```
                .section .vectors, "ax"
                B       _start              // reset vector
                B       SERVICE_UND         // undefined instruction vector
                B       SERVICE_SVC         // software interrupt vector
                B       SERVICE_ABT_INST    // aborted prefetch vector
                B       SERVICE_ABT_DATA    // aborted data vector
                .word   0                   // unused vector
                B       SERVICE_IRQ         // IRQ interrupt vector
                B       SERVICE_FIQ         // FIQ interrupt vector
                .text
                .global _start
/* Set up stack pointers for IRQ and SVC processor modes */
_start:         MOV     R1, #0b11010010     // interrupts masked, MODE = IRQ
                MSR     CPSR_c, R1          // change to IRQ mode
                LDR     SP, =0xFFFFFFFF - 3 // set IRQ stack to A9 onchip memory

                MOV     R1, #0b11010011     // interrupts masked, MODE = SVC
                MSR     CPSR, R1            // change to supervisor mode
                LDR     SP, =0x3FFFFFFF - 3 // set SVC stack to top of DDR3 memory

                BL      CONFIG_GIC          // configure the ARM generic interrupt
                controller

/* Configure the KEY pushbuttons port to generate interrupts */
                LDR     R0, =0xFF200050     // KEY address
                MOV     R1, #0xF            // set interrupt mask bits
                STR     R1, [R0, #0x8]      // interrupt mask register (base + 8)

/* Enable IRQ interrupts in the ARM processor */
                MOV      R0, #0b01010011    // IRQ unmasked, MODE = SVC
                MSR      CPSR_c, R0

IDLE:           B       IDLE                // main program simply idles

/* Define the exception service routines */
SERVICE_IRQ:    PUSH    {R0-R7, LR}
                LDR     R4, =0xFFFEC100     // GIC CPU interface base address
                LDR     R5, [R4, #0x0C]     // read the ICCIAR in the CPU interface

FPGA_IRQ1_HANDLER:
                CMP     R5, #73             // check the interrupt ID

UNEXPECTED:     BNE     UNEXPECTED          // if not recognized, stop here
                BL      KEY_ISR

EXIT_IRQ:       STR     R5, [R4, #0x10]     // write to the End of Interrupt Register
(ICCEOIR)
                POP     {R0-R7, LR}
                SUBS    PC, LR, #4          // return from exception

/* Check which key has been pressed and writes accordingly */
KEY_ISR:        LDR     R0, =0xFF200050     // base address of pushbutton KEY port
                LDR     R1, [R0, #0xC]      // read edge capture register
                MOV     R2, #0xF
                STR     R2, [R0, #0xC]      // clear the interrupt
                LDR     R0, =0xFF200020     // based address of HEX display

CHECK_KEY0:     MOV     R3, #0b0001
                CMP     R3, R1              // Check for KEY0
                BNE     CHECK_KEY1
                MOV     R2, #0b00111111     // '0'
                LDRB    R3, [R0]            // HEX0
                CMP     R2, R3
                BEQ     CLEAR_HEX0          // Check is HEX0 is already '0'
                STRB    R2, [R0]            // Display '0'
                B       END_KEY_ISR
CLEAR_HEX0:     MOV     R2, #0
                STRB    R2, [R0]            // Display blank
                B       END_KEY_ISR
```

```
68
69  CHECK_KEY1:     MOV     R3, #0b0010
70                  CMP     R3, R1              // Check for KEY1
71                  BNE     CHECK_KEY2
72                  MOV     R2, #0b00000110     // '1'
73                  LDRB    R3, [R0, #1]        // HEX1
74                  CMP     R2, R3
75                  BEQ     CLEAR_HEX1          // Check is HEX1 is already '1'
76                  STRB    R2, [R0, #1]        // Display '1'
77                  B       END_KEY_ISR
78  CLEAR_HEX1:     MOV     R2, #0
79                  STRB    R2, [R0, #1]        // Display blank
80                  B       END_KEY_ISR
81
82  CHECK_KEY2:     MOV     R3, #0b0100
83                  CMP     R3, R1              // Check for KEY2
84                  BNE     IS_KEY3
85                  MOV     R2, #0b01011011     // '2'
86                  LDRB    R3, [R0, #2]        // HEX2
87                  CMP     R2, R3
88                  BEQ     CLEAR_HEX2          // Check is HEX2 is already '2'
89                  STRB    R2, [R0, #2]        // Display '2'
90                  B       END_KEY_ISR
91  CLEAR_HEX2:     MOV     R2, #0
92                  STRB    R2, [R0, #2]        // Display blank
93                  B       END_KEY_ISR
94
95  IS_KEY3:        MOV     R2, #0b01001111     // '3'
96                  LDRB    R3, [R0, #3]        // HEX3
97                  CMP     R2, R3
98                  BEQ     CLEAR_HEX3          // Check is HEX3 is already '3'
99                  STRB    R2, [R0, #3]        // Display '3'
100                 B       END_KEY_ISR
101 CLEAR_HEX3:     MOV     R2, #0
102                 STRB    R2, [R0, #3]        // Display blank
103                 B       END_KEY_ISR
104
105 END_KEY_ISR:    BX      LR                  // Return
106
107                 .end
108
```

```
1                   .section .vectors, "ax"
2                   B       _start              // reset vector
3                   B       SERVICE_UND         // undefined instruction vector
4                   B       SERVICE_SVC         // software interrupt vector
5                   B       SERVICE_ABT_INST    // aborted prefetch vector
6                   B       SERVICE_ABT_DATA    // aborted data vector
7                   .word   0                   // unused vector
8                   B       SERVICE_IRQ         // IRQ interrupt vector
9                   B       SERVICE_FIQ         // FIQ interrupt vector
10                  .text
11                  .global _start
12  _start:
13  /* Set up stack pointers for IRQ and SVC processor modes */
14                  MOV     R1, #0b11010010     // interrupts masked, MODE = IRQ
15                  MSR     CPSR_c, R1          // change to IRQ mode
16                  LDR     SP, =0xFFFFFFFF - 3 // set IRQ stack to A9 onchip memory
17
18                  MOV     R1, #0b11010011     // interrupts masked, MODE = SVC
19                  MSR     CPSR, R1            // change to supervisor mode
20                  LDR     SP, =0x3FFFFFFF - 3 // set SVC stack to top of DDR3 memory
21
22                  BL      CONFIG_GIC          // configure the ARM generic interrupt
                    controller
23                  BL      CONFIG_TIMER        // configure the Interval Timer
24                  BL      CONFIG_KEYS         // configure the pushbutton KEYs port
25
26  /* Enable IRQ interrupts in the ARM processor */
27                  MOV      R0, #0b01010011    // IRQ unmasked, MODE = SVC
28                  MSR      CPSR_c, R0
29
30                  LDR     R5, =0xFF200000     // LEDR base address
31  LOOP:
32                  LDR     R3, COUNT           // global variable
33                  STR     R3, [R5]            // write to the LEDR lights
34                  B       LOOP
35
36  /* Configure the Interval Timer to create interrupts at 0.25 second intervals */
37  CONFIG_TIMER:   LDR     R0, =0xFF202000     // FPGA timer base address
38                  LDR     R1, =30784
39                  STR     R1, [R0, #8]        // Set lower bits of timer
40                  LDR     R1, =381
41                  STR     R1, [R0, #12]       // Set upper bits of timer
42                  MOV     R1, #0b0111         // Control register bits
43                  STR     R1, [R0, #4]        // Start timer, set auto-reload and enable
                    interrupts
44                  BX      LR
45
46  /* Configure the pushbutton KEYS to generate interrupts */
47  CONFIG_KEYS:
48                  LDR     R0, =0xFF200050     // KEY address
49                  MOV     R1, #0xF            // set interrupt mask bits
50                  STR     R1, [R0, #0x8]      // interrupt mask register (base + 8)
51                  BX      LR
52
53  /* Define the exception service routines */
54  SERVICE_IRQ:    PUSH    {R0-R7, LR}
55                  LDR     R4, =0xFFFEC100     // GIC CPU interface base address
56                  LDR     R5, [R4, #0x0C]     // read the ICCIAR in the CPU interface
57
58  FPGA_IRQ1_HANDLER:
59                  CMP     R5, #73             // check the interrupt ID
60                  BEQ     KEY_INTERRUPT
61                  CMP     R5, #72
62                  BEQ     CLK_INTERRUPT
63
64  UNEXPECTED:     B       UNEXPECTED          // if not recognized, stop here
65  KEY_INTERRUPT:  BL      KEY_ISR
66  CLK_INTERRUPT:  BL      TIMER_ISR
67
```

```
EXIT_IRQ:       STR     R5, [R4, #0x10]     // write to the End of Interrupt Register
(ICCEOIR)
                POP     {R0-R7, LR}
                SUBS    PC, LR, #4          // return from exception

/* Check if it has been 0.25 seconds and adds RUN to COUNT */
TIMER_ISR:      LDR     R0, =0xFF202000     // base address of FPGA timer
                LDR     R1, [R0]        // read edge capture register
                MOV     R2, #0
                STR     R2, [R0]            // clear the interrupt
                LDR     R0, =RUN            // Load RUN toggle
                LDR     R0, [R0]
                LDR     R1, =COUNT          // Load counter
                LDR     R2, [R1]
                ADD     R2, R0              // Increment counter by RUN
                STR     R2, [R1]            // Store incremented counter
END_TIMER_ISR:  BX      LR                  // Return

/* Check if a key has been pressed and toggles RUN */
KEY_ISR:        LDR     R0, =0xFF200050     // base address of pushbutton KEY port
                LDR     R1, [R0, #0xC]      // read edge capture register
                MOV     R2, #0xF
                STR     R2, [R0, #0xC]      // clear the interrupt
                LDR     R0, =RUN            // Address of RUN toggle

CHECK_KEYS:     MOV     R3, #0b1111
                ORRS    R3, R1              // Check for any KEY
                BEQ     END_KEY_ISR
                LDR     R1, [R0]            // Get RUN from memory
                EOR     R1, #1              // Toggle RUN
                STR     R1, [R0]            // Set RUN in memory

END_KEY_ISR:    BX      LR                  // Return

/* Global variables */
                .global COUNT
COUNT:          .word   0x0             // used by timer
                .global RUN                 // used by pushbutton KEYs
RUN:            .word   0x1                 // initial value to increment COUNT

                .end
```

```
1                   .section .vectors, "ax"
2                   B       _start              // reset vector
3                   B       SERVICE_UND         // undefined instruction vector
4                   B       SERVICE_SVC         // software interrupt vector
5                   B       SERVICE_ABT_INST    // aborted prefetch vector
6                   B       SERVICE_ABT_DATA    // aborted data vector
7                   .word   0                   // unused vector
8                   B       SERVICE_IRQ         // IRQ interrupt vector
9                   B       SERVICE_FIQ         // FIQ interrupt vector
10                  .text
11                  .global _start
12  _start:
13  /* Set up stack pointers for IRQ and SVC processor modes */
14                  MOV     R1, #0b11010010     // interrupts masked, MODE = IRQ
15                  MSR     CPSR_c, R1          // change to IRQ mode
16                  LDR     SP, =0xFFFFFFFF - 3 // set IRQ stack to A9 onchip memory
17
18                  MOV     R1, #0b11010011     // interrupts masked, MODE = SVC
19                  MSR     CPSR, R1            // change to supervisor mode
20                  LDR     SP, =0x3FFFFFFF - 3 // set SVC stack to top of DDR3 memory
21
22                  BL      CONFIG_GIC          // configure the ARM generic interrupt
                    controller
23                  BL      CONFIG_TIMER        // configure the Interval Timer
24                  BL      CONFIG_KEYS         // configure the pushbutton KEYs port
25
26  /* Enable IRQ interrupts in the ARM processor */
27                  MOV      R0, #0b01010011    // IRQ unmasked, MODE = SVC
28                  MSR      CPSR_c, R0
29
30                  LDR     R5, =0xFF200000     // LEDR base address
31  LOOP:
32                  LDR     R3, COUNT           // global variable
33                  STR     R3, [R5]            // write to the LEDR lights
34                  B       LOOP
35
36  /* Configure the Interval Timer to create interrupts at 0.25 second intervals */
37  CONFIG_TIMER:   LDR     R0, =0xFF202000     // FPGA timer base address
38                  LDR     R1, RATE
39                  STR     R1, [R0, #8]        // Set lower bits of timer
40                  LSR     R1, #16
41                  STR     R1, [R0, #12]       // Set upper bits of timer
42                  MOV     R1, #0b0111         // Control register bits
43                  STR     R1, [R0, #4]        // Start timer, set auto-reload and enable
                    interrupts
44                  BX      LR
45
46  /* Configure the pushbutton KEYS to generate interrupts */
47  CONFIG_KEYS:
48                  LDR     R0, =0xFF200050     // KEY address
49                  MOV     R1, #0xF            // set interrupt mask bits
50                  STR     R1, [R0, #0x8]      // interrupt mask register (base + 8)
51                  BX      LR
52
53  /* Define the exception service routines */
54  SERVICE_IRQ:    PUSH    {R0-R7, LR}
55                  LDR     R4, =0xFFFEC100     // GIC CPU interface base address
56                  LDR     R5, [R4, #0x0C]     // read the ICCIAR in the CPU interface
57
58  FPGA_IRQ1_HANDLER:
59                  CMP     R5, #73             // check the interrupt ID
60                  BEQ     KEY_INTERRUPT
61                  CMP     R5, #72
62                  BEQ     CLK_INTERRUPT
63
64  UNEXPECTED:     B       UNEXPECTED          // if not recognized, stop here
65  KEY_INTERRUPT:  BL      KEY_ISR
66  CLK_INTERRUPT:  BL      TIMER_ISR
67
```

```
68  EXIT_IRQ:       STR     R5, [R4, #0x10]     // write to the End of Interrupt Register
    (ICCEOIR)
69                  POP     {R0-R7, LR}
70                  SUBS    PC, LR, #4          // return from exception
71
72  /* Check if FPGA timer generated an interrupt and adds RUN to COUNT */
73  TIMER_ISR:      LDR     R0, =0xFF202000     // base address of FPGA timer
74                  MOV     R2, #0
75                  STR     R2, [R0]            // clear the interrupt
76                  LDR     R0, RUN             // Load RUN toggle
77                  LDR     R1, COUNT           // Load counter
78                  ADD     R1, R0              // Increment counter by RUN
79                  STR     R1, COUNT           // Store incremented counter
80  END_TIMER_ISR:  BX      LR                  // Return
81
82  /* Check if a key has been pressed and toggles RUN */
83  KEY_ISR:        LDR     R0, =0xFF200050     // base address of pushbutton KEY port
84                  LDR     R1, [R0, #0xC]      // read edge capture register
85                  MOV     R2, #0xF
86                  STR     R2, [R0, #0xC]      // clear the interrupt
87
88  CHECK_KEY0:     MOV     R3, #0b0001
89                  CMP     R3, R1              // Check for KEY0
90                  BNE     CHECK_KEY1
91                  LDR     R0, RUN             // LOAD RUN toggle
92                  EOR     R0, #1              // Toggle RUN
93                  STR     R0, RUN             // Set RUN in memory
94                  B       END_KEY_ISR
95
96  CHECK_KEY1:     MOV     R3, #0b0010
97                  CMP     R3, R1              // Check for KEY1
98                  BNE     CHECK_KEY2
99                  LDR     R0, =0xFF202000     // Address of FPGA timer
100                 MOV     R1, #0b1000
101                 STR     R1, [R0, #4]        // Stop timer
102                 LDR     R1, RATE
103                 LSR     R1, #1              // Double the rate (HALF the timeout)
104                 STR     R1, RATE            // Store the rate
105                 B       SET_TIMER
106
107 CHECK_KEY2: MOV     R3, #0b0100
108                 CMP     R3, R1              // Check for KEY2
109                 BNE     END_KEY_ISR
110                 LDR     R0, =0xFF202000     // Address of FPGA timer
111                 MOV     R1, #0b1000
112                 STR     R1, [R0, #4]        // Stop timer
113                 LDR     R1, RATE
114                 LSL     R1, #1              // Half the rate (DOUBLE the timeout)
115                 STR     R1, RATE            // Store the rate
116
117 SET_TIMER:      STR     R1, [R0, #8]        // Set lower bits of timer
118                 LSR     R1, #16
119                 STR     R1, [R0, #12]       // Set upper bits of timer
120                 MOV     R1, #0b0111         // Control register bits
121                 STR     R1, [R0, #4]        // Start timer, set auto-reload and enable
                    interrupts
122
123 END_KEY_ISR:    BX      LR                  // Return
124
125 /* Global variables */
126                 .global COUNT
127 COUNT:          .word   0x0                 // used by timer
128                 .global RUN                 // used by pushbutton KEYs
129 RUN:            .word   0x1                 // initial value to increment COUNT
130 RATE:           .word   25000000
131
132                 .end
133
```

```
                .section .vectors, "ax"
                B       _start              // reset vector
                B       SERVICE_UND         // undefined instruction vector
                B       SERVICE_SVC         // software interrupt vector
                B       SERVICE_ABT_INST    // aborted prefetch vector
                B       SERVICE_ABT_DATA    // aborted data vector
                .word   0                   // unused vector
                B       SERVICE_IRQ         // IRQ interrupt vector
                B       SERVICE_FIQ         // FIQ interrupt vector
                .text
                .global _start
_start:
/* Set up stack pointers for IRQ and SVC processor modes */
                MOV     R1, #0b11010010     // interrupts masked, MODE = IRQ
                MSR     CPSR_c, R1          // change to IRQ mode
                LDR     SP, =0xFFFFFFFF - 3 // set IRQ stack to A9 onchip memory

                MOV     R1, #0b11010011     // interrupts masked, MODE = SVC
                MSR     CPSR, R1            // change to supervisor mode
                LDR     SP, =0x3FFFFFFF - 3 // set SVC stack to top of DDR3 memory

                BL      CONFIG_GIC          // configure the ARM generic interrupt
                controller
                BL      CONFIG_PRIV_TIMER   // configure the private timer
                BL      CONFIG_TIMER        // configure the Interval Timer
                BL      CONFIG_KEYS         // configure the pushbutton KEYs port

/* Enable IRQ interrupts in the ARM processor */
                MOV      R0, #0b01010011    // IRQ unmasked, MODE = SVC
                MSR      CPSR_c, R0

                LDR     R5, =0xFF200000     // LEDR base address
                LDR     R6, =0xFF200020     // HEX3-0 base address
LOOP:
                LDR     R4, COUNT           // global variable
                STR     R4, [R5]            // light up the red lights
                LDR     R4, HEX_CODE        // global variable
                STR     R4, [R6]            // show the time in format SS:DD

                B       LOOP                // Endlessly loop

/* Configure the MPCore private timer to create interrupts every 1/100 seconds */
CONFIG_PRIV_TIMER:
                LDR     R0, =0xFFFEC600     // A9 private timer address
                LDR     R1, =2000000        // 0.01 seconds on 200MHz clock
                STR     R1, [R0]
                MOV     R1, #0b111          // Start timer and set it to auto-reload
                STR     R1, [R0, #0x8]
                BX      LR

/* Configure the Interval Timer to create interrupts at 0.25 second intervals */
CONFIG_TIMER:   LDR     R0, =0xFF202000     // FPGA timer base address
                LDR     R1, RATE
                STR     R1, [R0, #8]        // Set lower bits of timer
                LSR     R1, #16
                STR     R1, [R0, #12]       // Set upper bits of timer
                MOV     R1, #0b0111         // Control register bits
                STR     R1, [R0, #4]        // Start timer, set auto-reload and enable
                interrupts
                BX      LR

/* Configure the pushbutton KEYS to generate interrupts */
CONFIG_KEYS:
                LDR     R0, =0xFF200050     // KEY address
                MOV     R1, #0xF            // set interrupt mask bits
                STR     R1, [R0, #0x8]      // interrupt mask register (base + 8)
                BX      LR

/* Define the exception service routines */
```

```
68  SERVICE_IRQ:    PUSH    {R0-R7, LR}
69                  LDR     R4, =0xFFFEC100     // GIC CPU interface base address
70                  LDR     R5, [R4, #0x0C]     // read the ICCIAR in the CPU interface
71
72  FPGA_IRQ1_HANDLER:
73                  CMP     R5, #29             // check the interrupt ID
74                  BEQ     PRIV_INTERRUPT
75                  CMP     R5, #73
76                  BEQ     KEY_INTERRUPT
77                  CMP     R5, #72
78                  BEQ     CLK_INTERRUPT
79
80  UNEXPECTED:     B       UNEXPECTED          // if not recognized, stop here
81  PRIV_INTERRUPT: BL      PRIV_TIMER_ISR
82                  B       EXIT_IRQ
83  KEY_INTERRUPT:  BL      KEY_ISR
84                  B       EXIT_IRQ
85  CLK_INTERRUPT:  BL      TIMER_ISR
86                  B       EXIT_IRQ
87
88  EXIT_IRQ:       STR     R5, [R4, #0x10]     // write to the End of Interrupt Register
    (ICCEOIR)
89                  POP     {R0-R7, LR}
90                  SUBS    PC, LR, #4          // return from exception
91
92  /* Check if private timer generated an interrupt and increments time */
93  PRIV_TIMER_ISR: LDR     R0, =0xFFFEC600     // base address of private timer
94                  MOV     R1, #1
95                  STR     R1, [R0, #0xC]      // clear the interrupt
96                  LDR     R1, TIME            // Increment TIME
97                  ADD     R1, #1
98                  LDR     R2, =6000           // Load a literal
99                  CMP     R1, R2              // Wrap around to 0 when TIME > 5999
100                 BNE     SET_HEX
101                 MOV     R1, #0
102
103 SET_HEX:        PUSH    {LR}
104                 LDR     R3, =BIT_CODE
105                 STR     R1, TIME            // Stores TIME
106                 MOV     R0, R1              // Separate R1 into its digits
107                 BL      DIVIDE
108                 LDRB    R0, [R3, +R0]       // Get pattern for ones digit
109                 MOV     R2, R0
110
111                 MOV     R0, R1              // Get tens digit
112                 BL      DIVIDE
113                 LDRB    R0, [R3, +R0]       // Get pattern for tens digit
114                 LSL     R0, #8
115                 ORR     R2, R0
116
117                 MOV     R0, R1              // Get hundredth digit
118                 BL      DIVIDE              // Remainder from divide is thousandth digit
119                 LDRB    R0, [R3, +R0]       // Get pattern for hundreds digit
120                 LSL     R0, #16
121                 ORR     R2, R0
122                 LDRB    R1, [R3, +R1]       // Get pattern for thousandth digit
123                 LSL     R1, #24
124                 ORR     R2, R1
125
126                 STR     R2, HEX_CODE        // Set HEX_CODE for 7 segments
127
128 END_PRIV_TIMER_ISR:
129                 POP     {PC}                // Return
130
131 /* Check if FPGA timer generated an interrupt and adds RUN to COUNT */
132 TIMER_ISR:      LDR     R0, =0xFF202000     // base address of FPGA timer
133                 MOV     R2, #0
134                 STR     R2, [R0]            // clear the interrupt
135                 LDR     R0, RUN             // Load RUN toggle
```

```
136                 LDR     R1, COUNT           // Load counter
137                 ADD     R1, R0              // Increment counter by RUN
138                 STR     R1, COUNT           // Store incremented counter
139 END_TIMER_ISR:  BX      LR                  // Return
140
141 /* Check if a key has been pressed and toggles RUN */
142 KEY_ISR:        LDR     R0, =0xFF200050     // base address of pushbutton KEY port
143                 LDR     R1, [R0, #0xC]      // read edge capture register
144                 MOV     R2, #0xF
145                 STR     R2, [R0, #0xC]      // clear the interrupt
146
147 CHECK_KEY0:     MOV     R3, #0b0001
148                 CMP     R3, R1              // Check for KEY0
149                 BNE     CHECK_KEY1
150                 LDR     R0, RUN             // LOAD RUN toggle
151                 EOR     R0, #1              // Toggle RUN
152                 STR     R0, RUN             // Set RUN in memory
153                 B       END_KEY_ISR
154
155 CHECK_KEY1:     MOV     R3, #0b0010
156                 CMP     R3, R1              // Check for KEY1
157                 BNE     CHECK_KEY2
158                 LDR     R0, =0xFF202000     // Address of FPGA timer
159                 MOV     R1, #0b1000
160                 STR     R1, [R0, #4]        // Stop timer
161                 LDR     R1, RATE
162                 LSR     R1, #1              // Double the rate (HALF the timeout)
163                 CMP     R1, #0xFF               // Limit maximum rate
164                 BGT     SET_TIMER
165                 MOV     R1, #0xFF
166                 B       SET_TIMER
167
168 CHECK_KEY2:     MOV     R3, #0b0100
169                 CMP     R3, R1              // Check for KEY2
170                 BNE     END_KEY_ISR
171                 LDR     R0, =0xFF202000     // Address of FPGA timer
172                 MOV     R1, #0b1000
173                 STR     R1, [R0, #4]        // Stop timer
174                 LDR     R1, RATE
175                 LSL     R1, #1              // Half the rate (DOUBLE the timeout)
176                 CMP     R1, #0              // Limit minimum rate
177                 BNE     SET_TIMER
178                 MOV     R1, #0xA0000000
179
180 SET_TIMER:      STR     R1, RATE            // Store the rate
181                 STR     R1, [R0, #8]        // Set lower bits of timer
182                 LSR     R1, #16
183                 STR     R1, [R0, #12]       // Set upper bits of timer
184                 MOV     R1, #0b0111         // Control register bits
185                 STR     R1, [R0, #4]        // Start timer, set auto-reload and enable
                    interrupts
186
187 END_KEY_ISR:    BX      LR                  // Return
188
189 /* Subroutine to perform the integer division R0 / 10.
190  * Returns quotient in R1 and remainder in R0
191  */
192 DIVIDE:         PUSH    {R2,LR}
193                 MOV     R2, #0
194 CONT:           CMP     R0, #10
195                 BLT     DIV_END
196                 SUB     R0, #10
197                 ADD     R2, #1
198                 B       CONT
199 DIV_END:        MOV     R1, R2              // quotient in R1 (remainder in R0)
200                 POP     {R2,PC}
201
202 /* Global variables */
203                 .global COUNT
```

```
204   COUNT:          .word   0x0                 // used by timer
205                   .global RUN                 // used by pushbutton KEYs
206   RUN:            .word   0x1                 // initial value to increment COUNT
207   RATE:           .word   25000000
208   TIME:           .word   0x0                 // used for real-time clock
209                   .global HEX_CODE
210   HEX_CODE:       .word   0x0                 // used for 7-segment displays
211   BIT_CODE:       .byte   0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
212                   .byte   0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
213                   .skip   2                   // pad with 2 bytes to maintain word
                      alignment
214
215                   .end
216
```