```
; to_decimal.asm
; CSC 230: Summer 2022
;
; ************************************
; Sample code provided that may be useful
; in a solution to Assignment #3
;
; Author: Mike Zastre (2019-Jul-13)
;

.include "m2560def.inc"

.cseg
.org 0

.equ MAX_POS = 5     ; maximum length of text representation
.equ NUMBER = 32767  ; number to use for testing

    ; The code below simply uses some value to call
    ; the function to_decimal_text. This is meant to
    ; help with testing.
    ;

    ; First parameter: a 16-bit signed & positive integer.
    ;
    ; Second parameter: location in data memory where the
    ; character representation of the decimal equivalent
    ; is to be stored.
    ;
    ldi r17, high(NUMBER)
    ldi r16, low(NUMBER)
    push r17
    push r16

    ldi r17, high(COUNTER_TEXT)
    ldi r16, low(COUNTER_TEXT)
    push r17
    push r16

    rcall to_decimal_text

stop:
    rjmp stop

; First parameter: 16-bit value for which a
; text representation of its decimal form is to
; be stored.
;
; Second parameter: 16-bit address in data memory
; in which the text representation is to be stored.
;
to_decimal_text:
    .def countL=r18
    .def countH=r19
    .def factorL=r20
    .def factorH=r21
    .def multiple=r22
    .def pos=r23
    .def zero=r0
    .def ascii_zero=r16

    push countH
    push countL
    push factorH
    push factorL
```

```
    push multiple
    push pos
    push zero
    push ascii_zero
    push YH
    push YL
    push ZH
    push ZL
    in YH, SPH
    in YL, SPL

    ; fetch parameters from stack frame
    ;
    .set PARAM_OFFSET = 16
    ldd countH, Y+PARAM_OFFSET+3
    ldd countL, Y+PARAM_OFFSET+2

    ; this is only designed for positive
    ; signed integers; we force a negative
    ; integer to be positive.
    ;
    andi countH, 0b01111111

    clr zero
    clr pos
    ldi ascii_zero, '0'

    ; The idea here is to build the text representation
    ; digit by digit, starting from the left-most.
    ; Since we need only concern ourselves with final
    ; text strings having five characters (i.e., our
    ; text of the decimal will never be more than
    ; five characters in length), we begin we determining
    ; how many times 10000 fits into countH:countL, and
    ; use that to determine what character (from '0' to
    ; '9') should appear in the left-most position
    ; of the string.
    ;
    ; Then we do the same thing for 1000, then
    ; for 100, then for 10, and finally for 1.
    ;
    ; Note that for *all* of these cases countH:countL is
    ; modified. We never write these values back onto
    ; that stack. This means the caller of the function
    ; can assume call-by-value semantics for the argument
    ; passed into the function.
    ;

to_decimal_next:
    clr multiple

to_decimal_10000:
    cpi pos, 0
    brne to_decimal_1000
    ldi factorL, low(10000)
    ldi factorH, high(10000)
    rjmp to_decimal_loop

to_decimal_1000:
    cpi pos, 1
    brne to_decimal_100
    ldi factorL, low(1000)
    ldi factorH, high(1000)
    rjmp to_decimal_loop
```

```
to_decimal_100:
    cpi pos, 2
    brne to_decimal_10
    ldi factorL, low(100)
    ldi factorH, high(100)
    rjmp to_decimal_loop

to_decimal_10:
    cpi pos, 3
    brne to_decimal_1
    ldi factorL, low(10)
    ldi factorH, high(10)
    rjmp to_decimal_loop

to_decimal_1:
    mov multiple, countL
    rjmp to_decimal_write

to_decimal_loop:
    inc multiple
    sub countL, factorL
    sbc countH, factorH
    brpl to_decimal_loop
    dec multiple
    add countL, factorL
    adc countH, factorH

to_decimal_write:
    ldd ZH, Y+PARAM_OFFSET+1
    ldd ZL, Y+PARAM_OFFSET+0
    add ZL, pos
    adc ZH, zero
    add multiple, ascii_zero

    st Z, multiple

    inc pos
    cpi pos, MAX_POS
    breq to_decimal_exit
    rjmp to_decimal_next

to_decimal_exit:
    pop ZL
    pop ZH
    pop YL
    pop YH
    pop ascii_zero
    pop zero
    pop pos
    pop multiple
    pop factorL
    pop factorH
    pop countL
    pop countH

    .undef countL
    .undef countH
    .undef factorL
    .undef factorH
    .undef multiple
    .undef pos
    .undef zero
    .undef ascii_zero

    ret
```

```
.dseg
.org 0x200
COUNTER_TEXT: .byte 6
COUNTER_VAL: .byte 2
```