```
1 @ Project 2 - Part 2
2 @ The program uses GPIO pins to turn the user LEDs on and off in a specific
  pattern
3 @ It uses a push button to determine if the LED flash sequence should be on or
  off
4 @ the button triggers an IRQ interrupt that handles the process of changing a
  variable
5 @ that keeps track of the flash sequence status
6 @ Ryan Bentz - 11/30/17
7
8 .data
9 .align 2
10 LED_STATUS:     .word 0x00
11 STACK1:         .rept 1024
12                 .word 0x00
13                 .endr
14 STACK2:         .rept 1024
15                 .word 0x00
16                 .endr
17 .text
18 .global _start
19 .global INT_DIRECTOR
20 _start:
21
22 @ Define the Register Addressed, Offsets, and write values to control the LEDs
23 .equ    LED0, 0x00200000            @ constant for LED0
24 .equ    LED1, 0x00400000            @ constant for LED1
25 .equ    LED2, 0x00800000            @ constant for LED2
26 .equ    LED3, 0x01000000            @ constant for LED3
27 .equ    DELAY_VAL, 0x0022DCD5
28 .equ    BUTTON_PIN, 0x40000000      @ constant for the Button pin/bit
29 .equ    LED_ON, 0x00000001          @ constant for LED flash = ON
30 .equ    LED_OFF, 0x00000000         @ constant for LED flash = OFF
31
32 @ initialize the stack frames
33 LDR R13, =STACK1                    @ initialize stack one for supervisor mode
34 ADD R13, R13, #0x1000               @ point stack pointer to top of stack
35 CPS #0x12                           @ change to IRQ mode
36 LDR R13, =STACK2                    @ initialize stack for IRQ mode
37 ADD R13, R13, #0x1000               @ point stack pointer to top of stack
38 CPS #0x13                           @ change back to supervisor mode
39
40 @Initialize the clock to GPIO 1
41 LDR R0, =0x44E000AC
42 MOV R2, #0x00000002     @ value to turn on the GPIO module
43 LDR R1, [R0]            @ Read the register value
44 ORR R1, R1, R2          @ Combine new value and existing register value
45 STR R1, [R0]            @ Write the value to the register
46
47 @ Set the LED pin state as low (OFF state)
48 @ default values for GPIO pin states is OFF
49
50 @Configure the LED pins as output
51 LDR R0, =0x4804C134     @ GPIO_OE
52 LDR R1, [R0]            @ Read the register value
53 @ Write zeroes to register bits to enable as output
54 MOV R2, #0xFE1FFFFF     @ Value to enable the pin
55 AND R1, R1, R2          @ Combine value to write new to register
56 STR R1, [R0]            @ write new value to the register
57
```

```
58 @ initialize GPIO pin for falling edge detect
59 LDR R0, =0x4804C14C     @ GPIO_FALLING_DETECT
60 LDR R1, [R0]            @ read the register value
61 MOV R2, #BUTTON_PIN     @ word to program the register
62 ORR R1, R1, R2          @ modify the register value
63 STR R1, [R0]
64
65 @ initialize GPIO pin for external interrupt
66 LDR R0, =0x4804C034     @ GPIO_IRQ_STATUS_SET0
67 LDR R1, [R0]            @ read the register value
68 MOV R2, #BUTTON_PIN     @ load the word to program the register
69 ORR R1, R1, R2          @ modify the current value with new value
70 STR R1, [R0]            @ write new value to the register
71
72 @ initialize interrupt controller for GPIO pins
73 @ unmask interrupt by writing a 1 to MIR_CLEAR which sets the bit to 0
74 @ 0 = interrupt enabled, 1 = interrupt disabled
75 LDR R0, =0x482000E8     @ INTC_MIR_CLEAR3
76 LDR R1, [R0]            @ read the register value
77 MOV R2, #0x00000004     @ interrupt 98 = bit 3 in MIR3
78 ORR R1, R1, R2          @ unmask GPIO interrupt
79 STR R1, [R0]            @ write new value to the register
80
81 @ initialize interrupts in the processor
82 MRS R3, CPSR    @ copy CPSR to R3
83 BIC R3, #0x80   @ clear bit 7
84 MSR CPSR_c, R3  @ write back to CPSR
85
86 @ initialize button state variable
87 @ button state already initialized as unchanged
88
89 @ MAIN LOOP
90 MAIN_LOOP:
91     NOP             @ do nothing and wait for interrupt
92     B MAIN_LOOP
93     B END
94
95
96 @-------------------------------------------------------------------------
97 @ INTERRUPT SERVICE ROUTINE:
98 @ This procedure is hooked into the interrupt vector table to supercede
99 @ processor IRQ requests. It checks if the button was the source of the
100 @ interrupt and runs a specific procedure if it is the source of the interrupt
101 INT_DIRECTOR:
102     STMFD SP!, {R0-R3, LR}      @ push registers on to the stack
103
104 @ check if GPIO was the interrupt source
105     LDR R0, =0x482000F8         @ INTC_PENDING_IRQ3
106     LDR R1, [R0]                @ read the pending interrupt register
107     MOV R2, #0x00000004         @ load test value to check if GPIO triggered
    interrupt
108     AND R1, R1, R2              @ test if GPIO was interrupt source
109     CMP R1, #0x00
110     @ if the result is zero, GPIO was not source of interrupt
111     BEQ PASS_ON
112
113 @ If result is nonzero, GPIO triggered the interrupt
114 @ check to see if the button was the source of the GPIO interrupt
115     LDR R0, =0x4804C02C         @ GPIO_IRQSTATUS0
116     LDR R1, [R0]                @ read value of register
```

```
117     MOV R2, #BUTTON_PIN         @ load the value for the button pin
118     AND R1, R1, R2              @ make resultant word for comparison
119     CMP R1, #0x00               @ compare to see if result is nonzero.
120                                 @ Nonzero = GPIO flag was set
121 @ If button was pressed go to button service procedure
122     BNE BUTTON_SVC
123     @ ELSE
124         @ Do nothing
125
126 @ exit int director ISR
127 PASS_ON:
128 @ re-enable IRQ interrupts in the processor
129     MRS R3, CPSR    @ copy CPSR to R3
130     BIC R3, #0x80   @ clear bit 7
131     MSR CPSR_c, R3  @ write back to CPSR
132 @ restore register states and exit
133     LDMFD SP!, {R0-R3, LR}      @ restore register states
134     SUBS PC, LR, #4             @ return service to the system IRQ
135
136
137 @-------------------------------------------------------------------------
138 @ BUTTON SERVICE PROCEDURE
139 @ This procedure handles the specific button service requirements
140 @ It resets the GPIO IRQ flag and changes the LED flash status variable
141 BUTTON_SVC:
142 @ push saved program status register on stack
143     MRS R3, SPSR
144     STMFD R13!, {R3}
145
146 @ mask lower priority interrupts
147 @ not necessary
148
149 @ reset the GPIO interrupt request
150     @ R0 currently has the address for the GPIO IRQ status register
151     @ R2 currently has the word to check/write the button pin
152     STR R2, [R0]
153
154 @ change the LED flash status
155     LDR R0, =LED_STATUS     @ load the address for the button variable
156     LDR R1, [R0]            @ get status of the LED flashing
157     MOV R2, #LED_ON         @ load the test value to see if it is on
158     CMP R1, R2              @ compare with current state
159     MOVEQ R1, #LED_OFF      @ if equal = ON, load the status to turn it off
160     MOVNE R1, #LED_ON       @ if not equal = OFF, load the status to turn it
    on
161     STR R1, [R0]            @ write the new status to the variable
162
163 @ reset interrupt controller IRQ requests
164     LDR R0, =0x48200048     @ INTCT_CONTROL
165     LDR R1, [R0]            @ read the register value
166     MOV R2, #0x01           @ value to reset IRQ generation
167     STR R2, [R0]
168
169 @ re-enable IRQ interrupts in the processor
170     MRS R3, CPSR    @ copy CPSR to R3
171     BIC R3, #0x80   @ clear bit 7
172     MSR CPSR_c, R3  @ write back to CPSR
173
174 @ check the LED flash status
175     LDR R0, =LED_STATUS     @ load the address for the button variable
```

```
176     LDR R1, [R0]            @ get status of the LED flashing
177     MOV R2, #LED_ON
178     CMP R1, R2                  @ check to see if the LED status is ON
179     BEQ LED_FLASH               @ IF LED status is ON, go to LED flash
    sequence
180                                 @ otherwise exit back to main
181 @ disable IRQs
182     MRS R3, CPSR
183     ORR R3, R3, #0x80
184     MSR CPSR_c, R3
185
186 @ restore saved program status register from recent BUTTON SVC
187     LDMFD R13!, {R3}
188     MSR SPSR_cf, R3
189 @ restore register states from recent INT DIRECTOR
190     LDMFD SP!, {R0-R3, LR}      @ restore register states
191
192 @ pop delay stuff off the stack, not needed
193     LDMFD R13!, {R0, R1}
194
195 @ restore saved program status register from first BUTTON SVC
196     LDMFD R13!, {R3}
197     MSR SPSR_cf, R3
198 @ restore register states
199     LDMFD SP!, {R0-R3, LR}      @ restore register states
200     SUBS PC, LR, #4             @ return service to main
201
202
203 @-------------------------------------------------------------------------
204 @ LED Flasher Subroutine
205 @ This procedure handles the LED flash sequence
206 LED_FLASH:
207 @ save the register states and link register location
208 @STMFD R13!, {R0-R2, R14}
209
210 @ Turn on the LEDs one by one
211     LDR R0, =0x4804C13C
212     MOV R1, #LED0       @ Turn on LED 0
213     STR R1, [R0]
214     BL DELAY            @ Wait 1 second
215     ORR R1, #LED1       @ Turn on LED 1
216     STR R1, [R0]
217     BL DELAY            @ Wait 1 second
218     ORR R1, #LED2       @ Turn on LED 2
219     STR R1, [R0]
220     BL DELAY            @ Wait 1 second
221     ORR R1, #LED3       @ Turn on LED 3
222     STR R1, [R0]
223     BL DELAY            @ Wait 1 second
224 @ Turn off the LEDs all at once
225     MOV R1, #0x00
226     STR R1, [R0]            @ Write the value to the register
227     BL DELAY                @ Wait 1 second
228
229 @ check if the LED flash status is on
230 @ (status is turned off in button ISR)
231     LDR R0, =LED_STATUS         @ get the address for the LED status variable
232     LDR R1, [R0]                @ get the state of the variable
233     MOV R2, #LED_ON
234     CMP R1, R2                  @ check to see if the LED status is ON
```

```
235     @ IF LED status is ON, continue
236     BEQ LED_FLASH               @ IF button flag was set
237
238
239 @--------------------------------------------------------------------------
240 @ Delay Loop Subroutine
241 @ Handles the delay loop timing
242 DELAY:
243 STMFD R13!, {R4, R14}   @ save the register states and link register location
244 LDR R4, =DELAY_VAL
245 D_LOOP:
246     NOP
247     SUBS R4, #1
248     BNE D_LOOP
249 LDMFD R13!, {R4, PC}
250
251 END:
252 .END
```