# ECED 3204 Microprocessor

# Assignment #3 Reference Solution

# http://www.jasongu.org/3204/assignments.html

Assignment #3 contains the following problems:

**E5.1** Write a subroutine that can create a time delay that is a multiple of 1 ms. The multiple is passed in register r16, assuming that the CPU clock is 16 MHz.

The following subroutine can create a delay that is a multiple of 1 ms assuming that the CPU clock frequency is 16-MHz:

```
; ---------------------------------------------------------------------------------------------------
----------
; This subroutine can create a time delay that is a multiple of 1 ms. The multiple is passed
; in register r16. This subroutine assumes that the frequency of the CPU clock is 16 MHz.
; ---------------------------------------------------------------------------------------------------
----------
            .def    iLpCnt = r20
            .def    eLpCnt = r21
delayby1ms:
loop2:      ldi     eLpCnt,2        ; set external loop count to 2
loop1:      ldi     iLpCnt,250      ; set loop count to 250
loop0:      push    r0              ; 2 CPU clock cycles
            pop     r0              ; 2 CPU clock cycles
            push    r0
            pop     r0
            push    r0
            pop     r0
            push    r0
            pop     r0
            push    r0
            pop     r0
            push    r0
            pop     r0
            push    r0
            pop     r0
            push    r0
            pop     r0
            nop                     ; 1 CPU clock cycle
            dec     iLpCnt          ; 1 CPU clock cycle
            brne    loop0           ; take 2 (1) CPU clock cycles when branch is taken
(not taken)
            dec     eLpCnt
            brne    loop1
```

```
        dec     r16
        brne    loop2
        ret
```

**E5.3** Write a subroutine that can divide a 16-bit signed integer into another 16-bit signed integer. The dividend is passed in registers r16-r19, and r18-r19, respectively. The quotient is returned in registers r24-r25, and the remainder is returned in registers r22-r23, respectievly.

The following subroutine can divide a 16-bit signed integer into another 16-bit signed integer.

```
        .include <m1280def.inc>
        .equ    num1H = 0x76   ; high byte of the first test data
        .equ    num1L = 0x00   ; low byte of the first test data
        .equ    num2H = 0x00   ; high byte of the second test data
        .equ    num2L = 0x64   ; low byte of the second test data
        .cseg
        .org    0x00
        rjmp    start
        .org    0xF6
start:  ldi     r16,low(RAMEND)
        out     SPL,r16
        ldi     r16,high(RAMEND)
        out     SPH,r16
        ldi     r16,num1L       ; pass the dividend in r16~r17
        ldi     r17,num1H       ;    "
        ldi     r18,num2L       ; pass the divisor in r18~r19
        ldi     r19,num2H       ;    "
        call    div16S
forever: nop
        rjmp    forever
;-----------------------------------------------------------------------------------------------------
---
; The next subroutine divides the 16-bit signed numbers contained in r18~r19
; into the 16-bit signed number contained in r16~r17 and returns the quotient
; in r25~r24 and the remainder in r23~r22.
; -----------------------------------------------------------------------------------------------------
---
        .def    lpcnt = r20
        .def    tmpL = r0
        .def    tmpH = r1
        .def    signD = r8      ; sign of dividend
        .def    sign = r21      ; sign of quotient
```

```
div16S:  push    r27             ; save XH register
         push    r26             ; save XL register
         ldi     lpcnt,16        ; set up divide iteration count
         clr     signD
         clr     sign
         sbrs    r17,7           ; check the sign bit of the dividend
         rjmp    chkdvr          ; go to check the sign of divisor
         inc     signD           ; record the sign of dividend to be negative
         inc     sign            ; record the sign of quotient to be negative
         movw    r26,r16
         com     r26             ; find the magnitude of the dividend
         com     r27             ;    "
         adiw    r26,1
         movw    r16,r26         ; transfer the magnitude of the dividend back to
r16~r17
chkdvr:  sbrs    r19,7           ; check the sign of the divisor
         rjmp    startD          ; no need to change anything
         ldi     r27,0x01        ; toggle the sign bit of quotient
         eor     sign,r27;       "
         movw    r26,r18         ; compute the magnitude of the divisor
         com     r26             ;    "
         com     r27             ;    "
         adiw    r26,1           ;    "
         movw    r18,r26         ; transfer the magnitude of divisor back to r18~r19
startD:  movw    r24,r16         ; place dividend in Q register
         clr     r23             ; load 0 in R register
         clr     r22             ;    "
dvloop:  lsl     r24             ; shift R:Q to the left one place
         rol     r25             ;    "
         rol     r22             ;    "
         rol     r23             ;    "
         movw    tmpL,r22        ; save R in temporary registers
         sub     tmpL,r18        ; compute R - S
         sbc     tmpH,r19        ;    "
         brmi    less            ; branch if divisor is larger
         movw    r22,tmpL        ; save the difference in R
         ori     r24,0x01        ; set the lsb of Q to 1
         rjmp    nxtb
less:    andi    r24,0xFE        ; set the lsb of Q to 0
nxtb:    dec     lpcnt
         brne    dvloop
         sbrs    signD,0
         rjmp    chkSoQ
         movw    r26,r22         ; change the remainder to it's two's complement if
         com     r26             ; the dividend is negative
         com     r27             ;    "
```

```
            adiw    r26,1           ;   "
            movw    r22,r26         ;   "
chkSoQ:     sbrs    sign,0          ; check the sign of quotient
            rjmp    doneDS          ; nothing need to be done on quotient when it is
positive
            movw    r26,r24         ; quotient is negative, compute it's two's
complement
            com     r26             ;   "
            com     r27             ;   "
            adiw    r26,1           ;   "
            movw    r24,r26         ;   "
doneDS:     pop     r26
            pop     r27
            ret
// end of program
```

**E5.5** Write a subroutine that can divide a 32-bit signed integer into another 32-bit signed integer. The dividend is passed in registers r16-r19, and the divisor is passed in the stack. The quotient is returned in registers r22-r25, and the remainder is returned in registers r16-r19.

Write a subroutine that can divide a 32-bit signed integer into another 32-bit signed integer. The subroutine and its test program are as follows

```
            .include  <atxmega128A1def.inc>
            .macro dealloc_stk
            in      r28,CPU_SPL
            in      r29,CPU_SPH
            adiw    r28,@0          ; allocate k bytes in the stack
            out     CPU_SPL,r28
            out     CPU_SPH,r29
            .endmacro
            .equ    dd3 = 0xF1      ; test data for dividend
            .equ    dd2 = 0x18      ;   "
            .equ    dd1 = 0x65      ;   "
            .equ    dd0 = 0x7B      ;   "
            .equ    dr3 = 0x00      ; test data for divisor
            .equ    dr2 = 0x00      ;   "
            .equ    dr1 = 0xF4      ;   "
            .equ    dr0 = 0x24      ;   "

            .cseg
            .org    0x00
            rjmp    start
            .org    0xF6
```

```
start:      ldi     YL,low(RAMEND) ; initialize stack pointer SP
            ldi     YH,high(RAMEND)    ;         "
            out     CPU_SPL,YL  ;    "
            out     CPU_SPH,YH  ;    "
            ldi     r28,dr3         ; pass divisor in the stack
            push    r28             ;    "
            ldi     r28,dr2         ;    "
            push    r28             ;    "
            ldi     r28,dr1         ;    "
            push    r28             ;    "
            ldi     r28,dr0         ;    "
            push    r28             ;    "
            ldi     r16,dd0         ; pass dividend in r16~r19
            ldi     r17,dd1         ;    "
            ldi     r18,dd2         ;    "
            ldi     r19,dd3         ;    "
            call    div32S          ;
            dealloc_stk 4
forever:    rjmp    forever
; ------------------------------------------------------------------------------------------------------------
---------
; The subroutine divides a 32-bit unsigned integer into another 32-bit unsigned integer.
; The 32-bit divisor is passed in the stack whereas the 32-bit unsigned dividend is passed
; in registers r16~r19. The quotient is returned in registers r22~r25 and the remainder
; is returned in registers r16~r19.
; ------------------------------------------------------------------------------------------------------------
---------
            .def    lpcnt = r21     ; loop count for repeated shifted-subtraction
            .def    tmp3  = r11     ; used to hold difference temporarily
            .def    tmp2  = r10     ;    "
            .def    tmp1  = r9      ;    "
            .def    tmp0  = r8      ;    "
            .def    signD = r30     ; record the sign of the dividend
            .def    sign  = r31     ; record the sign of the quotient
            .equ    dvr3  = 12      ; offset of msb of divisor from stack pointer
            .equ    dvr2  = 11      ;    "
            .equ    dvr1  = 10      ;    "
            .equ    dvr0  = 9       ; (these offsets must be incremented  by 1 for
Xmega128A1)
div32S:     push    YH              ;
            push    YL              ;    "
            push    r31             ; save Z register in stack
            push    r30
            push    r27             ; save X register in stack
            push    r26
            in      YL,CPU_SPL      ; use Y register as pointer to the stack top
```

```
            in      YH,CPU_SPH  ;    "
            clr     signD       ; assume the dividend is positive
            clr     sign        ; assume the quotient is positive
            sbrs    r19,7       ; check the sign of dividend
            rjmp    chkDvr
            inc     signD       ; change the sign of dividend to negative
            inc     sign        ; change the sign of quotient to negative
            com     r16         ; compute the magnitude of dividend
            com     r17         ;    "
            com     r18         ;    "
            com     r19         ;    "
            movw    r26,r16     ;    "
            adiw    r26,1       ;    "
            movw    r16,r26     ;    "
            clr     r27         ;    "
            adc     r18,r27     ;    "
            adc     r19,r27     ;    "
chkDvr:     ldd     r0,Y+dvr0   ; place divisor in registers r0~r3
            ldd     r1,Y+dvr1   ;    "
            ldd     r2,Y+dvr2   ;    "
            ldd     r3,Y+dvr3   ;    "
            sbrs    r3,7        ; check the sign of divisor
            rjmp    startDv
            com     r0          ; compute the magnitude of the divisor
            com     r1          ;    "
            com     r2          ;    "
            com     r3          ;    "
            movw    r26,r0      ;    "
            adiw    r26,1       ;    "
            movw    r0,r26      ;    "
            clr     r27         ;    "
            adc     r2,r27      ;    "
            adc     r3,r27      ;    "
            ldi     r27,1       ;    ; toggle the sign of the quotient
            eor     sign,r27    ;    "
startDv:
            ldi     lpcnt,32    ; there should be 32 iterations
            movw    r24,r18     ; transfer dividend to Q register
            movw    r22,r16     ;    "
            clr     r16         ; use registers r16~r19 as R register and
            clr     r17         ; initialize them to 0
            clr     r18         ;    "
            clr     r19         ;    "
dvloop32:
            lsl     r22         ; shift R:Q to the left one place
            rol     r23         ;    "
```

```
            rol     r24             ;    "
            rol     r25             ;    "
            rol     r16             ;    "
            rol     r17             ;    "
            rol     r18             ;    "
            rol     r19             ;    "
            movw    tmp0,r16        ; transfer R register to tmp0~tmp3
            movw    tmp2,r18        ;    "
            sub     tmp0,r0
            sbc     tmp1,r1
            sbc     tmp2,r2
            sbc     tmp3,r3
            brmi    less32
            movw    r16,tmp0        ; save the difference in R register
            movw    r18,tmp2        ;    "
            ori     r22,0x01        ; set the lsb of Q register to 1
            rjmp    nextb
less32:     andi    r22,0xFE        ; clear the lsb of Q register to 0
nextb:      dec     lpcnt
            brne    dvloop32
            sbrs    signD,0
            rjmp    chkSoQ
            com     r16             ; compute the 2's complement of the remainder
            com     r17             ; if the dividend is negative
            com     r18             ;    "
            com     r19             ;    "
            movw    r26,r16         ;    "
            adiw    r26,1           ;    "
            movw    r16,r26         ;    "
            clr     r27             ;    "
            adc     r18,r27         ;    "
            adc     r19,r27         ;    "
chkSoQ:     sbrs    sign,0
            rjmp    doneD32         ; quotient is positive, so do nothing
            com     r22             ; quotient is negative, so compute its
            com     r23             ; two's complement
            com     r24             ;    "
            com     r25             ;    "
            movw    r26,r22         ;    "
            adiw    r26,1           ;    "
            movw    r22,r26         ;    "
            clr     r27             ;    "
            adc     r24,r27         ;    "
            adc     r25,r27         ;    "
doneD32:    pop     r26
            pop     r27
```

```
        pop     r30
        pop     r31
        pop     YL              ; restore Y from stack
        pop     YH              ;    "
        ret
// End of program
```

**E5.7** Write a subroutine that finds the greatest common divisor of two 16-bit unsigned integers.
Two registers are passed in registers r16-r17 and r18-r19, respectivly. The gcd is returned in registers r22-r23.

Write a subroutine that finds the greatest common divisor of two 16-bit unsigned integers.
The GCD of two integers can be found by using the Euclidian method:

**Step 1**

If m = n then
    gcd ← m;
    return;

**Step 2**

If n < m then swap m and n.

**Step 3**

gcd ← 1.
If m = 1 or n = 1 then return.

**Step 4**

p = n % m;

**Step 5**

if (p == 0) then m is the gcd.
else
    n ← m;
    m ← p;
go to step 4.

The assembly subroutine that finds the gcd and the test program are as follows:

```
        .include <can128def.inc>
        .equ    n1 = 18000      ; test number 1
        .equ    n2 = 9000       ; test number 2
        .cseg
        .org    0x00
        rjmp    start
        .org    0xF6
start:  ldi     r16,low(RAMEND) ; initialize SP
        out     SPL,r16         ;    "
        ldi     r16,high(RAMEND)    ;       "
```

```
            out     SPH,r16             ;    "
            ldi     r16,low(n1)     ; pass the first number
            ldi     r17,high(n1)    ;    "
            ldi     r18,low(n2)     ; pass the second number
            ldi     r19,high(n2)    ;    "
            call    findGCD
            nop
again:      rjmp    again
; ---------------------------------------------------------------------------------------------------
------------
; The findGCD subroutine computes the gcd of two 16-bit unsigned 16-bit integers held
; in r16~r17 and r18~r19, respectively. The GCD us returned in registers r22~r23.
; ---------------------------------------------------------------------------------------------------
------------
            .def    tH = r3
            .def    tL = r2
findGCD:    cp      r16,r18
            cpc     r17,r19
            brne    normal
            movw    r22,r16         ; if (m == n), then gcd = n
            ret
normal:     ldi     r22,1           ; initialize gcd to 1
            clr     r23
            cp      r16,r22         ; if (n == 1), then return
            cpc     r17,r23
            brne    chkN
            ret
chkN:       cp      r18,r22         ; if (m == 1), then return
            cpc     r19,r23
            brne    doIt
            ret
doIt:       movw    tL,r18          ; save a copy of divisor
            call    div16U          ; compute
            cpi     r23,0
            brne    next1
            cpi     r22,0
            brne    next1
            movw    r22,tL
            ret
next1:      movw    r16,tL          ; make the divisor as the dividend
            movw    r18,r22         ; make the remainder as the divisor
            rjmp    doIt
            .include "div16U.asm"
// End of program
```