

ORR R2, #RCC\_SYSCFGEN

STR R2, [R1, #RCC\_APB2ENR]

// Configure rows as input, 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

// Configure Pull-down

LDR R2, [R1, #GPIO\_PUPDR] // Read the current pull-up/down register

LDR R3, =0xAAAA // Load the mask to set our pins to pull-down

ORR R2, R3 // Apply mask

STR R2, [R1, #GPIO\_PUPDR] // Write back to pull-up/down register

// Configure all EXTI's to use Port C

LDR R1, =SYSCFG\_BASE // Load system config base addr

MOV R3, #EXTI\_PC // Load mask to configure EXTI's as Port C

LDR R2, [R1, #EXTI\_CR2] // Read from external interrupt CR2

BFI R2, R3, #0, #4 // Insert Port C mask into EXTI4

BFI R2, R3, #4, #4 // " " EXTI5

BFI R2, R3, #8, #4 // " " EXTI6

BFI R2, R3, #12, #4 // " " EXTI7

STR R2, [R1, #EXTI\_CR2] // Write

// Configure rising edge trigger

LDR R1, =EXTI\_BASE // Load EXTI base addr

MOV R3, #RISING\_MASK // Load mask for our pins

LDR R2, [R1, #EXTI\_RTSR] // Read

BFI R2, R3, #4, #4 // Insert rising edge mask

STR R2, [R1, #EXTI\_RTSR] // Write

// Configure interrupt mask

// EXTI base addr still in R1

MOV R3, #EXTI\_UNMASK // Load EXTI enable mask

LDR R2, [R1, #EXTI\_IMR] // Read

BFI R2, R3, #4, #4 // Insert mask

STR R2, [R1, #EXTI\_IMR] // Write

// At this point, the interrupt is configured all the way through

// the EXTI controls, we just need to enable it in the NVIC

// Enable interrupts in the NVIC

LDR R1, =NVIC\_BASE // Load NVIC base addr

LDR R3, =NVIC\_ENABLE // Load mask to enable interrupts

LDR R2, [R1, #NVIC\_ISER] // Read

ORR R2, R3 // Apply mask

STR R2, [R1, #NVIC\_ISER] // Write

// At this point, the interrupts should be enabled

// Broadcast onto keypad to "arm" interrupts

// Otherwise a keypress connects ground to ground

LDR R1, =GPIOC\_BASE

MOV R2, #0xF

STR R2, [R1, #GPIO\_ODR]

POP {R1-R3, PC} // Restore & Return

// Function: Key\_ToChar

// Register-safe!

// Description:

// Converts a provided one-hot keycode into

// an ASCII character

//

// Keycode format: xxxx\_yyyy

// where the x's represent one-hot encoded row index

// where the y's represent one-hot encoded col index

// Args:

// R1 - One-hot encoded keycode

// Returns:

// R0 - ASCII code

// Register Use:

// R0 - Return

// R1 - Argument

// R2 - Row index

// R3 - Column index

// R4 - Masks

**key\_ToChar:**

PUSH {R1-R4, LR}

// Extract row index

MOV R4, #COL\_MASK // Load mask to extract rows

AND R2, R1, R4 // Extract rows

LSR R2, R2, #4 // Shift

// Extract column index

MOV R4, #ROW\_MASK // Load mask to extract columns

AND R3, R1, R4 // Extract columns

CMP R2, #0b0001 // Compare to row 1 case

BEQ firstRow

CMP R2, #0b0010 // Compare to row 2 case

BEQ secondRow

CMP R2, #0b0100 // Compare to row 3 case

BEQ thirdRow

CMP R2, #0b1000 // Compare to row 4 case

BEQ fourthRow

// Default case, return a null char

B 2f

**firstRow:**

// Row 1, Col 1

// AKA "1"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'1'

BEQ 1f

// Row 1, Col 2

// AKA "2"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'2'

BEQ 1f

// Row 1, Col 3

// AKA "3"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'3'

BEQ 1f

// Row 1, Col 4

// AKA "A"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'A'

BEQ 1f

B 2f

**secondRow:**

// Row 2, Col 1

// AKA "4"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'4'

BEQ 1f

// Row 2, Col 2

// AKA "5"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'5'

BEQ 1f

// Row 2, Col 3

// AKA "6"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'6'

BEQ 1f

// Row 2, Col 4

// AKA "B"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'B'

BEQ 1f

B 2f

**thirdRow:**

// Row 3, Col 1

// AKA "7"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'7'

BEQ 1f

// Row 3, Col 2

// AKA "8"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'8'

BEQ 1f

// Row 3, Col 3

// AKA "9"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'9'

BEQ 1f

// Row 3, Col 4

// AKA "C"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'C'

BEQ 1f

B 2f

**fourthRow:**

// Row 4, Col 1

// AKA "\*"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'\*'

BEQ 1f

// Row 4, Col 2

// AKA "0"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'0'

BEQ 1f

// Row 4, Col 3

// AKA "#"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'#'

BEQ 1f

// Row 4, Col 4

// AKA "D"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'D'

BEQ 1f

**2:**

MOV R0, #0

**1:**

POP {R1-R4, PC}

// Function: Key\_ToHexChar

// Register-safe!

// Description:

// Converts a provided one-hot keycode into

// an ASCII character

//

// Keycode format: xxxx\_yyyy

// where the x's represent one-hot encoded row index

// where the y's represent one-hot encoded col index

// Args:

// R1 - One-hot encoded keycode

// Returns:

// R0 - ASCII code

// Register Use:

// R0 - Return

// R1 - Argument

// R2 - Row index

// R3 - Column index

// R4 - Masks

**key\_ToHexChar:**

PUSH {R1-R4, LR}

// Extract row index

MOV R4, #COL\_MASK // Load mask to extract rows

AND R2, R1, R4 // Extract rows

LSR R2, R2, #4 // Shift

// Extract column index

MOV R4, #ROW\_MASK // Load mask to extract columns

AND R3, R1, R4 // Extract columns

CMP R2, #0b0001 // Compare to row 1 case

BEQ firstRowHex

CMP R2, #0b0010 // Compare to row 2 case

BEQ secondRowHex

CMP R2, #0b0100 // Compare to row 3 case

BEQ thirdRowHex

CMP R2, #0b1000 // Compare to row 4 case

BEQ fourthRowHex

// Default case, return a null char

B 2f

**firstRowHex:**

// Row 1, Col 1

// AKA "1"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'1'

BEQ 1f

// Row 1, Col 2

// AKA "2"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'2'

BEQ 1f

// Row 1, Col 3

// AKA "3"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'3'

BEQ 1f

// Row 1, Col 4

// AKA "A"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'A'

BEQ 1f

B 2f

**secondRowHex:**

// Row 2, Col 1

// AKA "4"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'4'

BEQ 1f

// Row 2, Col 2

// AKA "5"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'5'

BEQ 1f

// Row 2, Col 3

// AKA "6"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'6'

BEQ 1f

// Row 2, Col 4

// AKA "B"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'B'

BEQ 1f

B 2f

**thirdRowHex:**

// Row 3, Col 1

// AKA "7"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'7'

BEQ 1f

// Row 3, Col 2

// AKA "8"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'8'

BEQ 1f

// Row 3, Col 3

// AKA "9"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'9'

BEQ 1f

// Row 3, Col 4

// AKA "C"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'C'

BEQ 1f

B 2f

**fourthRowHex:**

// Row 4, Col 1

// AKA "\*"

CMP R3, #0b0001

ITT EQ

MOVEQ R0, #'F'

BEQ 1f

// Row 4, Col 2

// AKA "0"

CMP R3, #0b0010

ITT EQ

MOVEQ R0, #'0'

BEQ 1f

// Row 4, Col 3

// AKA "#"

CMP R3, #0b0100

ITT EQ

MOVEQ R0, #'E'

BEQ 1f

// Row 4, Col 4

// AKA "D"

CMP R3, #0b1000

ITT EQ

MOVEQ R0, #'D'

BEQ 1f

**2:**

MOV R0, #0

**1:**

POP {R1-R4, PC}

**key\_ToNum:**

PUSH {LR}

CMP R1, #'0'

ITT EQ

MOVEQ R0, #0

BEQ 1f

CMP R1, #'1'

ITT EQ

MOVEQ R0, #1

BEQ 1f

CMP R1, #'2'

ITT EQ

MOVEQ R0, #2

BEQ 1f

CMP R1, #'3'

ITT EQ

MOVEQ R0, #3

BEQ 1f

CMP R1, #'4'

ITT EQ

MOVEQ R0, #4

BEQ 1f

CMP R1, #'5'

ITT EQ

MOVEQ R0, #5

BEQ 1f

CMP R1, #'6'

ITT EQ

MOVEQ R0, #6

BEQ 1f

CMP R1, #'7'

ITT EQ

MOVEQ R0, #7

BEQ 1f

CMP R1, #'8'

ITT EQ

MOVEQ R0, #8

BEQ 1f

CMP R1, #'9'

ITT EQ

MOVEQ R0, #9

BEQ 1f

CMP R1, #'A'

ITT EQ

MOVEQ R0, #10

BEQ 1f

CMP R1, #'B'

ITT EQ

MOVEQ R0, #11

BEQ 1f

CMP R1, #'C'

ITT EQ

MOVEQ R0, #12

BEQ 1f

CMP R1, #'D'

ITT EQ

MOVEQ R0, #13

BEQ 1f

CMP R1, #'E'

ITT EQ

MOVEQ R0, #14

BEQ 1f

CMP R1, #'F'

ITT EQ

MOVEQ R0, #15

BEQ 1f

// Default case

MOV R0, #16

**1:**

POP {PC}

**.global** EXTI4\_IRQHandler

.thumb\_func

**EXTI4\_IRQHandler:**

// Disable external interrupts

LDR R1, =EXTI\_BASE

MOV R3, #0

LDR R2, [R1, #EXTI\_IMR]

BFI R2, R3, #4, #4

STR R2, [R1, #EXTI\_IMR]

// Clear pending

LDR R1, =EXTI\_BASE

MOV R2, #1<<4

STR R2, [R1, #EXTI\_PR]

PUSH {LR}

MOV R1, #DEBOUNCE

BL delay\_ms

// Switch PC4 from input to output

LDR R1, =GPIOC\_BASE // GPIO base addr

MOV R3, #PIN\_OUTPUT // Output mask

LDR R2, [R1, #GPIO\_MODER] // Read

BFI R2, R3, #(4\*2), #2 // Set PC4 as output

STR R2, [R1, #GPIO\_MODER] // Write

// Switch PC0-PC3 (columns) to inputs

MOV R3, #PIN\_INPUT

LDR R2, [R1, #GPIO\_MODER] // Read

BFI R2, R3, #0, #2 // Apply mask to PC0

BFI R2, R3, #(1\*2), #2 // Apply mask to PC1

BFI R2, R3, #(2\*2), #2 // Apply mask to PC2

BFI R2, R3, #(3\*2), #2 // Apply mask to PC3

STR R2, [R1, #GPIO\_MODER] // Write

// Broadcast a '1' onto the first row

MOV R3, #1

LDR R2, [R1, #GPIO\_ODR] // Read

BFI R2, R3, #4, #1 // Insert onto PC4

STR R2, [R1, #GPIO\_ODR] // Write

MOV R1, #5

BL delay\_us

// Scan inputs to determine which key is active

LDR R1, =GPIOC\_BASE

MOV R3, #0b00010000 // Prepare a mask

LDR R2, [R1, #GPIO\_IDR] // Read the IDR

// Mask off all but column values

AND R2, #0xF

// Insert into row 4 key mask

ORR R3, R2

// Write the key code into memory

LDR R1, =button

STRB R3, [R1] // R3 should only contain a byte

// Write a non-zero value to the flag

// in memory indicating the interrupt occured

LDR R1, =press

MOV R2, #1

STRB R2, [R1]

// Reset GPIOC to default config

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

// Re-enable external interrupts

LDR R1, =EXTI\_BASE

MOV R3, #0xF

LDR R2, [R1, #EXTI\_IMR]

BFI R2, R3, #4, #4

STR R2, [R1, #EXTI\_IMR]

POP {LR}

BX LR

**.global** EXTI9\_5\_IRQHandler

.thumb\_func

**EXTI9\_5\_IRQHandler:**

// Disable external interrupts

LDR R1, =EXTI\_BASE

MOV R3, #0

LDR R2, [R1, #EXTI\_IMR]

BFI R2, R3, #4, #4

STR R2, [R1, #EXTI\_IMR]

// Load pending to decode

LDR R4, [R1, #EXTI\_PR]

// Clear pending

LDR R1, =EXTI\_BASE

MOV R2, #0xE0

STR R2, [R1, #EXTI\_PR]

PUSH {LR}

MOV R1, #DEBOUNCE

BL delay\_ms

// Prepare row index

LSR R4, #4 // Move row index to the first 4 bits

// Switch rows to be outputs and columns to 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

// Broadcast row pattern

LDR R2, [R1, #GPIO\_ODR] // Read

BFI R2, R4, #4, #4 // Insert pattern from PR onto row outputs

STR R2, [R1, #GPIO\_ODR] // Write

// Allow charge to propagate

MOV R1, #5

BL delay\_us

// Scan column inputs

LDR R3, =0xFF // Mask to allow our keycode

LDR R1, =GPIOC\_BASE // GPIO base addr

LDR R2, [R1, #GPIO\_IDR] // Read IDR

AND R2, R3 // Apply mask

// Write the key code into memory

LDR R1, =button

STRB R2, [R1] // R2 should only contain a byte

// Write a non-zero value to the flag

// in memory indicating the interrupt occurred

LDR R1, =press

MOV R2, #1

STRB R2, [R1]

// Reset GPIOC to default config

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

// Re-enable external interrupts

LDR R1, =EXTI\_BASE

MOV R3, #0xF

LDR R2, [R1, #EXTI\_IMR]

BFI R2, R3, #4, #4

STR R2, [R1, #EXTI\_IMR]

POP {LR}

BX LR

**.section** **.data**

// Reserved for the button that was pressed

**.global** button

**button:** **.byte** 0

// Indicates when a key was pressed

**.global** press

**press:** **.byte** 0