Ronobir Mookherjee

CPE 301

Design Assignment 1

Code:

*/\**

*\* CpE301DesignAssignment1.asm*

*\**

*\* Author: Ronobir Mookherjee*

*\* Description: Simple program to add 20 numbers to a stack, then parse the numbers, applying computer arithmetic alogrithms in the place of available instructions. Must ulitize loops and basic RISC ISA Assembly*

*\*/*

.INCLUDE "M328PDEF.INC"*; Defines the Memory Map*

*//.ORG 0; Beginning Address*

.EQU Loop1 = 6*; First loop that pushes 20 numbers to the stack*

.MACRO STACKPOP*; Macro designed to imitate POP for stacks, but using Load. Used to move one pointer's data to another, in this case, from the Primary Stack to the secondary data pointer*

LD @0L, @1+

.ENDMACRO

.MACRO STACKPUSH*; Macro to push data to stack, using Store. Every time it is called, after the data is stored, it moves down the ram location*

ST -@1, @0

.ENDMACRO

Main: *; Main Body of progam*

PartA:*; As in the product description, must initialize the memory stack using the end of the available memory locations of the device, and then using the stack pointers SPL and SPH for the low and high set of bits of the address, assigned to one register*

LDI ZH, HIGH(RAMEND) *; initiates the stack started at the end of the available memory location*

LDI ZL, LOW(RAMEND) *;l Inititates the stack, with the low order byte, using the Z pointer register*

LDI R18, 45*; initial value being pushed to the stack that is above 30 but below 250*

STACKPUSH R18, Z *; Using the macro to store the value of R18, to the address space of Z*

LDI R21, 4 *; value used to generate new numbers*

LDI R22, 8 *; above*

LDI R19, 7 *; above*

LDI R17, Loop1 *; Counter used for main loop in random value generation*

LDI R25, 0*; checking total number of increments, initializing second counter*

INC R25 *; increments the loop everytime any arithmetic action is performed on register 18, which holds the value being pushed to the stack*

L1: STACKPUSH R18, Z*; Loop initialized with the first value pushed to the stack*

ADD R18, R21 *; Adds 2 to the value in R18, generating a new number*

STACKPUSH R18, Z *; pushing the new value to the stack*

INC R25 *; increments the arithmetic count register that must equal 20*

SUB R18, R22 *; subtracts 4 from the value in R18, generating a new value*

STACKPUSH R18, Z *; push new value to stack*

INC R25 *; above*

ADD R18, R19 *; adds 8 to R18, generating a new value to be pushed*

INC R25 *;above*

DEC R17 *; decrements the main loop stated above.*

*// Loop information*

*//The loop was initialized with one of the 20 numbers already created, so the loop can consist of 3 parts, running for 6 iterations, creating 18 new numbers, which is even. The final number is made after the loop finishes, which is below*

BRNE L1 *; Based on the previous statement, once the counter, stored in R17 is complete, 19 numbers have been pushed to the stack with one more left.*

SUB R18, R22

STACKPUSH R18, Z

INC R27 *; Making sure that 20 numbers have been pushed, using this second counter*

*// The second part of the program, designed to parse the stack and find certain criteria, in this case divisibility by a certain quotient.*

PartB:

LDI XH,HIGH(0x06FF) *; Set the MSB for the data pointer. This is used to store a 16 bit sum between X and the register combinations 20:21 for 3, and 23:24 for 5, making the ADC instruction possible*

LDI XL,LOW(0x06FF) *; Set the LSB, above*

CLR XH*; Clears data at the address space of X*

CLR XL*; above*

LDI R20, 0 *; Clearing required register*

LDI R21, 0*; Clearing required register, both are used for storing the sum of numbers divisble by 3 in the stack*

LDI R23, 0*; Cleared and initialized register used to store the sum of numbers divisible by 5 in the stack*

CLR R18*; clearing register for use*

CLR R19*; for checking 3*

CLR R16*; for checking for 5*

LDI R17, 3*; REGISTER HOLDING 3*

LDI R22, 5*; REGISTER HOLDING 5*

L2: STACKPOP X, Z*; Using the StackPOP macro defined above to get the data and place into the data pointer*

MOV R19, XL*; Copies the data of XL to the register to test for divisibility*

MOV R16, XL *; same as above*

SUBLOOP3: *// This loop proves divisibility by 3, by using an algorithm based on subtracting. If it is greater than the value 3, keep subtracting, if it is 0, it is divisible, if carry is set and it is zero, then it is not divisble. Uses the Z and C flags*

SUB R19, R17*; Subtracts the current popped number from the stack and subtracts 3 from the value*

BREQ Addition*; If Carry is not on, and Z flag is on, then it is divisble by three, move on to addition label*

BRSH SUBLOOP3*; Value of R19 still larger than 3, so continue to loop*

BRLO SUBLOOP5*; Value is not divisble, move on to next check for divisibility by 5*

*//END LOOP*

Addition: ADD R21, XL*; Value is divisble by 3, now add the value to appropriate register*

ADC R20, XH*; if value is larger than 255, carry is set, therefore add to the High Byte with Carry flag data*

SUBLOOP5:

SUB R16, R22*; Algorithm is in the same form as above, however the value is now checking for divisibility by 5*

BREQ Addition2

BRSH SUBLOOP5

BRLO EOL

*//END LOOP*

Addition2: ADD R24, XL

ADC R23, XH

EOL: DEC R25*; Both checks have completed, now decrement counter for looping*

BRNE L2*; if counter not zero, continue looping*

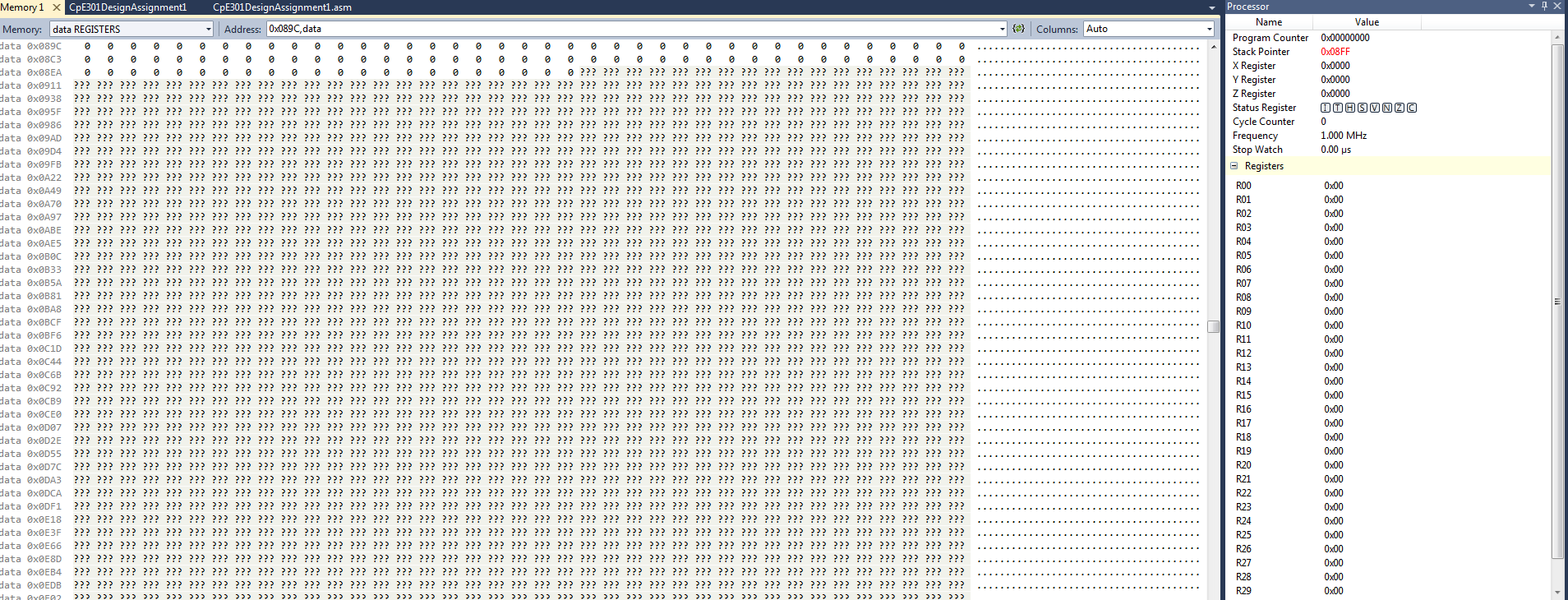
EndLoop: *; Main Program end*

sleep

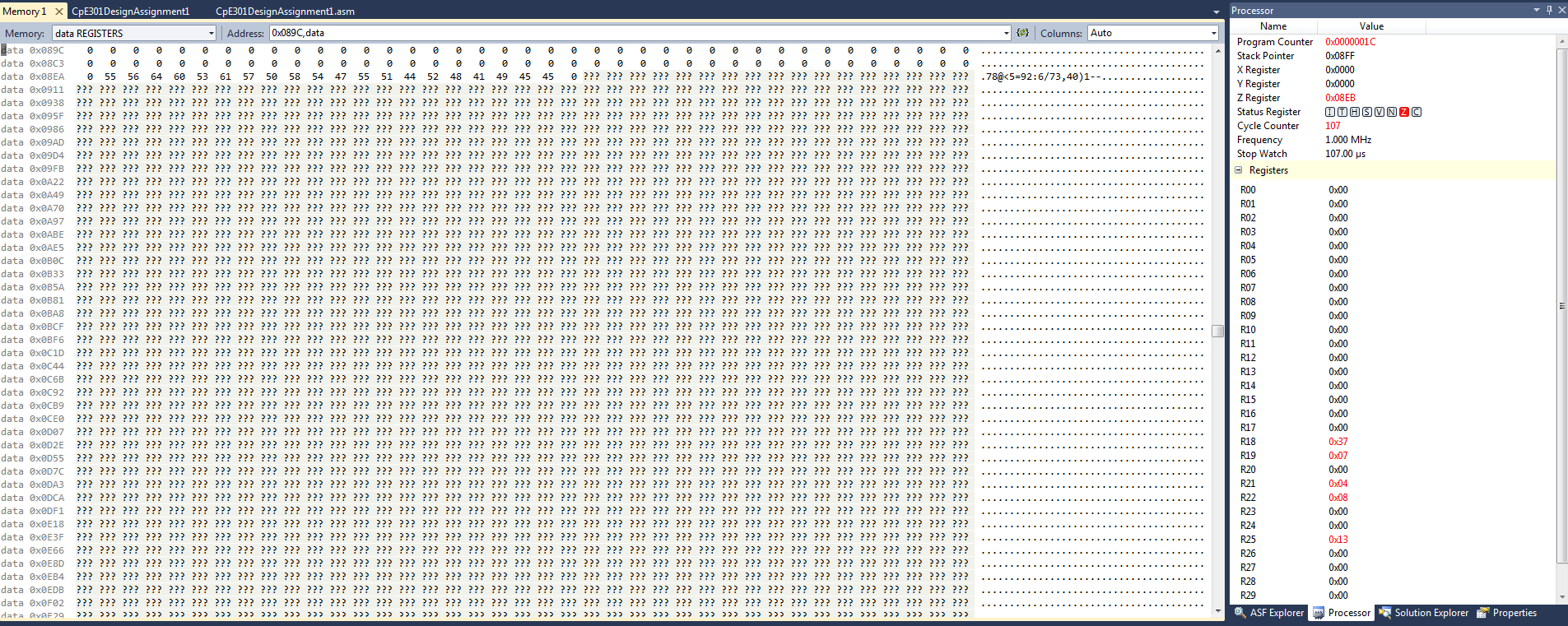
nop *; breakpoint*

rjmp EndLoop*; completed program, now in shutdown mode*

Beginning of part A:



End of first part A:



End of part B – E:

