```
1 @ Ryan Bentz
2 @ ECE 372 Project 1
3 @ PART 2
4 @ 2/12/18
5
6 @ This program waits for a button press to trigger the external
  interrupt.
7 @ When the button is pressed, the program sends a message to the RC8660
  and
8 @ initializes the timer for a 10s period. When the timer overflows, it
  sends
9 @ the same message. The program continues on until the button is
  pressed again
10 @ and the timer and UART are disabled.
11
12 @ allocate memory for the stack
13 .data
14 .align 2
15 LED_STATUS:      .word 0x00
16 STACK1:          .rept 1024
17                  .word 0x00
18                  .endr
19 STACK2:          .rept 1024
20                  .word 0x00
21                  .endr
22
23 @ define word messages to send
24 @-------------------------------------------------------------------
25 STATUS: .word 0x00
26 CHAR_INDEX: .word 0x00       @ memory location for the counter
27 MESSAGE:
28 .byte 0x0D
29 .ascii "YOUR BLOOD PRESSURE IS 120 OVER 70"
30 .byte 0x00
31
32
33
34 @ enable linking for start and INT DIRECTOR
35 .text
36 .global _start
37 .global INT_DIRECTOR
38
39 @ begin main program
40 _start:
41
42 @ define used constants
43 .equ    TIM_COUNT_VAL, 0xFFFAFFFF
44 .equ    BUTTON_PIN, 0x40000000      @ the Button pin/bit
45 .equ    MAXCHAR, 0x23               @ number of characters to send
46 @ TOTAL CHARACTERS + 1 = MAXCHAR
47
48 @ Initialize the stack frames
49 @-------------------------------------------------------------------
50 LDR R13, =STACK1        @ initialize stack one for supervisor mode
```

```
 51 ADD R13, R13, #0x1000   @ point stack pointer to top of stack
 52 CPS #0x12               @ change to IRQ mode
 53 LDR R13, =STACK2        @ initialize stack for IRQ mode
 54 ADD R13, R13, #0x1000   @ point stack pointer to top of stack
 55 CPS #0x13               @ change back to supervisor mode
 56
 57
 58 @ Enable the timer clock source in CMDLL at 32.768
 59 @---------------------------------------------------------------------
 60 LDR R0, =0x44E0050C     @ CM_DLL Timer 3 Register
 61 LDR R1, [R0]            @ read the register value
 62 MOV R2, #0x02           @ enable 32kHz clock
 63 ORR R1, R1, R2          @ modify the current value
 64 STR R1, [R0]            @ write new value to register
 65
 66
 67 @ Initialize clocks to the peripherals
 68 @---------------------------------------------------------------------
 69 @ Enable GPIO 1 clock
 70 LDR R0, =0x44E000AC     @ CM_PER GPIO1 Register
 71 LDR R1, [R0]            @ Read the register value
 72 MOV R2, #0x00000002     @ value to turn on the GPIO module
 73 ORR R1, R1, R2          @ Combine new value and existing register value
 74 STR R1, [R0]            @ Write the value to the register
 75
 76 @ Enable the UART clock
 77 LDR R0, =0x44E00070     @ CM_PER UART2 register
 78 LDR R1, [R0]            @ read the register value
 79 MOV R2, #0x00000002     @ value to turn on the UART module
 80 ORR R1, R1, R2          @ combine new value and existing register value
 81 STR R1, [R0]            @ write the value to the register
 82
 83 @ Enable Timer 3 clock
 84 LDR R0, =0x44E00084     @ CM_PER Timer 3 Register
 85 LDR R1, [R0]            @ get the value of the register
 86 MOV R2, #0x02
 87 ORR R1, R1, R2          @ modify the register
 88 STR R1, [R0]
 89
 90 @ delay and wait for peripherals to be ready
 91 BL DELAY
 92
 93
 94 @ Initialize the UART: Protocol, Baud Rate, and Interrupt Settings
 95 @---------------------------------------------------------------------
 96 @ disable the UART to access DLL and DLH
 97 LDR R0, =0x48024020     @ MDR1 register
 98 LDR R1, [R0]            @ read the register value
 99 MOV R2, #0x07           @ put the UART in disabled state
100 ORR R1, R1, R2          @ combine new value and existing register value
101 STR R1, [R0]            @ write the value to the register
102
103 @ switch to register configuration mode B
104 LDR R0, =0x4802400C     @ UART_LCR
```

```
105 LDR R1, [R0]            @ read the register value
106 MOV R2, #0xFF00         @ mask the upper half and zero the lower half
107 AND R1, R1, R2
108 MOV R2, #0x00BF         @ put the UART in disabled state
109 ORR R1, R1, R2          @ combine new value and existing register value
110 STR R1, [R0]            @ write the value to the register
111
112 @ enable access to the IER UART register
113 @ (save the current status of the enhanced functions bit)
114 LDR R0, =0x48024008     @ UART_EFR
115 LDR R1, [R0]            @ read the register value
116 MOV R2, #0x0010         @ enhanced functions bit = bit 4 (5th place)
117 AND R1, R1, R2
118 MOV R3, R1              @ move to R3 for safe keeping
119 @ (enable enhanced functions write in EFR register)
120 ORR R1, R1, R2          @ combine new value and existing register value
121 STR R1, [R0]            @ write the value to the register
122
123 @ switch to register operational mode
124 LDR R0, =0x4802400C     @ UART_LCR
125 MOV R1, #0x0000         @ write 0x00 to LCR
126 STR R1, [R0]            @ write the value to the register
127
128 @ clear the IER register to disable the UART interrupts
129 @ (disable sleep mode to access DLL and DLH registers
130 LDR R0, =0x48024004     @ UART_IER interrupt enable register
131 LDR R1, [R0]            @ read the register value
132 MOVW R2, #0xFFEF        @ sleep mode = bit 4
133 AND R1, R1, R2          @ clear the bit
134 STR R1, [R0]            @ write value to the register
135
136 @ switch to register configuration mode B
137 LDR R0, =0x4802400C     @ UART_LCR
138 LDR R1, [R0]            @ read the register value
139 MOV R2, #0xFF00         @ mask the upper half and zero the lower half
140 AND R1, R1, R2
141 MOV R2, #0x00BF         @ put the UART in disabled state
142 ORR R1, R1, R2          @ combine new value and existing register value
143 STR R1, [R0]            @ write the value to the register
144
145 @ load the baud rate values for DLL and DLH
146 LDR R0, =0x48024000     @ UART_DLL
147 LDR R1, [R0]            @ read the register value
148 MOV R1, #0x004E
149 STR R1, [R0]            @ load the register value
150
151 LDR R0, =0x48024004     @ UART_DLH
152 LDR R1, [R0]            @ read the register value
153 MOV R1, #0x0000
154 STR R1, [R0]            @ load the register value
155
156 @ switch to register operational mode
157 LDR R0, =0x4802400C     @ UART_LCR
158 MOV R1, #0x0000         @ write 0x00 to LCR
```

```
159 STR R1, [R0]            @ write the value to the register
160
161 @ load the new interrupt configuration
162 LDR R0, =0x48024004      @ UART_IER
163 LDR R1, [R0]            @ load the register value
164 MOV R2, #0x0082          @ enable CTS and THR interrupts
165 ORR R1, R1, R2           @ combine new value and existing register value
166 STR R1, [R0]            @ write the value to the register
167
168 @ switch to configuration mode B
169 LDR R0, =0x4802400C      @ UART_LCR
170 LDR R1, [R0]            @ read the register value
171 MOV R2, #0xFF00          @ mask the upper half and zero the lower half
172 AND R1, R1, R2
173 MOV R2, #0x00BF          @ put the UART in disabled state
174 ORR R1, R1, R2           @ combine new value and existing register value
175 STR R1, [R0]            @ write the value to the register
176
177 @ restore the EFR ENHANCED_EN bit (turns off enhanced feature mode)
178 LDR R0, =0x48024008      @ UART_EFR
179 LDR R1, [R0]            @ read the register value
180 ADD R1, R1, R3           @ restore the EN bit
181 STR R1, [R0]            @ write the value to the register
182
183 @ configure the UART protocol
184 @ switch to operational mode (set DIV_EN and BREAK_EN to 0)
185 LDR R0, =0x4802400C      @ UART_LCR
186 LDR R1, [R0]            @ read the register value
187 MOVW R2, #0xFF03         @ configure LCR register
188 AND R1, R1, R2
189 STR R1, [R0]            @ write the value to the register
190
191 @ set the new UART mode
192 LDR R0, =0x48024020      @ UART_MDR1
193 LDR R1, [R0]            @ read the register value
194 MOVW R2, #0xFF00         @ MDR[0:2] = 0x00 - enable 16-bit UART mode
195 AND R1, R1, R2           @ combine new value and existing value
196 STR R1, [R0]
197
198
199 @ Initialize the RC8660
200 @--------------------------------------------------------------------
201 @ set the baudrate
202 LDR     R0, =0x48024000         @ load the address for THR
203 MOV     R1, #0x0D
204 STRB    R1, [R0]                @ load the characters into THR
205
206
207 @ Initialize the GPIO
208 @--------------------------------------------------------------------
209 @ initialize GPIO pin for falling edge detect
210 LDR R0, =0x4804C14C      @ GPIO1_FALLING_DETECT
211 LDR R1, [R0]            @ read the register value
212 MOV R2, #BUTTON_PIN      @ word to program the register
```

```
213 ORR R1, R1, R2          @ modify the register value
214 STR R1, [R0]            @ write the value to the register
215
216 @ initialize GPIO pins for external interrupt
217 LDR R0, =0x4804C034     @ GPIO1_IRQSTATUS_SET0
218 LDR R1, [R0]            @ read the register value
219 MOV R2, #BUTTON_PIN     @ load the word to program the register
220 ORR R1, R1, R2          @ modify the current value with new value
221 STR R1, [R0]            @ write new value to the register
222
223 @ configure the UART TXD pin
224 LDR R0, =0x44E10954     @ CONTROL_MODULE.SPI0_D0 (B17)
225 LDR R1, [R0]            @ load value of register
226 MOVW R2, #0xFFF8           @ load mask for upper bits
227 AND R1, R1, R2          @ clear the mode bits [0:2]
228 MOV R2, #0x01           @ load new value for mode bits
229 ORR R1, R1, R2          @ make new register value
230 STR R1, [R0]
231
232 @ configure the UART CTS pin
233 LDR R0, =0x44E108C0     @ CONTROL_MODULE.LCD_DATA8
234 LDR R1, [R0]            @ load value of the register
235 MOVW R2, #0xFFF8           @ load mask for the upper bits
236 AND R1, R1, R2          @ clear the mode bits [0:2]
237 MOV R2, #0x06           @ load the value for mode bits
238 ORR R1, R1, R2          @ make new register value
239 STR R1, [R0]
240
241 @ Initialize the timer
242 @-----------------------------------------------------------------------
243 @ enable auto-reload for the timer overflow
244 LDR R0, =0x48042038     @ load the address for TIM 3 CONTROL
245 LDR R1, [R0]            @ read the register value
246 MOV R2, #0x02           @ enable auto-reload
247 ORR R1, R1, R2          @ modify the current value with new value
248 STR R1, [R0]            @ write new value to the register
249
250 @ set the counter value
251 LDR R0, =0x48042040     @ TIM3 Load Register
252 LDR R1, =TIM_COUNT_VAL  @ load the counter value
253 STR R1, [R0]            @ write new value to register
254
255 LDR R0, =0x4804203C     @ TIM3 Counter Register
256 LDR R1, =TIM_COUNT_VAL
257 STR R1, [R0]
258
259 @ enable timer IRQs for overflow
260 LDR R0, =0x4804202C     @ load the address for TIM3_IRQ_SET
261 LDR R1, [R0]            @ read the register value
262 MOV R2, #0x02           @ load the word to program the register
263 ORR R1, R1, R2          @ modify the current value with new value
264 STR R1, [R0]            @ write new value to the register
265
266
```

```
267 @ Initialize the interrupt controller
268 @---------------------------------------------------------------------
269 @ initialize interrupt controller for GPIO pins
270 LDR R0, =0x482000E8     @ INTC_MIR_CLEAR3
271 LDR R1, [R0]            @ read the register value
272 MOV R2, #0x00000004     @ interrupt 98 = bit 3 in MIR3
273 ORR R1, R1, R2          @ unmask GPIO interrupt
274 STR R1, [R0]            @ write new value to the register
275
276 @ initialize interrupt controller for timer
277 LDR R0, =0x482000C8     @ INTC_MIR_CLEAR2
278 LDR R1, [R0]
279 MOV R2, #0x00000020     @ set bit 5 for interrupt 69
280 ORR R1, R1, R2          @ unmask Timer interrupt
281 STR R1, [R0]            @ write new value to the register
282
283 @ Initialize interrupts in the processor
284 @---------------------------------------------------------------------
285 MRS R3, CPSR    @ copy CPSR to R3
286 BIC R3, #0x80   @ clear bit 7
287 MSR CPSR_c, R3  @ write back to CPSR
288
289
290 MAIN_LOOP:
291     NOP
292     NOP
293     B MAIN_LOOP
294     B END
295
296 @---------------------------------------------------------------------
297 @ INTERRUPT SERVICE ROUTINE:
298 @ This procedure is hooked into the interrupt vector table to supercede
299 @ processor IRQ requests. It checks if the button was the source of the
300 @ interrupt and runs a specific procedure if it is the source of the
    interrupt
301 INT_DIRECTOR:
302     STMFD SP!, {R0-R3, LR}  @ push registers on to the stack
303
304 @ check if GPIO was the interrupt source
305     LDR R0, =0x482000F8 @ INTC_PENDING_IRQ3
306     LDR R1, [R0]        @ read the pending interrupt register
307     MOV R2, #0x00000004 @ check if GPIO triggered interrupt
308     AND R1, R1, R2      @ test if GPIO was interrupt source
309     CMP R1, #0x00
310     BEQ UART_CHECK      @ if zero, GPIO was not source of interrupt
311                         @ if nonzero, GPIO triggered the interrupt
312                         @ see if the button was the source
313     LDR R0, =0x4804C02C @ GPIO1_IRQSTATUS_0
314     LDR R1, [R0]        @ read value of register
315     MOV R2, #BUTTON_PIN @ load the value for the button pin
316     AND R1, R1, R2      @ make resultant word for comparison
317     CMP R1, #0x00       @ compare to see if result is nonzero.
318                         @ Nonzero = GPIO flag was set
319     BNE BUTTON_SVC      @ go to button service procedure
```

```
320     B PASS_ON           @ otherwise different GPIO caused interrupt
321
322     UART_CHECK:             @ check if the UART caused the interrupt
323     LDR R0, =0x482000D8     @ INTC_PENDING_IRQ2
324     LDR R1, [R0]            @ read the pending interrupt register
325     MOV R2, #0x00000400     @ check if UART triggered interrupt
326     AND R1, R1, R2          @ test if UART was interrupt source
327     CMP R1, #0x00           @ if nonzero, timer was source
328     BNE UART2_INT_SVC       @ otherwise timer was not the source
329
330     TIMER_CHECK:
331     LDR R0, =0x482000D8     @ INTC_PENDING_IRQ2
332     LDR R1, [R0]            @ read the pending interrupt register
333     MOV R2, #0x00000020     @ check if GPIO triggered interrupt
334     AND R1, R1, R2          @ test if GPIO was interrupt source
335     CMP R1, #0x00           @ if nonzero, timer was source
336     BNE TIMER_SVC_RTN       @ otherwise timer was not the source
337
338     @ exit int director ISR
339     PASS_ON:
340     @ re-enable IRQ interrupts in the processor
341     MRS R3, CPSR            @ copy CPSR to R3
342     BIC R3, #0x80           @ clear bit 7
343     MSR CPSR_c, R3          @ write back to CPSR
344
345     LDMFD SP!, {R0-R3, LR}  @ restore register states
346     SUBS PC, LR, #4         @ return service to the system IRQ
347
348
349 @---------------------------------------------------------------------
350 @ TIMER SERVICE ROUTINE
351 @ Handles the LED flashing sequence
352 TIMER_SVC_RTN:
353 @ initialize interrupt controller for UART to begin sending the message
354 LDR R0, =0x482000C8     @ INTC_MIR_CLEAR2
355 LDR R1, [R0]            @ read the register value
356 MOV R2, #0x00000400     @ interrupt 74 = bit 10 on MIR2
357 ORR R1, R1, R2          @ unmask the UART interrupt
358 STR R1, [R0]            @ write new value to the register
359
360 @ reset the Timer IRQ flags
361 LDR R0, =0x48042028         @ TIM3 IRQ Status Register
362 MOV R1, #0x2                @ mask to clear bit 2
363 STR R1, [R0]
364
365 @ reset the interrupt controller IRQ flag
366 LDR R0, =0x48200048         @ INTC_CONTROL
367 MOV R1, #0x1                @ value to reset IRQ generation
368 STR R1, [R0]
369
370 @ re-enable IRQ interrupts in the processor
371 MRS R3, CPSR                @ copy CPSR to R3
372 BIC R3, #0x80               @ clear bit 7
373 MSR CPSR_c, R3              @ write back to CPSR
```

```
374
375 LDMFD SP!, {R0-R3, LR}      @ restore register states
376 SUBS PC, LR, #4             @ return service to main
377
378
379
380 @---------------------------------------------------------------------
381 @ BUTTON SERVICE PROCEDURE
382 @ This procedure handles the specific button service requirements
383 @ IRQ #98
384 BUTTON_SVC:
385 @ push SPSR on stack
386     MRS R3, SPSR             @ copy the saved program status register
387     STMFD R13!, {R3}         @ push on to stack
388
389 @ mask lower priority interrupts
390 @ already masked
391
392 @ set the timer counter value
393     LDR R0, =0x48042040      @ TIM3 Load Register
394     LDR R1, =TIM_COUNT_VAL   @ load the counter value
395     STR R1, [R0]             @ write new value to register
396
397     LDR R0, =0x4804203C      @ TIM3 Counter Register
398     LDR R1, =TIM_COUNT_VAL
399     STR R1, [R0]
400
401 @ enable/disable the timer
402     LDR R0, =0x48042038      @ TIM3 Control Register
403     LDR R1, [R0]             @ read the register value
404     AND R2, R1, #0x00000001 @ mask all but the start bit
405     MOV R3, #0x00            @ load test value
406     CMP R2, R3               @ test to see if timer is off
407     MOVEQ R3, #0x03          @ put new value in R3
408     MOVNE R3, #0x02
409     STR R3, [R0]             @ update the register
410
411 @ update the ON/OFF status
412     LDR R0, =STATUS
413     LDR R1, [R0]
414     CMP R1, #0x01            @ Test
415     BNE TURN_UART_ON
416
417 @ turn the status to OFF
418     LDR R0, =STATUS
419     MOV R1, #0x00
420     STR R1, [R0]
421
422 @ disable interrupt controller for UART
423     LDR R0, =0x482000CC      @ INCT_MIR_SET2
424     LDR R1, [R0]
425     MOV R2, #0x00000400      @ interrupt 74 = bit 10 on MIR2
426     ORR R1, R1, R2           @ mask the UART interrupt
427     STR R1, [R0]             @ write new value to the register
```

```
428     B TURN_UART_OFF
429
430 TURN_UART_ON:
431 @ turn the status to ON
432     LDR R0, =STATUS
433     MOV R1, #0x01
434     STR R1, [R0]
435 @ enable interrupt controller for UART
436     LDR R0, =0x482000C8     @ INTC_MIR_CLEAR2
437     LDR R1, [R0]            @ read the register value
438     MOV R2, #0x00000400     @ interrupt 74 = bit 10 on MIR2
439     ORR R1, R1, R2          @ unmask the UART interrupt
440     STR R1, [R0]            @ write new value to the register
441
442 TURN_UART_OFF:
443 @ reset the GPIO interrupt request
444     LDR R0, =0x4804C02C
445     MOV R2, #BUTTON_PIN
446     STR R2, [R0]
447
448 @ reset interrupt controller IRQ requests
449     LDR R0, =0x48200048     @ INTC_CONTROL
450     MOV R1, #0x01           @ value to reset IRQ generation
451     STR R1, [R0]
452
453 @ unmask lower priority interrupts
454 @ not necessary
455
456 @ restore SPSR
457     LDMFD R13!, {R3}            @ get saved SPSR
458     MSR SPSR_cf, R3             @ restore the SPSR
459
460     LDMFD SP!, {R0-R3, LR}      @ restore register states
461     SUBS PC, LR, #4             @ return service to the system IRQ
462
463 @---------------------------------------------------------------------
464 @ UART2 SERVICE PROCEDURE
465 @ This procedure handles the UART interrupt logic
466 # 74
467 UART2_INT_SVC:
468 @ push SPSR on stack
469     MRS R3, SPSR            @ copy the saved program status register
470     STMFD R13!, {R3}        @ push on to stack
471
472 @ check if CTS and not THR was source of interrupt
473     LDR R0, =0x48024008     @ interrupt identification register
474     LDR R1, [R0]            @ load the state of the register
475     MOVW R2, #0x0020        @ check the CTS bit
476     AND R1, R1, R2          @ CTS = 0x10 at IIR[5:1]
477     CMP R1, #0x0020
478     BEQ UART_INT_CLOSE      @ if CTS but not THR -> exit
479                             @ else check the THR bit
480 @ check THR bit
481     LDR R0, =0x48024008     @ interrupt identification register
```

```
482     LDR R1, [R0]              @ load the state of the register
483     MOVW R2, 0x0002           @ check the THR bit
484     AND R1, R1, R2            @ CTS = 0x01 at IIR [5:1]
485     CMP R1, #0x0002
486     BNE UART_MASK_THR         @ if not CTS but was THR
487                               @ -> mask THR and exit
488     BL  UART_TRANSMIT         @ else transmit next character
489     B   UART_INT_CLOSE        @ come back from transmission,
490                               @ exit the procedure
491
492 UART_MASK_THR:
493 @ mask the THR interrupt
494     LDR R0, =0x48024004       @ UART_IER
495     LDR R1, [R0]              @ load the register value
496     MOVW R2, #0xFFFD          @ disable the THR interrupt
497     AND R1, R1, R2            @ combine new and existing
498     STR R1, [R0]              @ write the value to the register
499
500 UART_INT_CLOSE:
501 @ reset interrupt controller IRQ requests
502     LDR R0, =0x48200048       @ INTC_CONTROL
503     MOV R1, #0x01             @ value to reset IRQ generation
504     STR R1, [R0]
505
506 @ re-enable IRQ interrupts in the processor
507     MRS R3, CPSR    @ copy CPSR to R3
508     BIC R3, #0x80   @ clear bit 7
509     MSR CPSR_c, R3  @ write back to CPSR
510
511     LDMFD R13!, {R3}
512     MSR SPSR_cf, R3
513
514     LDMFD SP!, {R0-R3, LR}  @ restore register states
515     SUBS PC, LR, #4         @ return service to the system IRQ
516
517
518 @-------------------------------------------------------------------
519 @ UART TRANSMIT PROCEDURE
520 @ This procedure handles the UART transmission process
521 UART_TRANSMIT:
522     STMFD   R13!, {R0-R3, R14}      @ push registers on to the stack
523
524 @ load the character counter
525     LDR R0, =CHAR_INDEX       @ load the address of the character count
526                               @ index
527     LDR R1, [R0]              @ get the value of the counter
528
529     LDR R2, =MESSAGE          @ load the base of the array
530     ADD R2, R2, R1            @ add the array pointer and counter
531                               @ (acts as an offset)
532     LDRB R3, [R2]             @ get the value at that location
533
534     LDR R2, =0x48024000       @ load the address for THR
535     STRB R3, [R2]             @ load the characters into THR
```

```
536
537     LDR R0, =CHAR_INDEX     @ load the index address
538     LDR R1, [R0]            @ get the index value
539     MOV R2, #0x01           @ value to increment
540     ADD R1, R1, R2          @ increment index by 1
541     STR R1, [R0]            @ update the memory location
542     CMP R1, #MAXCHAR
543     BNE UART_TRANSMIT_CLOSE     @ exit the UART
544                                 @ Otherwise, end of the transmission
545 @ reset the char index counter
546     MOV R1, #0x00           @ reset the char index counter
547     STR R1, [R0]            @ write to the location for the index
548
549 @ disable the UART interrupts in interrupt controller
550     LDR R0, =0x482000CC     @ INCT_MIR_SET2
551     LDR R1, [R0]
552     MOV R2, #0x00000400     @ interrupt 74 = bit 10 on MIR2
553     ORR R1, R1, R2          @ unmask the UART interrupt
554     STR R1, [R0]            @ write new value to the register
555
556
557 UART_TRANSMIT_CLOSE:
558     LDMFD R13!, {R0-R3, PC}     @ return to caller
559
560
561 @-------------------------------------------------------------------
562 @ DELAY ROUTINE
563 @ Necessary to give some buffer after we turn on the peripheral clocks
564 @ before we start accessing registers
565 DELAY:
566 STMFD R13!, {R4, R14}   @ save the register states and
567                         @ link register location
568 LDR R4, =0x0022DCD5
569 D_LOOP:
570     NOP
571     SUBS R4, #1
572     BNE D_LOOP
573 LDMFD R13!, {R4, PC}
574
575 END:
576 .END
```