```
@ Ryan Bentz
@ ECE 372 Project 1
@ PART 1
@ 2/12/18

@ This program uses GPIO and interrupts to detect a falling edge on a button press.
@ It uses the UART to communicate with and RC8660 Talk Talk and transmits a
@ predetermined message for every button press.

@ allocate memory for the stack
.data
.align 2
LED_STATUS:     .word 0x00
STACK1:         .rept 1024
                .word 0x00
                .endr
STACK2:         .rept 1024
                .word 0x00
                .endr

@ define word messages to send
@--------------------------------------------------------------------
CHAR_INDEX: .word 0x00          @ memory location for the counter
MESSAGE:
.byte 0x0D
.ascii "YOUR BLOOD PRESSURE IS 120 OVER 70"
.byte 0x00

@ enable linking for start and INT DIRECTOR
.text
.global _start
.global INT_DIRECTOR

@ begin main program
_start:

@ define used constants
.equ    BUTTON_PIN, 0x40000000      @ the Button pin/bit
.equ    MAXCHAR, 0x23               @ number of characters to send
@ TOTAL CHARACTERS + 1 = MAXCHAR

@ Initialize the stack frames
@--------------------------------------------------------------------
LDR R13, =STACK1            @ initialize stack one for supervisor mode
ADD R13, R13, #0x1000       @ point stack pointer to top of stack
CPS #0x12                   @ change to IRQ mode
LDR R13, =STACK2            @ initialize stack for IRQ mode
ADD R13, R13, #0x1000       @ point stack pointer to top of stack
CPS #0x13                   @ change back to supervisor mode

@ initialize clocks to the peripherals
@--------------------------------------------------------------------
```

```
@ Enable GPIO 1 clock
LDR R0, =0x44E000AC     @ CM_PER GPIO1 Register
LDR R1, [R0]            @ Read the register value
MOV R2, #0x00000002     @ value to turn on the GPIO module
ORR R1, R1, R2          @ Combine new value and existing register value
STR R1, [R0]            @ Write the value to the register

@ Enable the UART clock
LDR R0, =0x44E00070     @ CM_PER UART2 register
LDR R1, [R0]            @ read the register value
MOV R2, #0x00000002     @ value to turn on the UART module
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ delay and wait for peripherals to be ready
BL DELAY




@ Initialize the UART: Protocol, Baud Rate, and Interrupt Settings
@--------------------------------------------------------------------
@ disable the UART to access DLL and DLH
LDR R0, =0x48024020     @ MDR1 register
LDR R1, [R0]            @ read the register value
MOV R2, #0x07           @ put the UART in disabled state
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ switch to register configuration mode B
LDR R0, =0x4802400C     @ UART_LCR
LDR R1, [R0]            @ read the register value
MOV R2, #0xFF00         @ mask the upper half and zero the lower half
AND R1, R1, R2
MOV R2, #0x00BF         @ put the UART in disabled state
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ enable access to the IER UART register
@ (save the current status of the enhanced functions bit)
LDR R0, =0x48024008     @ UART_EFR
LDR R1, [R0]            @ read the register value
MOV R2, #0x0010         @ enhanced functions bit = bit 4 (5th place)
AND R1, R1, R2
MOV R3, R1              @ move to R3 for safe keeping
@ (enable enhanced functions write in EFR register)
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ switch to register operational mode
LDR R0, =0x4802400C     @ UART_LCR
MOV R1, #0x0000         @ write 0x00 to LCR
STR R1, [R0]            @ write the value to the register

```

```
@ clear the IER register to disable the UART interrupts
@ (disable sleep mode to access DLL and DLH registers
LDR R0, =0x48024004     @ UART_IER interrupt enable register
LDR R1, [R0]            @ read the register value
MOVW R2, #0xFFEF        @ sleep mode = bit 4
AND R1, R1, R2          @ clear the bit
STR R1, [R0]            @ write value to the register

@ switch to register configuration mode B
LDR R0, =0x4802400C     @ UART_LCR
LDR R1, [R0]            @ read the register value
MOV R2, #0xFF00         @ mask the upper half and zero the lower half
AND R1, R1, R2
MOV R2, #0x00BF         @ put the UART in disabled state
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ load the baud rate values for DLL and DLH
LDR R0, =0x48024000     @ UART_DLL
LDR R1, [R0]            @ read the register value
MOV R1, #0x004E
STR R1, [R0]            @ load the register value

LDR R0, =0x48024004     @ UART_DLH
LDR R1, [R0]            @ read the register value
MOV R1, #0x0000
STR R1, [R0]            @ load the register value

@ switch to register operational mode
LDR R0, =0x4802400C     @ UART_LCR
MOV R1, #0x0000         @ write 0x00 to LCR
STR R1, [R0]            @ write the value to the register

@ load the new interrupt configuration
LDR R0, =0x48024004     @ UART_IER
LDR R1, [R0]            @ load the register value
MOV R2, #0x0082         @ enable CTS and THR interrupts
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ switch to configuration mode B
LDR R0, =0x4802400C     @ UART_LCR
LDR R1, [R0]            @ read the register value
MOV R2, #0xFF00         @ mask the upper half and zero the lower half
AND R1, R1, R2
MOV R2, #0x00BF         @ put the UART in disabled state
ORR R1, R1, R2          @ combine new value and existing register value
STR R1, [R0]            @ write the value to the register

@ restore the EFR ENHANCED_EN bit (turns off enhanced feature mode)
LDR R0, =0x48024008     @ UART_EFR
LDR R1, [R0]            @ read the register value
ADD R1, R1, R3          @ restore the EN bit
STR R1, [R0]            @ write the value to the register
```

```

@ configure the UART protocol
@ switch to operational mode (set DIV_EN and BREAK_EN to 0)
LDR R0, =0x4802400C     @ UART_LCR
LDR R1, [R0]            @ read the register value
MOVW R2, #0xFF03        @ configure LCR register
AND R1, R1, R2
STR R1, [R0]            @ write the value to the register

@ set the new UART mode
LDR R0, =0x48024020     @ UART_MDR1
LDR R1, [R0]            @ read the register value
MOVW R2, #0xFF00        @ MDR[0:2] = 0x00 - enable 16-bit UART mode
AND R1, R1, R2          @ combine new value and existing value
STR R1, [R0]


@ Initialize the RC8660
@--------------------------------------------------------------------
@ set the baudrate
LDR     R0, =0x48024000         @ load the address for THR
MOV     R1, #0x0D
STRB    R1, [R0]                @ load the characters into THR

@ Initialize the GPIO
@--------------------------------------------------------------------
@ initialize GPIO pin for falling edge detect
LDR R0, =0x4804C14C     @ GPIO1_FALLING_DETECT
LDR R1, [R0]            @ read the register value
MOV R2, #BUTTON_PIN     @ word to program the register
ORR R1, R1, R2          @ modify the register value
STR R1, [R0]            @ write the value to the register

@ initialize GPIO pins for external interrupt
LDR R0, =0x4804C034     @ GPIO1_IRQSTATUS_SET0
LDR R1, [R0]            @ read the register value
MOV R2, #BUTTON_PIN     @ load the word to program the register
ORR R1, R1, R2          @ modify the current value with new value
STR R1, [R0]            @ write new value to the register

@ configure the UART TXD pin
LDR R0, =0x44E10954     @ CONTROL_MODULE.SPI0_D0 (B17)
LDR R1, [R0]            @ load value of register
MOVW R2, #0xFFF8            @ load mask for upper bits
AND R1, R1, R2          @ clear the mode bits [0:2]
MOV R2, #0x01           @ load new value for mode bits
ORR R1, R1, R2          @ make new register value
STR R1, [R0]

@ configure the UART CTS pin
LDR R0, =0x44E108C0     @ CONTROL_MODULE.LCD_DATA8
LDR R1, [R0]            @ load value of the register
MOVW R2, #0xFFF8            @ load mask for the upper bits
AND R1, R1, R2          @ clear the mode bits [0:2]
```

```
MOV R2, #0x06           @ load the value for mode bits
ORR R1, R1, R2          @ make new register value
STR R1, [R0]


@ Initialize the interrupt controller
@--------------------------------------------------------------------
@ initialize interrupt controller for GPIO pins
LDR R0, =0x482000E8     @ INTC_MIR_CLEAR3
LDR R1, [R0]            @ read the register value
MOV R2, #0x00000004     @ interrupt 98 = bit 3 in MIR3
ORR R1, R1, R2          @ unmask GPIO interrupt
STR R1, [R0]            @ write new value to the register


@ Initialize interrupts in the processor
@--------------------------------------------------------------------
MRS R3, CPSR    @ copy CPSR to R3
BIC R3, #0x80   @ clear bit 7
MSR CPSR_c, R3  @ write back to CPSR


MAIN_LOOP:
    NOP
    NOP
    B MAIN_LOOP
    B END

@--------------------------------------------------------------------
@ INTERRUPT SERVICE ROUTINE:
@ This procedure is hooked into the interrupt vector table to supercede
@ processor IRQ requests. It checks if the button was the source of the
@ interrupt and runs a specific procedure if it is the source of the interrupt
INT_DIRECTOR:
    STMFD SP!, {R0-R3, LR}      @ push registers on to the stack

@ check if GPIO was the interrupt source
    LDR R0, =0x482000F8 @ INTC_PENDING_IRQ3
    LDR R1, [R0]        @ read the pending interrupt register
    MOV R2, #0x00000004 @ check if GPIO triggered interrupt
    AND R1, R1, R2      @ test if GPIO was interrupt source
    CMP R1, #0x00
    BEQ UART_CHECK      @ if zero, GPIO was source
                        @ if nonzero, GPIO triggered the interrupt
                        @ check to see if the button was source
    LDR R0, =0x4804C02C @ GPIO1_IRQSTATUS_0
    LDR R1, [R0]        @ read value of register
    MOV R2, #BUTTON_PIN @ load the value for the button pin
    AND R1, R1, R2      @ make resultant word for comparison
    CMP R1, #0x00       @ compare to see if result is nonzero.
                        @ Nonzero = GPIO flag was set
    BNE BUTTON_SVC      @ go to button service procedure
    B PASS_ON           @ otherwise different GPIO caused interrupt

    UART_CHECK:         @ check if the UART caused the interrupt
    LDR R0, =0x482000D8 @ INTC_PENDING_IRQ2
    LDR R1, [R0]        @ read the pending interrupt register
    MOV R2, #0x00000400 @ check if UART triggered interrupt
    AND R1, R1, R2      @ test if UART was interrupt source
    CMP R1, #0x00       @ if nonzero, timer was the source
    BNE UART2_INT_SVC   @ otherwise timer was not the source

    @ exit int director ISR
    PASS_ON:
    @ re-enable IRQ interrupts in the processor
    MRS R3, CPSR            @ copy CPSR to R3
    BIC R3, #0x80           @ clear bit 7
    MSR CPSR_c, R3          @ write back to CPSR

    LDMFD SP!, {R0-R3, LR}  @ restore register states
    SUBS PC, LR, #4         @ return service to the system IRQ


@--------------------------------------------------------------------
@ BUTTON SERVICE PROCEDURE
@ This procedure handles the specific button service requirements
@ IRQ #98
BUTTON_SVC:
@ push SPSR on stack
    MRS R3, SPSR            @ copy the saved program status register
    STMFD R13!, {R3}        @ push on to stack

@ mask lower priority interrupts
@ already masked

@ initialize interrupt controller for UART
    LDR R0, =0x482000C8     @ INTC_MIR_CLEAR2
    LDR R1, [R0]            @ read the register value
    MOV R2, #0x00000400     @ interrupt 74 = bit 10 on MIR2
    ORR R1, R1, R2          @ unmask the UART interrupt
    STR R1, [R0]            @ write new value to the register

@ reset the GPIO interrupt request
    LDR R0, =0x4804C02C
    MOV R2, #BUTTON_PIN
    STR R2, [R0]

@ reset interrupt controller IRQ requests
    LDR R0, =0x48200048     @ INTC_CONTROL
    MOV R1, #0x01           @ value to reset IRQ generation
    STR R1, [R0]

@ disable IRQ for exit code critical region
@   MRS R3, CPSR
@   ORR R3, R3, #0x80
@   MSR CPSR_c, R3

@ unmask lower priority interrupts
@ not necessary

@ restore SPSR
    LDMFD R13!, {R3}        @ get saved SPSR
    MSR SPSR_cf, R3         @ restore the SPSR
    LDMFD SP!, {R0-R3, LR}  @ restore register states
    SUBS PC, LR, #4         @ return service to the system IRQ

@---------------------------------------------------------------------
@ UART2 SERVICE PROCEDURE
@ This procedure handles the UART interrupt logic
# 74
UART2_INT_SVC:
@ push SPSR on stack
    MRS R3, SPSR            @ copy the saved program status register
    STMFD R13!, {R3}        @ push on to stack

@ check if CTS and not THR was source of interrupt
    LDR R0, =0x48024008     @ interrupt identification register
    LDR R1, [R0]            @ load the state of the register
    MOVW R2, #0x0020        @ check the CTS bit
    AND R1, R1, R2          @ CTS = 0x10 at IIR[5:1]
    CMP R1, #0x0020
    BEQ UART_INT_CLOSE      @ if CTS but not THR -> exit
                            @ else check the THR bit
@ check THR bit
    LDR R0, =0x48024008     @ interrupt identification register
    LDR R1, [R0]            @ load the state of the register
    MOVW R2, 0x0002         @ check the THR bit
    AND R1, R1, R2          @ CTS = 0x01 at IIR [5:1]
    CMP R1, #0x0002
    BNE UART_MASK_THR       @ if not CTS but was THR
                            @ -> mask THR and exit
    BL  UART_TRANSMIT       @ else transmit next character
    B   UART_INT_CLOSE      @ come back from transmission
                            @ exit the procedure

UART_MASK_THR:
@ mask the THR interrupt
    LDR R0, =0x48024004     @ UART_IER
    LDR R1, [R0]            @ load the register value
    MOVW R2, #0xFFFD        @ disable the THR interrupt
    AND R1, R1, R2          @
    STR R1, [R0]            @ write the value to the register

UART_INT_CLOSE:
@ reset interrupt controller IRQ requests
    LDR R0, =0x48200048     @ INTC_CONTROL
    MOV R1, #0x01           @ value to reset IRQ generation
    STR R1, [R0]

@ re-enable IRQ interrupts in the processor
    MRS R3, CPSR    @ copy CPSR to R3
    BIC R3, #0x80   @ clear bit 7
    MSR CPSR_c, R3  @ write back to CPSR

    LDMFD R13!, {R3}
    MSR SPSR_cf, R3

    LDMFD SP!, {R0-R3, LR}      @ restore register states
    SUBS PC, LR, #4             @ return service to the system IRQ


@-------------------------------------------------------------------
@ UART TRANSMIT PROCEDURE
@ This procedure handles the UART transmission process
UART_TRANSMIT:
    STMFD   R13!, {R0-R3, R14}      @ push registers on to the stack

@ load the character counter
    LDR R0, =CHAR_INDEX     @ load the character index address
    LDR R1, [R0]            @ get the value of the counter

    LDR R2, =MESSAGE        @ load the base of the array
    ADD R2, R2, R1          @ add the array pointer and counter
                            @ (acts as an offset)
    LDRB R3, [R2]           @ get the value at that location

    LDR R2, =0x48024000     @ load the address for THR
    STRB R3, [R2]           @ load the characters into THR

    LDR R0, =CHAR_INDEX     @ load the index address
    LDR R1, [R0]            @ get the index value
    MOV R2, #0x01           @ value to increment
    ADD R1, R1, R2          @ increment index by 1
    STR R1, [R0]            @ update the counter with the new value
    CMP R1, #MAXCHAR
    BNE UART_TRANSMIT_CLOSE @ exit the UART
                            @ Otherwise, end of the transmission
@ reset the char index counter
    MOV R1, #0x00           @ reset the char index counter
    STR R1, [R0]            @ write to the location for the index

@ disable the UART interrupts in interrupt controller
    LDR R0, =0x482000CC     @ INCT_MIR_SET2
    LDR R1, [R0]
    MOV R2, #0x00000400     @ interrupt 74 = bit 10 on MIR2
    ORR R1, R1, R2          @ unmask the UART interrupt
    STR R1, [R0]            @ write new value to the register

UART_TRANSMIT_CLOSE:
    LDMFD R13!, {R0-R3, PC}     @ return to caller


@-------------------------------------------------------------------
@ DELAY ROUTINE
@ Necessary to give some buffer after we turn on the peripheral clocks
@ before we start accessing registers
DELAY:
STMFD R13!, {R4, R14}   @ save the register states
                        @ and link register location
LDR R4, =0x0022DCD5
D_LOOP:
    NOP
    SUBS R4, #1
    BNE D_LOOP
LDMFD R13!, {R4, PC}

END:
.END
```