Embedded Systems Lab 4

Introduction

In this lab we built a cooling fan that uses PWM for speed control connected with a LCD display to provide basic information about the fan. To control the speed of the fan, we are using a RPG and finally we have a push button to cycle through the modes of the LCD display.

Pseudocode

For this lab we start with a large configuration block that configures, sets up, and initializes the timers, ports, LCD, ISRs, and then enters the main loop. Within the main loop, we either run the code for Mode A or B. To switch between the modes, we have an interrupt that will increment between the modes. To control the duty cycle we use a register that holds an offset, this changes the preload for the timer. When printing to the LCD screen, we have a subroutine that will print everything we need.

Configure and Setup Ports/Pins

Initialize LCD

Configure and Activate ISRs

Initialize Timers

Main loop:

modeA:

check RPG

display/update mode

display/update DC

If tachometer timer overflowed:

Warning

Else

ok

modeB:

check RPG

display/update mode

display/update DC

If RPM low:

Warning

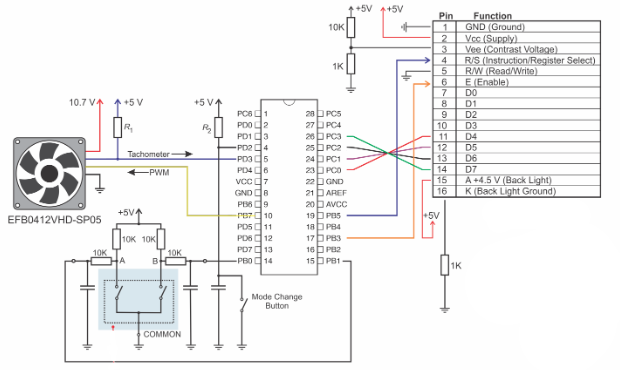
Else

Ok

Interrupts and helper functions

Schematic

We connected the micro controller together according to the following diagram.



Conclusion

While developing out code I realized that we were doing things slightly differently from most other people in the class. I also learned that this lab was very useful for the exam and was glad that we did part of it before the exam. Due to this, it was very hard to troubleshoot. I also learned, as always, it takes more time than I thought to complete the lab. Within our code, we used 2 functions from libraries, we used mpy16u and div16 u from the AVR 200 code provided in resources to us.

Code

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Lab4

;;

;; Authors : Ben Weinberg, Daniel Nunez

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

#define offset r28 ; Offset from base count

#define mode r27

#define MODEA 0

#define MODEB 1

#define RS PORTB,5

#define EN PORTB,3

rjmp begining ; Jump over ISRs to the start of the code

.org 0x0001 ; Pushbutton ISR

rjmp pushISR

.org 0x0002 ; Tachometer ISR

rjmp tachISR

.org 0x000F ; PWM ISR

rjmp timer0\_cmp

; Start Strings

startMSG: .db "Welcome!",0x00,0x00

emptyLine: .db " ",0x00,0x00 ; string of 16 blanks

; ModeA Strings

modeABegin: .db "Mode A: ", 0x00,0x00

modeAOK: .db "OK ",0x00,0x00

modeAWarn: .db "ALARM ",0x00,0x00

; ModeB Strings

modeBBegin: .db "Mode B: ",0x00,0x00

modeBOK: .db "OK ",0x00

modeBWarn: .db "LOW RPM",0x00

; DC String

dcBegin: .db "DC = ",0x00

begining:

; Configs for everything

; Set the SP

ldi r22,LOW(RAMEND)

out spl,r22

ldi r22,HIGH(RAMEND)

out sph,r22

; LCD Data Lines are outputs

push r23

ldi r23,0x0F

out DDRC,r23

; PB5 & PB3 are outputs

ldi r23,0x28

out DDRB,r23

sbi EN

; Configure both INT0 and INT1 to fire on HIGH-LOW transistion.

lds r23,EICRA

ori r23,0x02 ; INT0

ori r23,(0x02<<2) ; INT1

sts EICRA,r23

; Turn INT0 & INT1 on.

in r23,EIMSK

ori r23,0x01 ; INT0

ori r23,(1<<1) ; INT1

out EIMSK,r23

pop r23

ldi offset,INT(2\*50) ; Startup duty cycle

; Initialize TIMER0, LCD, and print Starting msg

rcall timerInit

rcall LCDInit

rcall clearLCD

rcall printStartMSG

push r24

ldi r24,2

p1: rcall delay\_5ms

dec r24

brne p1

pop r24

; Initialize 16-bit Timer that is used to measure RPM

rcall startTimer16

; Initialize the startup display.

ldi mode,MODEA

sei

ldi r24,1

; Main loop

main:

cpi mode,MODEA

breq mA

rjmp mb

; Mode A

mA:

rcall checkRPG

rcall showDC

rcall showMode

in r22,TIFR1

sbrs r22,0 ; no overflow

rjmp AOK

ldi r30,LOW(2\*modeAWarn)

ldi r31,2\*HIGH(modeAWarn)

rcall printCStr

rjmp main

AOK:

ldi r30,LOW(2\*modeAOK)

ldi r31,2\*HIGH(modeAOK)

rcall printCStr

rjmp main

; Mode B

mB:

rcall checkRPG

rcall showDC

rcall showMode

mov r22,r12

cpi r22,1 ; Check in the count in r12:r11

brsh BOK ; greater than 97 => 2400 rpm

mov r22,r11

cpi r22,97

brsh BOK

ldi r30,LOW(2\*modeBOK)

ldi r31,2\*HIGH(modeBOK)

rcall printCStr

rjmp main

BOK:

ldi r30,LOW(2\*modeBWarn)

ldi r31,2\*HIGH(modeBWarn)

rcall printCStr

rjmp main

; Helper Functions

printStartMSG:

ldi r30,LOW(2\*startMSG)

ldi r31,2\*HIGH(startMSG)

rcall printCStr

ret

clear2ndLine:

rcall moveCursor2ndLine

ldi r30,LOW(2\*emptyLine)

ldi r31,2\*HIGH(emptyLine)

rcall printCStr

ret

showMode:

rcall moveCursor2ndLine

cpi mode,MODEA

brne showModeB

ldi r30,LOW(2\*modeABegin)

ldi r31,2\*HIGH(modeABegin)

rcall printCStr

ret

showModeB:

ldi r30,LOW(2\*modeBBegin)

ldi r31,2\*HIGH(modeBBegin)

rcall printCStr

ret

checkMode:

sbic PIND,2

rjmp cm2 ; high -> return

cm1:

sbis PIND,2 ; low -> wait for high

rjmp cm1

inc mode

cpi mode,0x02 ; bounds

brne cm2 ; within bounds -> return

ldi mode,0x00 ; out of bounds -> back to 0

cm2:

ret

; restart timer for PWN generation (with new ofset)

timer0\_cmp:

push r25

in r25,SREG

push r25

mov r25,offset

out OCR0B,r25

pop r25

out SREG,r25

pop r25

reti

; ISR for pushbutton, cycle the mode

pushISR:

push r25

in r25,SREG

push r25

rcall checkmode

pop r25

out SREG,r25

pop r25

reti

; ISR for tachometer. reloads timer/clears flag

tachISR:

push r25

in r25,SREG

push r25

ldi r25,0

rcall stopTimer16

cli

lds r11,TCNT1L ; Read TCNT1 into r12:r11

lds r12,TCNT1H

sts TCNT1H,r25 ; Set TCNT1 to 0

sts TCNT1L,r25

sei

in r25,TIFR1 ; Clear overflow flag

ori r25,0x01

out TIFR1,r25

rcall startTimer16

pop r25

out SREG,r25

pop r25

reti

.dseg

dtxt: .db 0x00,0x00,0x00,0x00,0x00,0x00,0x00

.cseg

; displays duty cycle on LCD

showDC:

rcall moveCursorHome

ldi r30,LOW(2\*dcBegin)

ldi r31,2\*HIGH(dcBegin)

rcall printCStr

push r26

; Insert percentage symbol.

ldi r26,'%'

sts dtxt+5,r26

; Multiply by 5.

mov r26,offset

inc r26

mov mc16uL,r26

ldi mc16uH,0

ldi mp16uL,5

ldi mp16uH,0

rcall mpy16u

; Isolate the least significant digit, which is the remainder

; after dividing by 10.

mov dd16uL,m16u0

mov dd16uH,m16u1

ldi dv16uL,10

ldi dv16uH,0

rcall div16u ; Result: r17:r16, rem: r15:r14

; Format the remainder as ASCII and move to its place in RAM.

ldi r26,0x30

add r26,r14 ; Convert to ASCII

sts dtxt+4,r26 ; Store in RAM

; Insert decimal point.

ldi r26,'.'

sts dtxt+3,r26

; Repeat using the remainder as our starting point.

mov dd16uL,r16

mov dd16uH,r17

rcall div16u ; Result: r17:r16, rem: r15:r14

; Format the remainder as ASCII and move to its place in RAM.

ldi r26,0x30

add r26,r14 ; Convert to ASCII

sts dtxt+2,r26 ; Store in RAM

; Repeat using the remainder as our starting point.

mov dd16uL,r16

mov dd16uH,r17

rcall div16u ; Result: r17:r16, rem: r15:r14

; Format the remainder as ASCII and move to its place in RAM.

ldi r26,0x30

add r26,r14 ; Convert to ASCII

sts dtxt+1,r26 ; Store in RAM

; Repeat using the remainder as our starting point.

mov dd16uL,r16

mov dd16uH,r17

rcall div16u ; Result: r17:r16, rem: r15:r14

; Format the remainder as ASCII and move to its place in RAM.

ldi r26,0x30

add r26,r14 ; Convert to ASCII

sts dtxt,r26 ; Store in RAM

; Now print out the characters.

lds r29,dtxt

rcall printChar

lds r29,dtxt+1

rcall printChar

lds r29,dtxt+2

rcall printChar

lds r29,dtxt+3

rcall printChar

lds r29,dtxt+4

rcall printChar

lds r29,dtxt+5

rcall printChar

pop r26

ret

; RPG handling

checkRPG:

push r24

push r26

in r26,PINB

andi r26,0x03

cpi r26,0x03

breq cR4

cRx: in r24,PINB

andi r24,0x03

cpi r24,0x03

brne cRx

cpi r26,0x01

brne cR1

inc offset

rjmp cR2

cR1:

cpi r26,0x02

brne cR2

dec offset

rjmp cR2

cR2:

cpi offset,199

brlo cR3

ldi offset,199

rjmp cR4

cR3:

cpi offset,0

brne cR4

ldi offset,1

cR4: pop r26

pop r24

ret

; print string to LCD

printCStr:

push r29

str01:

lpm

mov r29,r0

cpi r29,0 ; Check for end of string

brne str02

pop r29

ret

str02:

rcall printChar

adiw zl,1 ; Increment Z, point to next char

rjmp str01

pop r29

ret

; Prints char to LCD, from r29

printChar:

push r29

sbi EN

swap r29

out PORTC,r29

cbi EN

sbi EN

swap r29

out PORTC,r29

cbi EN

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

rcall delay\_100us

pop r29

ret

; Clears LCD

clearLCD:

push r29

cbi RS

ldi r29,0x01

rcall printChar

rcall delay\_5ms

ldi r29,0x02

rcall printChar

rcall delay\_5ms

sbi RS

pop r29

ret

; Move cursor home

moveCursorHome:

push r29

cbi RS

ldi r29,0x02

rcall printChar

sbi RS

pop r29

ret

; Move cursor to 2nd line

moveCursor2ndLine:

push r29

cbi RS

ldi r29,0x40

ori r29,(1<<7)

rcall printChar

sbi RS

pop r29

ret

; initilize LCD screen

LCDInit: ; initilize LDC screen

push r25 ; Wait 0.1 seconds

ldi r25,99

lc01:

dec r25

rcall delay\_1ms

brne lc01

cbi RS ; char->cmd mode

rcall delay\_1ms

ldi r25,0x03 ; Set to 8-bit mode

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x03 ; Set to 8-bit mode

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x03 ; Set to 8-bit mode

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x02 ; Set to 4-bit mode

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

; F(x) set 0x28: 2 lines, 5x7 font

ldi r25,0x02

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x08

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

; Clear display (0x01)

ldi r25,0x00

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x01

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

; Cursor + display

ldi r25,0x00

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x0C

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

; Cursor increment (0x06)

ldi r25,0x00

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

ldi r25,0x06

out PORTC,r25

cbi EN ; Strobe

rcall delay\_5ms

sbi EN

rcall delay\_100us

sbi RS ;cmd->char mode

rcall delay\_100us

pop r25

ret

; make a 100us delay

delay\_100us:

push r23

in r23,SREG

push r23

push r24

push r25

ldi r23,10

du1: ldi r24,5

du2: ldi r25,4

du3: dec r25

brne du3

dec r24

brne du2

dec r23

brne du1

pop r25

pop r24

pop r23

out SREG,r23

pop r23

ret

; make a 1ms delay

delay\_1ms:

push r23

in r23,SREG

push r23

push r24

push r25

ldi r23,100

d11: ldi r24,5

d12: ldi r25,4

d13: dec r25

brne d13

dec r24

brne d12

dec r23

brne d11

pop r25

pop r24

pop r23

out SREG,r23

pop r23

ret

; make a 5ms delay

delay\_5ms:

rcall delay\_1ms

rcall delay\_1ms

rcall delay\_1ms

rcall delay\_1ms

rcall delay\_1ms

ret

; starts 16 bit timer

startTimer16:

push r25

lds r25,TCCR1B

sbr r25,(1<<CS12)

cbr r25,(1<<CS11)

sbr r25,(1<<CS10)

sts TCCR1B,r25

pop r25

ret

; stop the 16 bit timer

stopTimer16:

push r25

lds r25,TCCR1B

cbr r25,(1<<CS12)

cbr r25,(1<<CS11)

cbr r25,(1<<CS10)

sts TCCR1B,r25

pop r25

ret

; initialize TIMER0 (makes 40kHz wave on PD5)

timerInit:

push r25

sbi DDRD,5 ; Make PD5 an output

in r25,TCCR0A

sbr r25,(1<<COM0B1)

sbr r25,(1<<WGM01)

sbr r25,(1<<WGM00)

out TCCR0A,r25

in r25,TCCR0B

sbr r25,(1<<WGM02)

sbr r25,(1<<CS00)

out TCCR0B,r25

ldi r25,0xC7

out OCR0A,r25

ldi r25,0x24

out OCR0B,r25

lds r25,TIMSK0

sbr r25,(1<<OCIE0A)

sts TIMSK0,r25

pop r25

ret

; "div16u" - 16/16 Bit Unsigned Division

; Subroutine Register Variables

.def drem16uL=r14

.def drem16uH=r15

.def dres16uL=r16

.def dres16uH=r17

.def dd16uL =r16

.def dd16uH =r17

.def dv16uL =r18

.def dv16uH =r19

; Code

div16u: clr drem16uL ;clear remainder Low byte

sub drem16uH,drem16uH;clear remainder High byte and carry

rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_1 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_2 ;else

d16u\_1: sec ; set carry to be shifted into result

d16u\_2: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_3 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_4 ;else

d16u\_3: sec ; set carry to be shifted into result

d16u\_4: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_5 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_6 ;else

d16u\_5: sec ; set carry to be shifted into result

d16u\_6: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_7 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_8 ;else

d16u\_7: sec ; set carry to be shifted into result

d16u\_8: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_9 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_10 ;else

d16u\_9: sec ; set carry to be shifted into result

d16u\_10:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_11 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_12 ;else

d16u\_11:sec ; set carry to be shifted into result

d16u\_12:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_13 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_14 ;else

d16u\_13:sec ; set carry to be shifted into result

d16u\_14:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_15 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_16 ;else

d16u\_15:sec ; set carry to be shifted into result

d16u\_16:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_17 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_18 ;else

d16u\_17: sec ; set carry to be shifted into result

d16u\_18:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_19 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_20 ;else

d16u\_19:sec ; set carry to be shifted into result

d16u\_20:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_21 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_22 ;else

d16u\_21:sec ; set carry to be shifted into result

d16u\_22:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_23 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_24 ;else

d16u\_23:sec ; set carry to be shifted into result

d16u\_24:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_25 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_26 ;else

d16u\_25:sec ; set carry to be shifted into result

d16u\_26:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_27 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_28 ;else

d16u\_27:sec ; set carry to be shifted into result

d16u\_28:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_29 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_30 ;else

d16u\_29:sec ; set carry to be shifted into result

d16u\_30:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_31 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_32 ;else

d16u\_31:sec ; set carry to be shifted into result

d16u\_32:rol dd16uL ;shift left dividend

rol dd16uH

ret

; "mpy16u" - 16x16 Bit Unsigned Multiplication

; Subroutine Register Variables

.def mc16uL =r16 ;multiplicand low byte

.def mc16uH =r17 ;multiplicand high byte

.def mp16uL =r18 ;multiplier low byte

.def mp16uH =r19 ;multiplier high byte

.def m16u0 =r18 ;result byte 0 (LSB)

.def m16u1 =r19 ;result byte 1

.def m16u2 =r20 ;result byte 2

.def m16u3 =r21 ;result byte 3 (MSB)

; Code

mpy16u: clr m16u3 ;clear 2 highest bytes of result

clr m16u2

lsr mp16uH ;rotate multiplier Low

ror mp16uL ;rotate multiplier High

brcc noadd0 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd0: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd1 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd1: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd2 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd2: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd3 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd3: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd4 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd4: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd5 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd5: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd6 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd6: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd7 ;if carry sett

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd7: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd8 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd8: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noadd9 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noadd9: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noad10 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noad10: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noad11 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noad11: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noad12 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noad12: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noad13 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noad13: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noad14 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noad14: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

brcc noad15 ;if carry set

add m16u2,mc16uL ; add multiplicand Low to byte 2 of res

adc m16u3,mc16uH ; add multiplicand high to byte 3 of res

noad15: ror m16u3 ;shift right result byte 3

ror m16u2 ;rotate right result byte 2

ror m16u1 ;rotate result byte 1 and multiplier High

ror m16u0 ;rotate result byte 0 and multiplier Low

ret