```
1 ;-------------------------------------------------------------------------------
2 ; MSP430 Assembler Code Template for use with TI Code Composer Studio
3 ;
4 ;
5 ;-------------------------------------------------------------------------------
6             .cdecls C,LIST,"msp430.h"       ; Include device header file
7
8 ;-------------------------------------------------------------------------------
9             .def    RESET                   ; Export program entry-point to
10                                            ; make it known to linker.
11
12 ;-------------------------------------------------------------------------------
13            .data                           ; Assemble into program memory.
14            .retain                         ; Override ELF conditional linking
15            .retainrefs                     ; And retain any sections that have
16
17 LENGTH      .set    10
18 x_array:    .word       2, 160, 252, 105,  42
19 e_array:    .word    1030,  77,   0, 180, 103
20 N_array:    .word     221, 221, 207, 225, 209
21 y_array:    .space  LENGTH
22 ;-------------------------------------------------------------------------------
23            .text                           ; Assemble into program memory.
24            .retain                         ; Override ELF conditional linking
25            .retainrefs                     ; And retain any sections that have
26 ;-------------------------------------------------------------------------------
27 RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
28 StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
29
30 ;-------------------------------------------------------------------------------
31 ; Main loop here
32 ;-------------------------------------------------------------------------------
33            mov.w   #LENGTH-2, R4
34
35 next:       mov.w   x_array(R4), R6
36            mov.w   e_array(R4), R7
37            mov.w   N_array(R4), R8
38            call    #mod_exp
39            mov.w   R5, y_array(R4)
40
41            decd.w  R4
42            jhs     next
43
44 done:       jmp     done
45            nop
46
47
48 ;-------------------------------------------------------------------------------
49 ; Subroutine: mod_exp
50 ; Inputs: R6 unsigned 16-bit integer x -- returned unchanged
51 ;         R7 unsigned 16-bit integer e -- returned unchanged
52 ;         R8 unsigned 16-bit nonzero integer N -- returned unchanged
53 ;
54 ;           Both x and N need to be strictly less than 256
55 ;
56 ; Output: R5 unsigned 16 bit integer y -- R5 is output, may be changed
57 ;           y = x^e % N
```

```
58 ;               i.e., 0 <= y < N is the remainder when x^e is divided by N
59 ;
60 ;
61 ; All other core registers in R4-R15 unchanged
62 ;-------------------------------------------------------------------------------
63 mod_exp:
64 ; Level 1
65 ; Indicate your level above and add your code below
66
67             push.w  R6                  ; R6 is x
68             push.w  R7                  ; R7 is e
69             push.w  R8                  ; R8 is N
70             push.w  R9                  ; R9 is index
71
72             clr.w   R5
73             clr.w   R9
74
75             inc.w   R9                  ; Increment counter to 1
76             mov.w   R6, R5              ; Copy R6 to R5
77
78 edgeCase0:
79             cmp.w   #0, R7              ; Check if e = 0
80             jne     edgeCase1
81             mov.w   #1, R5              ; Assign 0 to R5
82             jmp     end
83
84 edgeCase1:
85             cmp.w   #1, R7              ; Check if e = 1
86             jne     loop1
87             jmp     end
88
89 loop1:
90             push.w  R6
91             mov.w   R5, R6
92             jmp     x_times_y           ; Multiply y times y
93             pop.w   R6                  ; Restore x
94             jmp     mod                 ; Find mod of y
95             mov.w   R8, R5              ; Move N to y
96             push.w  R9
97             rra.w   R9                  ; Double counter
98             cmp.w   R7, R9              ; Compare doubled counter to e
99             jlo     loop1
100            pop.w   R9                  ; Restore counter otherwise
101
102 loop2:
103            cmp.w   R7, R9              ; Compare counter to e
104            jge     end                 ; End loop if counter equals e
105            jmp     x_times_y           ; Multiply x times y
106            jmp     mod                 ; Find mod of y
107            inc.w   R9
108            jmp     loop2
109
110 end:
111            pop.w   R11                 ; Restore all registers
112            pop.w   R10
113            pop.w   R8
114            pop.w   R7
```

```
115             pop.w   R6
116
117             ret
118
119 ;-------------------------------------------------------------------------------
120 ; Subroutine: mod
121 ; input: R5 unsigned 16-bit integer x -- may be modified
122 ;        R8 unsigned 16-bit nonzero integer N -- returned unchanged
123 ;           you can assume that R8 is nonzero, no need to check
124 ;
125 ; output: R5 unsigned 16-bit integer y
126 ;           y = x % N is a number 0 <= y < N s.t. x=y mod N
127 ;           y is the remainder when x is divided by N
128 ;
129 ; All other core registers in R4-R15 unchanged
130 ;-------------------------------------------------------------------------------
131 mod:
132 ; Add your code below
133
134 divisionLoop:                       ; Division loop
135             sub.w   R8, R5          ; Subtract R8 from R5
136             cmp.w   R8, R5          ; Break loop if R8 > R5
137             jlo     divisionLoop
138
139             ret
140
141 ;-------------------------------------------------------------------------------
142 ; Subroutine: x_times_y
143 ; Inputs: unsigned byte x in R5 -- returned unchanged
144 ;         unsigned byte y in R6 -- returned unchanged
145 ;
146 ; Output: unsigned word in R12 -- R12 = R5 * R6
147 ;
148 ; All other core registers in R4-R15 unchanged
149 ;-------------------------------------------------------------------------------
150 x_times_y:
151
152 ; Save afftected core registers on stack - You can add this part last once you
153 ; know which registered are modified
154             push.w  R6
155             push.w  R10
156             push.w  R11
157
158             clr.w   R12             ; R12 will accumulate R5*R6
159             clr.w   R10             ; R10 will index bits j = 0, 1, ..., 7
160             mov.w   #BIT0, R11      ; R11 has the bitmask to use with tst.w
161
162 check_next_bit:
163             bit.w   R11, R5         ; Is the jth bit 1?
164             jnc     prep_next_bit   ; If not prepare for checking next bit
165
166             add.w   R6, R12         ; Bit j is 1, add
167
168 prep_next_bit:
169             rla.w   R11             ; Prepare next bitmask
170             rla.w   R6              ; Prepare shifted version of R6
171             inc.w   R10             ; increase bit index
```

```
172             cmp.w   #8, R10         ; Are we done with all bits?
173             jlo     check_next_bit
174
175 ; Restore saved core registers from stack
176 ; Watch the order and make sure not to leave anything behind
177             pop.w   R11
178             pop.w   R10
179             pop.w   R6
180
181             ret
182
183 ;-------------------------------------------------------------------------------
184 ; Stack Pointer definition
185 ;-------------------------------------------------------------------------------
186             .global __STACK_END
187             .sect   .stack
188
189 ;-------------------------------------------------------------------------------
190 ; Interrupt Vectors
191 ;-------------------------------------------------------------------------------
192             .sect   ".reset"                ; MSP430 RESET Vector
193             .short  RESET
194
195
```