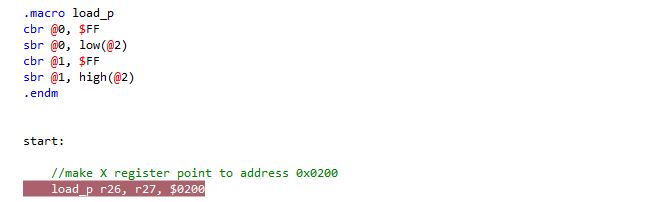
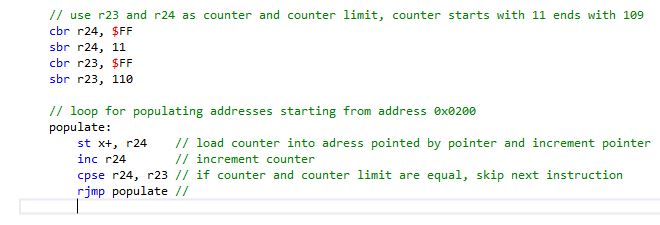
In this design assignment, we are doing indirect addressing. I used macro directive to load pointers, instead of using extra instructions.

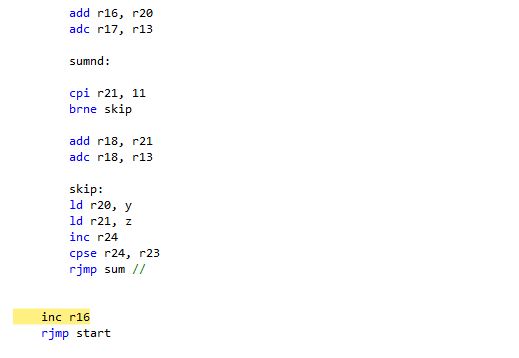
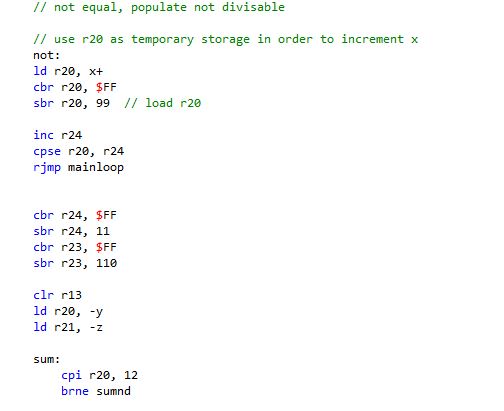
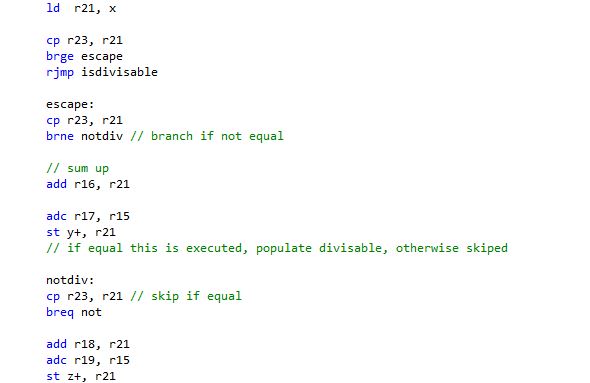
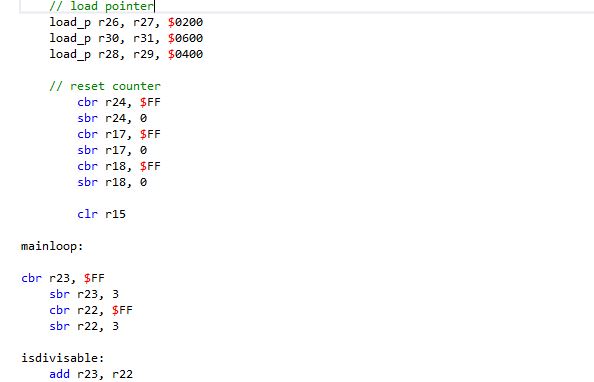


After setting R24 and R23 as counter and counter limit respectively, I stored numbers 11 to 110 starting at address 0x0200 by pointing register X to that address (screenshot above). Then I generated a loop to populate the addresses from starting address to the last (99th) address.

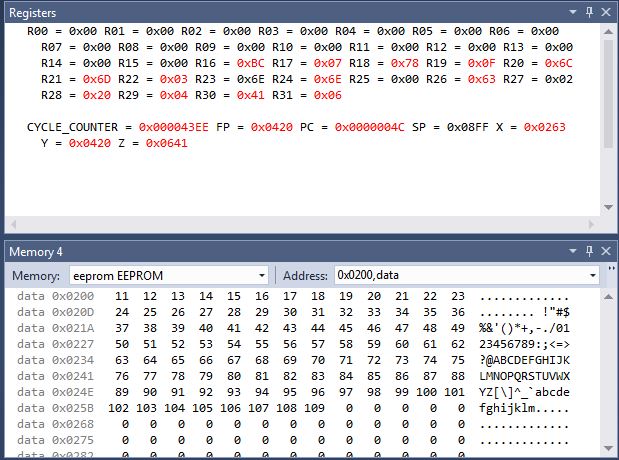


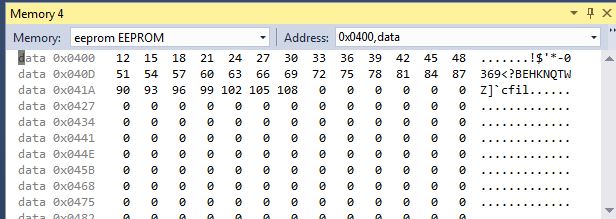
X pointer has been incremented every time that loop has been executed and each time I compare R24 and R23 to see if they are equal to skip the rjmp instruction.

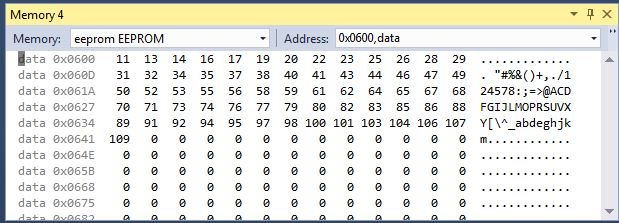
For the second part of the code, I loaded X/Y/Z pointers to required addresses. In the main loop, I generate two loops to check if the number is divisible by 3 or not. If it is, it is stored in the addresses pointing by Y pointer, otherwise is stored in the addresses pointing by Z pointer .



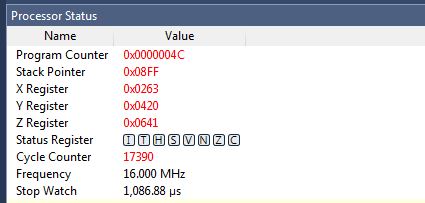
After simulating the code, results are as expected:







The following is a screenshot of Cylcle Counter and Stop Watch at frequency of 16 MHz.



Here is the complete assembly code:

;

; DA1B.asm

; Indirect addressing

;

; Created: 2.23.2019. 17:42:04

; Author : Ali Asadi

;

.macro load\_p

cbr @0, $FF

sbr @0, low(@2)

cbr @1, $FF

sbr @1, high(@2)

.endm

start:

//make X register point to address 0x0200

load\_p r26, r27, $0200

// use r23 and r24 as counter and counter limit, counter starts with 11 ends with 109

cbr r24, $FF

sbr r24, 11

cbr r23, $FF

sbr r23, 110

// loop for populating addresses starting from address 0x0200

populate:

st x+, r24 // load counter into adress pointed by pointer and increment pointer

inc r24 // increment counter

cpse r24, r23 // if counter and counter limit are equal, skip next instruction

rjmp populate //

// load pointer

load\_p r26, r27, $0200

load\_p r30, r31, $0600

load\_p r28, r29, $0400

// reset counter

cbr r24, $FF

sbr r24, 0

cbr r17, $FF

sbr r17, 0

cbr r18, $FF

sbr r18, 0

clr r15

mainloop:

cbr r23, $FF

sbr r23, 3

cbr r22, $FF

sbr r22, 3

isdivisable:

add r23, r22

ld r21, x

cp r23, r21

brge escape

rjmp isdivisable

escape:

cp r23, r21

brne notdiv // branch if not equal

// sum up

add r16, r21

adc r17, r15

st y+, r21

// if equal this is executed, populate divisable, otherwise skiped

notdiv:

cp r23, r21 // skip if equal

breq not

add r18, r21

adc r19, r15

st z+, r21

// not equal, populate not divisable

// use r20 as temporary storage in order to increment x

not:

ld r20, x+

cbr r20, $FF

sbr r20, 99 // load r20

inc r24

cpse r20, r24

rjmp mainloop

cbr r24, $FF

sbr r24, 11

cbr r23, $FF

sbr r23, 110

clr r13

ld r20, -y

ld r21, -z

sum:

cpi r20, 12

brne sumnd

add r16, r20

adc r17, r13

sumnd:

cpi r21, 11

brne skip

add r18, r21

adc r18, r13

skip:

ld r20, y

ld r21, z

inc r24

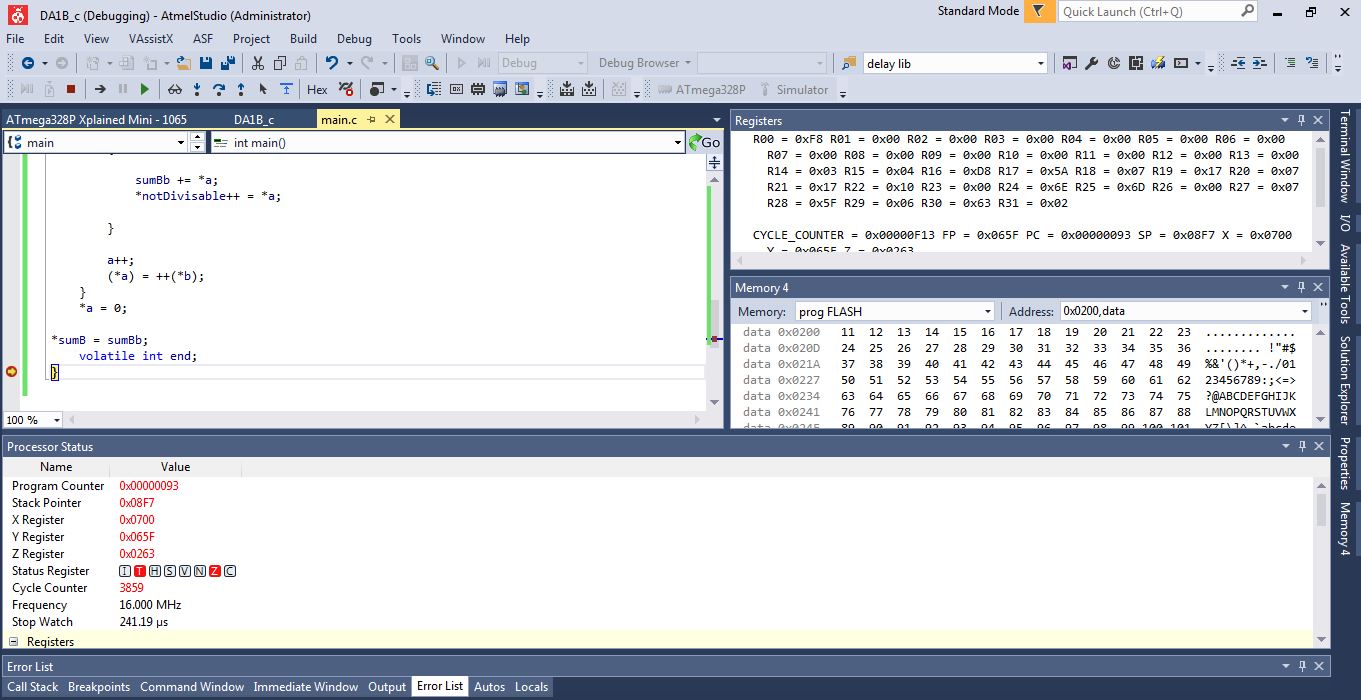
cpse r24, r23

rjmp sum //

inc r16

rjmp start

The following is a screenshot of the C code to verify the assignment.



The following is the complete C code:

/\*

\* DA1B.c

\*

\* Created: 2.24.2019. 22:12:51

\* Author : Ali Asadi

\*/

#include <avr/io.h>

int main()

{

volatile char\* b = 0x0700;

\*b = 11;

volatile char\* a = (char\*)0x0200;

volatile char\* divisable = (char\*)0x0400;

volatile char\* notDivisable = (char\*)0x0600;

volatile short int\* sumA = (short int\*)16;

volatile short int\* sumB = (short int\*)18;

int sumBb;

\*sumA = 0;

\*sumB = 0;

\*a = 11;

while(\*a <= 109)

{

if(\*a%3 == 0)

{

\*sumA += \*a;

\*divisable++ = \*a;

}

else

{

sumBb += \*a;

\*notDivisable++ = \*a;

}

a++;

(\*a) = ++(\*b);

}

\*a = 0;

\*sumB = sumBb;

volatile int end;

}