```
1 @ Ryan Bentz
2 @ ECE 372 Project 1
3 @ PART 1
4 @ 2/12/18
5
6 @ This program uses GPIO and interrupts to detect a falling edge on a
  button press.
7 @ It uses the UART to communicate with and RC8660 Talk Talk and
  transmits a
8 @ predetermined message for every button press.
9
10 @ allocate memory for the stack
11 .data
12 .align 2
13 LED_STATUS:      .word 0x00
14 STACK1:          .rept 1024
15                  .word 0x00
16                  .endr
17 STACK2:          .rept 1024
18                  .word 0x00
19                  .endr
20
21 @ define word messages to send
22 @-------------------------------------------------------------------
23 CHAR_INDEX: .word 0x00          @ memory location for the counter
24 MESSAGE:
25 .byte 0x0D
26 .ascii "YOUR BLOOD PRESSURE IS 120 OVER 70"
27 .byte 0x00
28
29 @ enable linking for start and INT DIRECTOR
30 .text
31 .global _start
32 .global INT_DIRECTOR
33
34 @ begin main program
35 _start:
36
37 @ define used constants
38 .equ    BUTTON_PIN, 0x40000000      @ the Button pin/bit
39 .equ    MAXCHAR, 0x23               @ number of characters to send
40 @ TOTAL CHARACTERS + 1 = MAXCHAR
41
42 @ Initialize the stack frames
43 @-------------------------------------------------------------------
44 LDR R13, =STACK1            @ initialize stack one for supervisor mode
45 ADD R13, R13, #0x1000       @ point stack pointer to top of stack
46 CPS #0x12                   @ change to IRQ mode
47 LDR R13, =STACK2            @ initialize stack for IRQ mode
48 ADD R13, R13, #0x1000       @ point stack pointer to top of stack
49 CPS #0x13                   @ change back to supervisor mode
50
51 @ initialize clocks to the peripherals
52 @-------------------------------------------------------------------
```

```
53 @ Enable GPIO 1 clock
54 LDR R0, =0x44E000AC      @ CM_PER GPIO1 Register
55 LDR R1, [R0]             @ Read the register value
56 MOV R2, #0x00000002      @ value to turn on the GPIO module
57 ORR R1, R1, R2           @ Combine new value and existing register value
58 STR R1, [R0]             @ Write the value to the register
59
60 @ Enable the UART clock
61 LDR R0, =0x44E00070      @ CM_PER UART2 register
62 LDR R1, [R0]             @ read the register value
63 MOV R2, #0x00000002      @ value to turn on the UART module
64 ORR R1, R1, R2           @ combine new value and existing register value
65 STR R1, [R0]             @ write the value to the register
66
67 @ delay and wait for peripherals to be ready
68 BL DELAY
69
70
71
72
73 @ Initialize the UART: Protocol, Baud Rate, and Interrupt Settings
74 @-------------------------------------------------------------------
75 @ disable the UART to access DLL and DLH
76 LDR R0, =0x48024020      @ MDR1 register
77 LDR R1, [R0]             @ read the register value
78 MOV R2, #0x07            @ put the UART in disabled state
79 ORR R1, R1, R2           @ combine new value and existing register value
80 STR R1, [R0]             @ write the value to the register
81
82 @ switch to register configuration mode B
83 LDR R0, =0x4802400C      @ UART_LCR
84 LDR R1, [R0]             @ read the register value
85 MOV R2, #0xFF00          @ mask the upper half and zero the lower half
86 AND R1, R1, R2
87 MOV R2, #0x00BF          @ put the UART in disabled state
88 ORR R1, R1, R2           @ combine new value and existing register value
89 STR R1, [R0]             @ write the value to the register
90
91 @ enable access to the IER UART register
92 @ (save the current status of the enhanced functions bit)
93 LDR R0, =0x48024008      @ UART_EFR
94 LDR R1, [R0]             @ read the register value
95 MOV R2, #0x0010          @ enhanced functions bit = bit 4 (5th place)
96 AND R1, R1, R2
97 MOV R3, R1              @ move to R3 for safe keeping
98 @ (enable enhanced functions write in EFR register)
99 ORR R1, R1, R2           @ combine new value and existing register value
100 STR R1, [R0]            @ write the value to the register
101
102 @ switch to register operational mode
103 LDR R0, =0x4802400C     @ UART_LCR
104 MOV R1, #0x0000         @ write 0x00 to LCR
105 STR R1, [R0]            @ write the value to the register
106
```

```
107 @ clear the IER register to disable the UART interrupts
108 @ (disable sleep mode to access DLL and DLH registers
109 LDR R0, =0x48024004     @ UART_IER interrupt enable register
110 LDR R1, [R0]           @ read the register value
111 MOVW R2, #0xFFEF       @ sleep mode = bit 4
112 AND R1, R1, R2         @ clear the bit
113 STR R1, [R0]           @ write value to the register
114
115 @ switch to register configuration mode B
116 LDR R0, =0x4802400C     @ UART_LCR
117 LDR R1, [R0]           @ read the register value
118 MOV R2, #0xFF00         @ mask the upper half and zero the lower half
119 AND R1, R1, R2
120 MOV R2, #0x00BF         @ put the UART in disabled state
121 ORR R1, R1, R2          @ combine new value and existing register value
122 STR R1, [R0]           @ write the value to the register
123
124 @ load the baud rate values for DLL and DLH
125 LDR R0, =0x48024000     @ UART_DLL
126 LDR R1, [R0]           @ read the register value
127 MOV R1, #0x004E
128 STR R1, [R0]           @ load the register value
129
130 LDR R0, =0x48024004     @ UART_DLH
131 LDR R1, [R0]           @ read the register value
132 MOV R1, #0x0000
133 STR R1, [R0]           @ load the register value
134
135 @ switch to register operational mode
136 LDR R0, =0x4802400C     @ UART_LCR
137 MOV R1, #0x0000         @ write 0x00 to LCR
138 STR R1, [R0]           @ write the value to the register
139
140 @ load the new interrupt configuration
141 LDR R0, =0x48024004     @ UART_IER
142 LDR R1, [R0]           @ load the register value
143 MOV R2, #0x0082         @ enable CTS and THR interrupts
144 ORR R1, R1, R2          @ combine new value and existing register value
145 STR R1, [R0]           @ write the value to the register
146
147 @ switch to configuration mode B
148 LDR R0, =0x4802400C     @ UART_LCR
149 LDR R1, [R0]           @ read the register value
150 MOV R2, #0xFF00         @ mask the upper half and zero the lower half
151 AND R1, R1, R2
152 MOV R2, #0x00BF         @ put the UART in disabled state
153 ORR R1, R1, R2          @ combine new value and existing register value
154 STR R1, [R0]           @ write the value to the register
155
156 @ restore the EFR ENHANCED_EN bit (turns off enhanced feature mode)
157 LDR R0, =0x48024008     @ UART_EFR
158 LDR R1, [R0]           @ read the register value
159 ADD R1, R1, R3          @ restore the EN bit
160 STR R1, [R0]           @ write the value to the register
```

```
161
162 @ configure the UART protocol
163 @ switch to operational mode (set DIV_EN and BREAK_EN to 0)
164 LDR R0, =0x4802400C     @ UART_LCR
165 LDR R1, [R0]           @ read the register value
166 MOVW R2, #0xFF03        @ configure LCR register
167 AND R1, R1, R2
168 STR R1, [R0]           @ write the value to the register
169
170 @ set the new UART mode
171 LDR R0, =0x48024020     @ UART_MDR1
172 LDR R1, [R0]           @ read the register value
173 MOVW R2, #0xFF00        @ MDR[0:2] = 0x00 - enable 16-bit UART mode
174 AND R1, R1, R2          @ combine new value and existing value
175 STR R1, [R0]
176
177
178 @ Initialize the RC8660
179 @-----------------------------------------------------------------------
180 @ set the baudrate
181 LDR     R0, =0x48024000         @ load the address for THR
182 MOV     R1, #0x0D
183 STRB    R1, [R0]                @ load the characters into THR
184
185 @ Initialize the GPIO
186 @-----------------------------------------------------------------------
187 @ initialize GPIO pin for falling edge detect
188 LDR R0, =0x4804C14C     @ GPIO1_FALLING_DETECT
189 LDR R1, [R0]           @ read the register value
190 MOV R2, #BUTTON_PIN     @ word to program the register
191 ORR R1, R1, R2          @ modify the register value
192 STR R1, [R0]           @ write the value to the register
193
194 @ initialize GPIO pins for external interrupt
195 LDR R0, =0x4804C034     @ GPIO1_IRQSTATUS_SET0
196 LDR R1, [R0]           @ read the register value
197 MOV R2, #BUTTON_PIN     @ load the word to program the register
198 ORR R1, R1, R2          @ modify the current value with new value
199 STR R1, [R0]           @ write new value to the register
200
201 @ configure the UART TXD pin
202 LDR R0, =0x44E10954     @ CONTROL_MODULE.SPI0_D0 (B17)
203 LDR R1, [R0]           @ load value of register
204 MOVW R2, #0xFFF8           @ load mask for upper bits
205 AND R1, R1, R2          @ clear the mode bits [0:2]
206 MOV R2, #0x01           @ load new value for mode bits
207 ORR R1, R1, R2          @ make new register value
208 STR R1, [R0]
209
210 @ configure the UART CTS pin
211 LDR R0, =0x44E108C0     @ CONTROL_MODULE.LCD_DATA8
212 LDR R1, [R0]           @ load value of the register
213 MOVW R2, #0xFFF8           @ load mask for the upper bits
214 AND R1, R1, R2          @ clear the mode bits [0:2]
```

```
215 MOV R2, #0x06           @ load the value for mode bits
216 ORR R1, R1, R2          @ make new register value
217 STR R1, [R0]
218
219
220 @ Initialize the interrupt controller
221 @---------------------------------------------------------------------
222 @ initialize interrupt controller for GPIO pins
223 LDR R0, =0x482000E8     @ INTC_MIR_CLEAR3
224 LDR R1, [R0]            @ read the register value
225 MOV R2, #0x00000004     @ interrupt 98 = bit 3 in MIR3
226 ORR R1, R1, R2          @ unmask GPIO interrupt
227 STR R1, [R0]            @ write new value to the register
228
229
230 @ Initialize interrupts in the processor
231 @---------------------------------------------------------------------
232 MRS R3, CPSR    @ copy CPSR to R3
233 BIC R3, #0x80   @ clear bit 7
234 MSR CPSR_c, R3  @ write back to CPSR
235
236
237 MAIN_LOOP:
238     NOP
239     NOP
240     B MAIN_LOOP
241     B END
242
243 @---------------------------------------------------------------------
244 @ INTERRUPT SERVICE ROUTINE:
245 @ This procedure is hooked into the interrupt vector table to supercede
246 @ processor IRQ requests. It checks if the button was the source of the
247 @ interrupt and runs a specific procedure if it is the source of the
    interrupt
248 INT_DIRECTOR:
249     STMFD SP!, {R0-R3, LR}      @ push registers on to the stack
250
251 @ check if GPIO was the interrupt source
252     LDR R0, =0x482000F8 @ INTC_PENDING_IRQ3
253     LDR R1, [R0]        @ read the pending interrupt register
254     MOV R2, #0x00000004 @ check if GPIO triggered interrupt
255     AND R1, R1, R2      @ test if GPIO was interrupt source
256     CMP R1, #0x00
257     BEQ UART_CHECK      @ if zero, GPIO was source
258                         @ if nonzero, GPIO triggered the interrupt
259                         @ check to see if the button was source
260     LDR R0, =0x4804C02C @ GPIO1_IRQSTATUS_0
261     LDR R1, [R0]        @ read value of register
262     MOV R2, #BUTTON_PIN @ load the value for the button pin
263     AND R1, R1, R2      @ make resultant word for comparison
264     CMP R1, #0x00       @ compare to see if result is nonzero.
265                         @ Nonzero = GPIO flag was set
266     BNE BUTTON_SVC      @ go to button service procedure
267     B PASS_ON           @ otherwise different GPIO caused interrupt
```

```
268
269     UART_CHECK:          @ check if the UART caused the interrupt
270     LDR R0, =0x482000D8 @ INTC_PENDING_IRQ2
271     LDR R1, [R0]         @ read the pending interrupt register
272     MOV R2, #0x00000400 @ check if UART triggered interrupt
273     AND R1, R1, R2       @ test if UART was interrupt source
274     CMP R1, #0x00        @ if nonzero, timer was the source
275     BNE UART2_INT_SVC    @ otherwise timer was not the source
276
277     @ exit int director ISR
278     PASS_ON:
279     @ re-enable IRQ interrupts in the processor
280     MRS R3, CPSR             @ copy CPSR to R3
281     BIC R3, #0x80            @ clear bit 7
282     MSR CPSR_c, R3           @ write back to CPSR
283
284     LDMFD SP!, {R0-R3, LR}  @ restore register states
285     SUBS PC, LR, #4          @ return service to the system IRQ
286
287
288 @-------------------------------------------------------------------
289 @ BUTTON SERVICE PROCEDURE
290 @ This procedure handles the specific button service requirements
291 @ IRQ #98
292 BUTTON_SVC:
293 @ push SPSR on stack
294     MRS R3, SPSR             @ copy the saved program status register
295     STMFD R13!, {R3}         @ push on to stack
296
297 @ mask lower priority interrupts
298 @ already masked
299
300 @ initialize interrupt controller for UART
301     LDR R0, =0x482000C8      @ INTC_MIR_CLEAR2
302     LDR R1, [R0]             @ read the register value
303     MOV R2, #0x00000400      @ interrupt 74 = bit 10 on MIR2
304     ORR R1, R1, R2           @ unmask the UART interrupt
305     STR R1, [R0]             @ write new value to the register
306
307 @ reset the GPIO interrupt request
308     LDR R0, =0x4804C02C
309     MOV R2, #BUTTON_PIN
310     STR R2, [R0]
311
312 @ reset interrupt controller IRQ requests
313     LDR R0, =0x48200048      @ INTC_CONTROL
314     MOV R1, #0x01            @ value to reset IRQ generation
315     STR R1, [R0]
316
317 @ disable IRQ for exit code critical region
318 @   MRS R3, CPSR
319 @   ORR R3, R3, #0x80
320 @   MSR CPSR_c, R3
321
```

```
322 @ unmask lower priority interrupts
323 @ not necessary
324
325 @ restore SPSR
326     LDMFD R13!, {R3}         @ get saved SPSR
327     MSR SPSR_cf, R3          @ restore the SPSR
328     LDMFD SP!, {R0-R3, LR}   @ restore register states
329     SUBS PC, LR, #4          @ return service to the system IRQ
330
331 @-------------------------------------------------------------------
332 @ UART2 SERVICE PROCEDURE
333 @ This procedure handles the UART interrupt logic
334 # 74
335 UART2_INT_SVC:
336 @ push SPSR on stack
337     MRS R3, SPSR             @ copy the saved program status register
338     STMFD R13!, {R3}         @ push on to stack
339
340 @ check if CTS and not THR was source of interrupt
341     LDR R0, =0x48024008      @ interrupt identification register
342     LDR R1, [R0]             @ load the state of the register
343     MOVW R2, #0x0020         @ check the CTS bit
344     AND R1, R1, R2           @ CTS = 0x10 at IIR[5:1]
345     CMP R1, #0x0020
346     BEQ UART_INT_CLOSE       @ if CTS but not THR -> exit
347                              @ else check the THR bit
348 @ check THR bit
349     LDR R0, =0x48024008      @ interrupt identification register
350     LDR R1, [R0]             @ load the state of the register
351     MOVW R2, 0x0002          @ check the THR bit
352     AND R1, R1, R2           @ CTS = 0x01 at IIR [5:1]
353     CMP R1, #0x0002
354     BNE UART_MASK_THR        @ if not CTS but was THR
355                              @ -> mask THR and exit
356     BL  UART_TRANSMIT        @ else transmit next character
357     B   UART_INT_CLOSE       @ come back from transmission
358                              @ exit the procedure
359
360 UART_MASK_THR:
361 @ mask the THR interrupt
362     LDR R0, =0x48024004      @ UART_IER
363     LDR R1, [R0]             @ load the register value
364     MOVW R2, #0xFFFD         @ disable the THR interrupt
365     AND R1, R1, R2           @
366     STR R1, [R0]             @ write the value to the register
367
368 UART_INT_CLOSE:
369 @ reset interrupt controller IRQ requests
370     LDR R0, =0x48200048      @ INTC_CONTROL
371     MOV R1, #0x01            @ value to reset IRQ generation
372     STR R1, [R0]
373
374 @ re-enable IRQ interrupts in the processor
375     MRS R3, CPSR    @ copy CPSR to R3
```

```
376     BIC R3, #0x80   @ clear bit 7
377     MSR CPSR_c, R3  @ write back to CPSR
378
379     LDMFD R13!, {R3}
380     MSR SPSR_cf, R3
381
382     LDMFD SP!, {R0-R3, LR}      @ restore register states
383     SUBS PC, LR, #4             @ return service to the system IRQ
384
385
386 @---------------------------------------------------------------------
387 @ UART TRANSMIT PROCEDURE
388 @ This procedure handles the UART transmission process
389 UART_TRANSMIT:
390     STMFD   R13!, {R0-R3, R14}      @ push registers on to the stack
391
392 @ load the character counter
393     LDR R0, =CHAR_INDEX     @ load the character index address
394     LDR R1, [R0]            @ get the value of the counter
395
396     LDR R2, =MESSAGE        @ load the base of the array
397     ADD R2, R2, R1          @ add the array pointer and counter
398                             @ (acts as an offset)
399     LDRB R3, [R2]           @ get the value at that location
400
401     LDR R2, =0x48024000     @ load the address for THR
402     STRB R3, [R2]           @ load the characters into THR
403
404     LDR R0, =CHAR_INDEX     @ load the index address
405     LDR R1, [R0]            @ get the index value
406     MOV R2, #0x01           @ value to increment
407     ADD R1, R1, R2          @ increment index by 1
408     STR R1, [R0]            @ update the counter with the new value
409     CMP R1, #MAXCHAR
410     BNE UART_TRANSMIT_CLOSE @ exit the UART
411                             @ Otherwise, end of the transmission
412 @ reset the char index counter
413     MOV R1, #0x00           @ reset the char index counter
414     STR R1, [R0]            @ write to the location for the index
415
416 @ disable the UART interrupts in interrupt controller
417     LDR R0, =0x482000CC     @ INCT_MIR_SET2
418     LDR R1, [R0]
419     MOV R2, #0x00000400     @ interrupt 74 = bit 10 on MIR2
420     ORR R1, R1, R2          @ unmask the UART interrupt
421     STR R1, [R0]            @ write new value to the register
422
423 UART_TRANSMIT_CLOSE:
424     LDMFD R13!, {R0-R3, PC}     @ return to caller
425
426
427 @---------------------------------------------------------------------
428 @ DELAY ROUTINE
429 @ Necessary to give some buffer after we turn on the peripheral clocks
```

```
430 @ before we start accessing registers
431 DELAY:
432 STMFD R13!, {R4, R14}   @ save the register states
433                          @ and link register location
434 LDR R4, =0x0022DCD5
435 D_LOOP:
436     NOP
437     SUBS R4, #1
438     BNE D_LOOP
439 LDMFD R13!, {R4, PC}
440
441 END:
442 .END
```