CPE301 – SPRING 2020

Design Assignment 1B

Student Name: Mateo Markovic

Student #: 2001338139

Student Email: Markom1@unlv.nevada.edu

Primary Github address: <https://github.com/mateom99/submission_da>

Directory: DesignAssignments/DA1B

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

.include <m328pdef.inc>

.equ STARTADDS = 0x0300 ; Starting address to store all 200 numbers

.equ DIV7 = 0x0500 ; Starting address to store numbers divisible by 7

.equ DIV3 = 0x0600 ; Starting address to store numbers divisible by 3

.equ DIV3\_7 = 0x0700 ; Starting address to store numbers divisible by both 3 and 7

.equ ELSE = 0x0800 ; Starting address to store all other numbers

.def JL = R22 ; Create a 'J' pointer with R5 being low bit

.def JH = R23 ; Create a 'J' pointer with R6 being high bit

.def KL = R24 ; Create a 'K' pointer with R7 being low bit

.def KH = R25 ; Create a 'K' pointer with R8 being high bit

.org 0x00

start:

CLR R0 ; Make sure R0 is set to zero for later user

CLR R1 ; Make sure Registers are initialized to zero

CLR R2 ; Make sure Registers are initialized to zero

CLR R3 ; Make sure Registers are initialized to zero

CLR R4 ; Make sure Registers are initialized to zero

CLR R5 ; Make sure Registers are initialized to zero

CLR R6 ; Make sure Registers are initialized to zero

CLR R7 ; Make sure Registers are initialized to zero

CLR R8 ; Make sure Registers are initialized to zero

CLR R9 ; Make sure Registers are initialized to zero

CLR R10 ; Make sure Registers are initialized to zero

CLR R11 ; Make sure Registers are initialized to zero

CLR R12 ; Make sure Registers are initialized to zero

CLR R13 ; Make sure Registers are initialized to zero

CLR R14 ; Make sure Registers are initialized to zero

CLR R15 ; Make sure Registers are initialized to zero

LDI XL,LOW(STARTADDS) ; Load the low bytes of the startadds address

LDI XH,HIGH(STARTADDS) ; Load the high bytes of the startadds address

LDI YL,LOW(DIV7) ; Load the low bytes of DIV7 address

LDI YH,HIGH(DIV7) ; Load the high bytes of DIV7 address

LDI ZL,LOW(DIV3) ; Load the low bytes of DIV3 address

LDI ZH,HIGH(DIV3) ; Load the high bytes of DIV3 address

LDI R16, LOW(DIV3\_7) ; Load the low bytes of DIV3\_7 address

MOV JL, R16 ; Copy low bytes of DIV3\_7 address to JL

LDI R16,HIGH(DIV3\_7) ; Load the high bytes of DIV3-7 address

MOV JH, R16 ; Copy High bytes of DIV3\_7 address to JH

LDI R16, LOW(ELSE) ; Load the low bytes of ELSE address

MOV KL, R16 ; Copy low bytes of ELSE address to KL

LDI R16,HIGH(ELSE) ; Load the high bytes of DIV3ELSE7 address

MOV KH, R16 ; Copy High bytes of ELSE address to KH

LDI R16, 26 ; starting number to store

BEGIN:

ST X+, R16

ADD R1, R16 ; Add the number to R3:R2:R1

ADC R2, R0 ; Add the number to R3:R2:R1

ADC R3, R0 ; Add the number to R3:R2:R1

MOV R17, R16 ; Copy r16 to r17 so we can work on it without changing the original

CHECK\_DIV7:

SUBI R17, 7 ; Subtract 7 from R17

CPI R17, 0

BREQ IS\_DIV7 ; if R17 is zero then it is divisible

CPI R17, 7

BRLO NOT\_DIV7 ; if R17 is lower than 7 then it won't be divisible

JMP CHECK\_DIV7 ; Keep subtracting

IS\_DIV7:

ST Y+, R16 ; Store the value in memory location Y and increment after

ADD R4, R16 ; Add the values to the 3-register pair

ADC R5, R0

ADC R6, R0

MOV R17, R16 ; Reset r17 to R16

CHECK\_DIV3\_AND\_DIV7:

SUBI R17, 3 ; Subtract 3 from R17

CPI R17, 0

BREQ IS\_DIV3\_AND\_DIV7 ; if R17 is zero then it is divisible

CPI R17, 3

BRLO FINISHED ; if R17 is lower than 3 then it won't be divisible

JMP CHECK\_DIV3\_AND\_DIV7 ; Keep subtracting

NOT\_DIV7:

MOV R17, R16 ; Reset r17 to R16

CHECK\_DIV3:

SUBI R17, 3 ; Subtract 3 from R17

CPI R17, 0

BREQ IS\_DIV3 ; if R17 is zero then it is divisible

CPI R17, 3

BRLO NOT\_DIV3\_OR\_DIV7 ; if R17 is lower than 3 then it won't be divisible

JMP CHECK\_DIV3 ; Keep subtracting

IS\_DIV3:

ST Z+, R16 ; Store the value in memory location Z and increment after

ADD R7, R16 ; Add the values to the 3-register pair

ADC R8, R0

ADC R9, R0

MOV R17, R16 ; Reset r17 to R16

JMP FINISHED ; There is nothing else to check, jump to finished

IS\_DIV3\_AND\_DIV7:

MOV R19, XH ; keep track of the original X pointer (high bytes)

MOV R18, XL ; keep track of the original X pointer (low bytes)

MOV XH, JH ; set X pointer to J pointer (high bytes)

MOV XL, JL ; set X pointer to J pointer (low bytes)

ST X, R16 ; Store the value in the memory locations set above

inc JL ; Since you can't do ADIW on R23:22, increment JL

ADC JH, R0 ; Then add any carry to JH

MOV XH, R19 ; Restore original X pointer (high bytes)

MOV XL, R18 ; Restore original X pointer (low bytes)

ADD R10, R16 ; Add the values to the 3-register pair

ADC R11, R0

ADC R12, R0

MOV R17, R16 ; Reset r17 to R16

JMP FINISHED ; There is nothing else to check, jump to finished

NOT\_DIV3\_OR\_DIV7:

MOV R19, XH ; keep track of the original X pointer (high bytes)

MOV R18, XL ; keep track of the original X pointer (low bytes)

MOV XH, KH ; set X pointer to J pointer (high bytes)

MOV XL, KL ; set X pointer to J pointer (low bytes)

ST X, R16 ; Store the value in the memory locations set above

ADIW KH:KL, 1 ; Increment the custom 'K' pointer by 1

MOV XH, R19 ; Restore original X pointer (high bytes)

MOV XL, R18 ; Restore original X pointer (low bytes)

ADD R13, R16 ; Add the values to the 3-register pair

ADC R14, R0

ADC R15, R0

MOV R17, R16 ; Reset r17 to R16

FINISHED:

INC R16 ; Increment R16 to the next value to check

CPI R16, 226 ; If we hit 226 then we are done so branch to end

BREQ end

JMP BEGIN ; otherwise return to begin to process the next number

end: JMP end ; End loop

1. **C++ Verification Code**

int main(){

cout << std::hex << std::uppercase;

cout << "BEGIN LIST ALL VALUES" << endl;

cout << "---------------------" << endl;

int sum = 0;

for (int i = 26; i < 226; i++){

cout << i << " ";

sum = sum + i;

}

cout << endl;

cout << "SUM: " << sum << endl << endl;

cout << "BEGIN LIST VALUES DIVISIBLE BY 3" << endl;

cout << "--------------------------------" << endl;

sum = 0;

for (int i = 26; i < 226; i++)

if ((i % 3) == 0){

cout << i << " ";

sum = sum + i;

}

cout << endl;

cout << "SUM: " << sum << endl << endl;

cout << "BEGIN LIST VALUES DIVISIBLE BY 7" << endl;

cout << "--------------------------------" << endl;

sum = 0;

for (int i = 26; i < 226; i++)

if ((i % 7) == 0){

cout << i << " ";

sum = sum + i;

}

cout << endl;

cout << "SUM: " << sum << endl << endl;

cout << "BEGIN LIST VALUES DIVISIBLE BY 3 AND 7" << endl;

cout << "--------------------------------------" << endl;

sum = 0;

for (int i = 26; i < 226; i++)

if ((i % 7) == 0 and (i % 3) == 0){

cout << i << " ";

sum = sum + i;

}

cout << endl;

cout << "SUM: " << sum << endl << endl;

cout << "BEGIN LIST VALUES DIVISIBLE NEITHER 3 OR 7" << endl;

cout << "------------------------------------------" << endl;

sum = 0;

for (int i = 26; i < 226; i++)

if ((i % 7) != 0 and (i % 3) != 0){

cout << i << " ";

sum = sum + i;

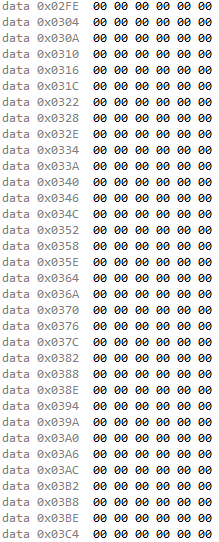
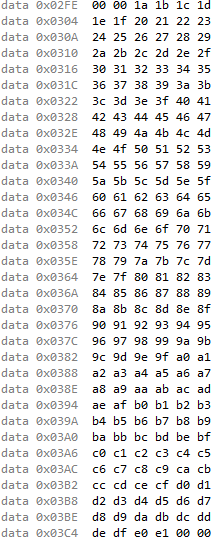
}

cout << endl;

cout << "SUM: " << sum << endl << endl;

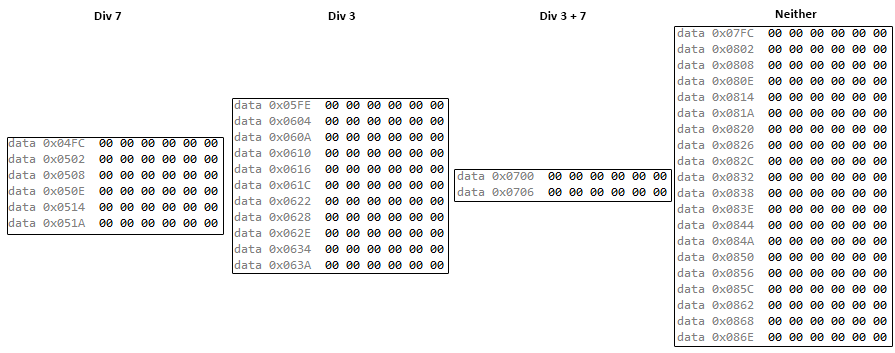
}

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

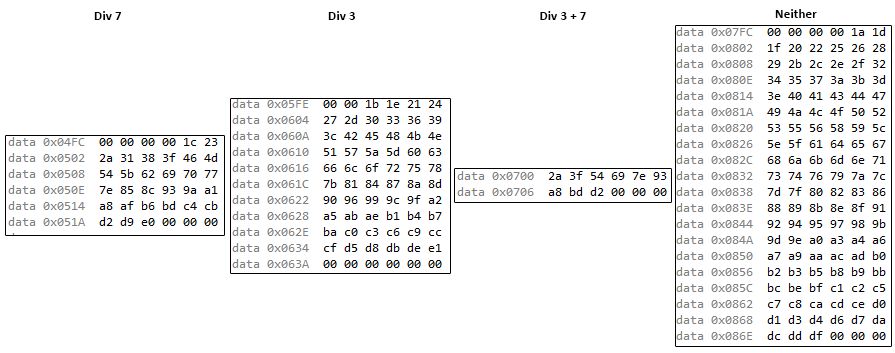
****

Memory from 0x300 before task 1

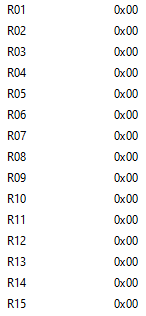
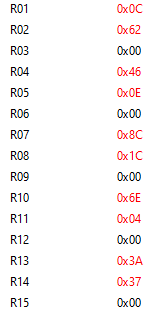
Memory from 0x300 after task 1



Memory from 0x500, 0x600, 0x700 before task 2

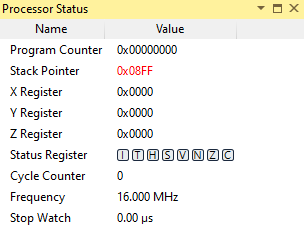
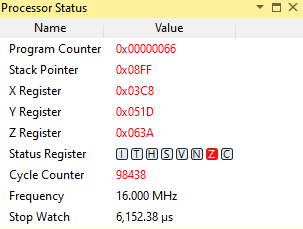


Memory from 0x500, 0x600, 0x700 after task 2



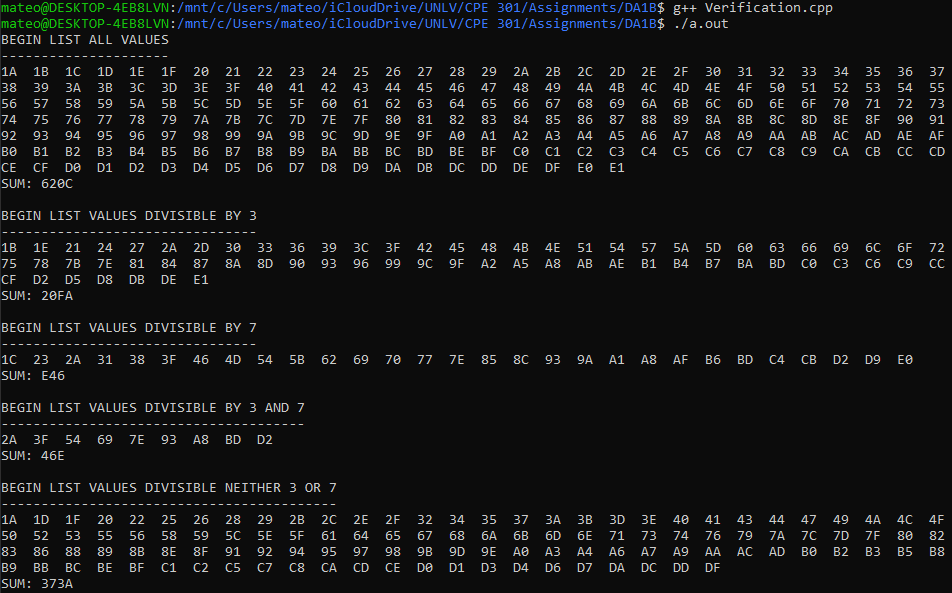
Registers R0-R15 after task 3

Registers R0-R15 before task 3



Clock Cycles after task 5

Clock Cycles before task 5



Proof using C++ program from above (matches AVR results)

1. **EXECUTION TIME**

Processing numbers from 26 to 225 using my algorithm takes 98438 cycles. This would take (98438/10x10^6) = 6.15238ms.

1. **GITHUB LINK OF THIS DA**

<https://github.com/mateom99/submission_da/tree/master/DesignAssignments/DA1B>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Mateo Markovic