Tenniel Takenaka-Fuller

CPE301 – SPRING 2018

Design Assignment 1

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

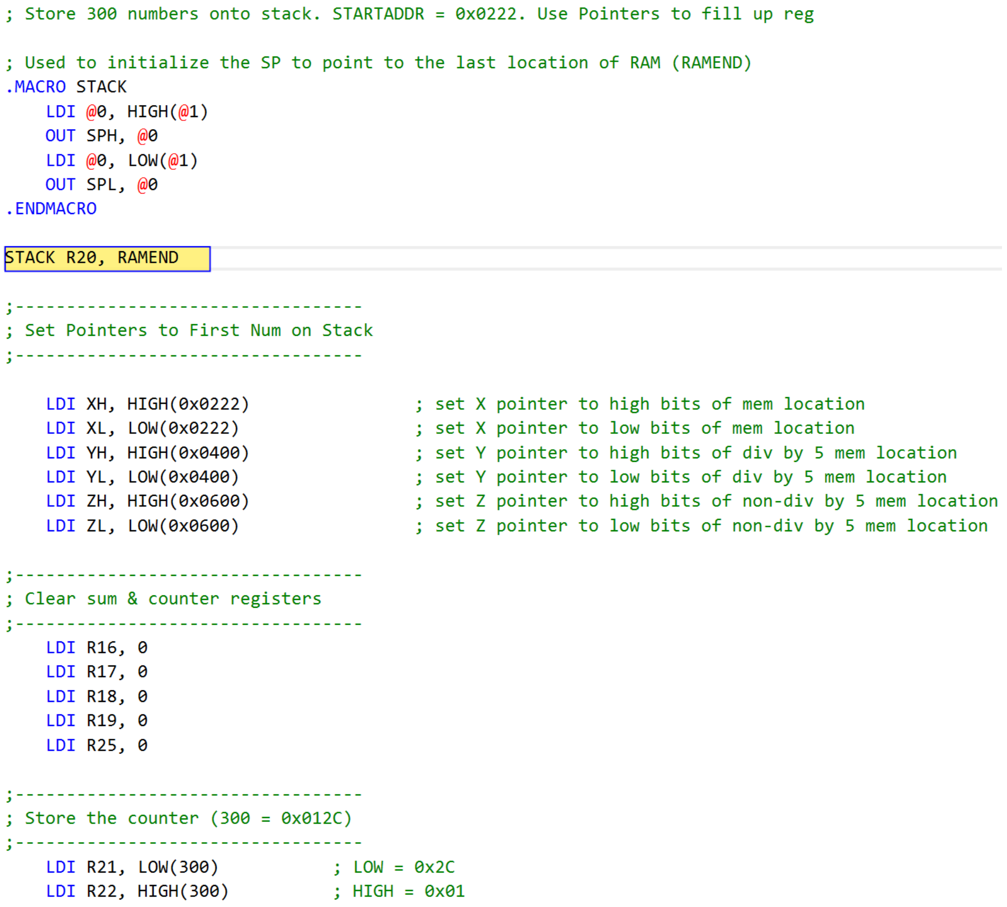
The student understands that all required components should be submitted in complete for grading of this assignment.

NO	SUBMISSION ITEM	COMPLETED (Y/N)	MARKS (/MAX)
1	COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS		
2.	INITIAL CODE OF TASK 1/A		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 2/B		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 3/C		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 4/D		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 5/E		
4.	SCHEMATICS		
5.	SCREENSHOTS OF EACH TASK OUTPUT		
5.	SCREENSHOT OF EACH DEMO		
6.	VIDEO LINKS OF EACH DEMO		
7.	GOOGLECODE LINK OF THE DA		

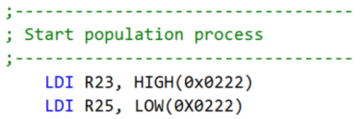
1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

Atmel Studio 7

2. INITIAL/DEVELOPED CODE OF TASK 1/A



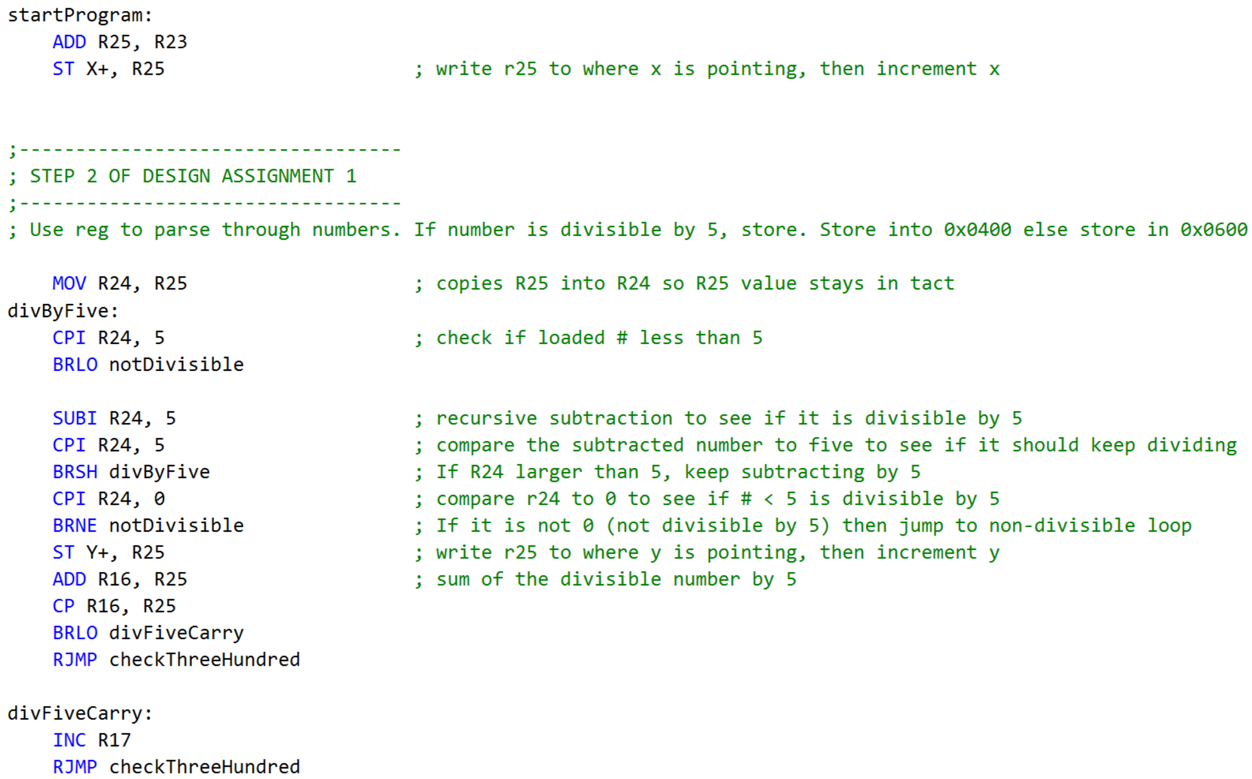
```
; Store 300 numbers onto stack. STARTADDR = 0x0222. Use Pointers to fill up reg

; Used to initialize the SP to point to the last location of RAM (RAMEND)
.MACRO STACK
    LDI @0, HIGH(@1)
    OUT SPH, @0
    LDI @0, LOW(@1)
    OUT SPL, @0
.ENDMACRO

STACK R20, RAMEND

;----------------------------------
; Set Pointers to First Num on Stack
;----------------------------------

    LDI XH, HIGH(0x0222)                ; set X pointer to high bits of mem location
    LDI XL, LOW(0x0222)                 ; set X pointer to low bits of mem location
    LDI YH, HIGH(0x0400)                ; set Y pointer to high bits of div by 5 mem location
    LDI YL, LOW(0x0400)                 ; set Y pointer to low bits of div by 5 mem location
    LDI ZH, HIGH(0x0600)                ; set Z pointer to high bits of non-div by 5 mem location
    LDI ZL, LOW(0x0600)                 ; set Z pointer to low bits of non-div by 5 mem location

;----------------------------------
; Clear sum & counter registers
;----------------------------------
    LDI R16, 0
    LDI R17, 0
    LDI R18, 0
    LDI R19, 0
    LDI R25, 0

;----------------------------------
; Store the counter (300 = 0x012C)
;----------------------------------
    LDI R21, LOW(300)           ; LOW = 0x2C
    LDI R22, HIGH(300)          ; HIGH = 0x01
```



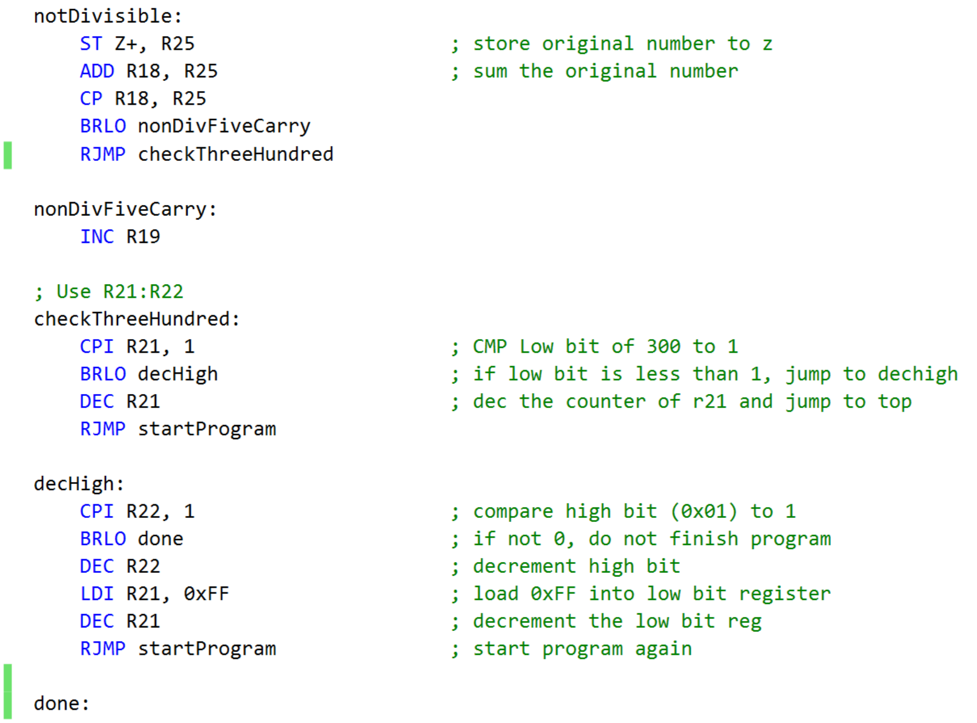
```
;----------------------------------
; Start population process
;----------------------------------
    LDI R23, HIGH(0x0222)
    LDI R25, LOW(0X0222)
```

3. CODE OF TASK 1/B AND 1/C COMBINED INTO ONE SECTION

Start program is the continuation of the population process.

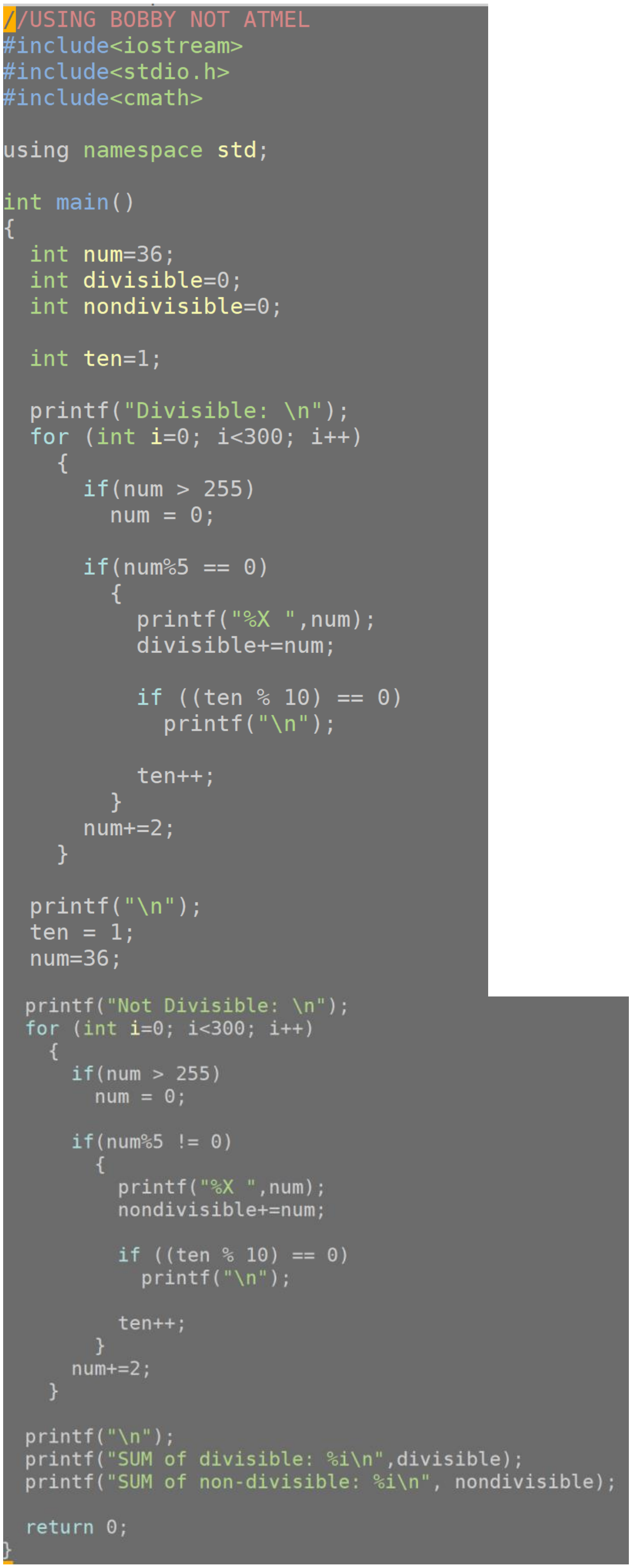


```
startProgram:
    ADD R25, R23
    ST X+, R25                    ; write r25 to where x is pointing, then increment x


;----------------------------------
; STEP 2 OF DESIGN ASSIGNMENT 1
;----------------------------------
; Use reg to parse through numbers. If number is divisible by 5, store. Store into 0x0400 else store in 0x0600

    MOV R24, R25                  ; copies R25 into R24 so R25 value stays in tact
divByFive:
    CPI R24, 5                    ; check if loaded # less than 5
    BRLO notDivisible

    SUBI R24, 5                   ; recursive subtraction to see if it is divisible by 5
    CPI R24, 5                    ; compare the subtracted number to five to see if it should keep dividing
    BRSH divByFive                ; If R24 larger than 5, keep subtracting by 5
    CPI R24, 0                    ; compare r24 to 0 to see if # < 5 is divisible by 5
    BRNE notDivisible             ; If it is not 0 (not divisible by 5) then jump to non-divisible loop
    ST Y+, R25                    ; write r25 to where y is pointing, then increment y
    ADD R16, R25                  ; sum of the divisible number by 5
    CP R16, R25
    BRLO divFiveCarry
    RJMP checkThreeHundred

divFiveCarry:
    INC R17
    RJMP checkThreeHundred
```



```
notDivisible:
    ST Z+, R25                    ; store original number to z
    ADD R18, R25                  ; sum the original number
    CP R18, R25
    BRLO nonDivFiveCarry
    RJMP checkThreeHundred

nonDivFiveCarry:
    INC R19

; Use R21:R22
checkThreeHundred:
    CPI R21, 1                    ; CMP Low bit of 300 to 1
    BRLO decHigh                  ; if low bit is less than 1, jump to dechigh
    DEC R21                       ; dec the counter of r21 and jump to top
    RJMP startProgram

decHigh:
    CPI R22, 1                    ; compare high bit (0x01) to 1
    BRLO done                     ; if not 0, do not finish program
    DEC R22                       ; decrement high bit
    LDI R21, 0xFF                 ; load 0xFF into low bit register
    DEC R21                       ; decrement the low bit reg
    RJMP startProgram             ; start program again

done:
```

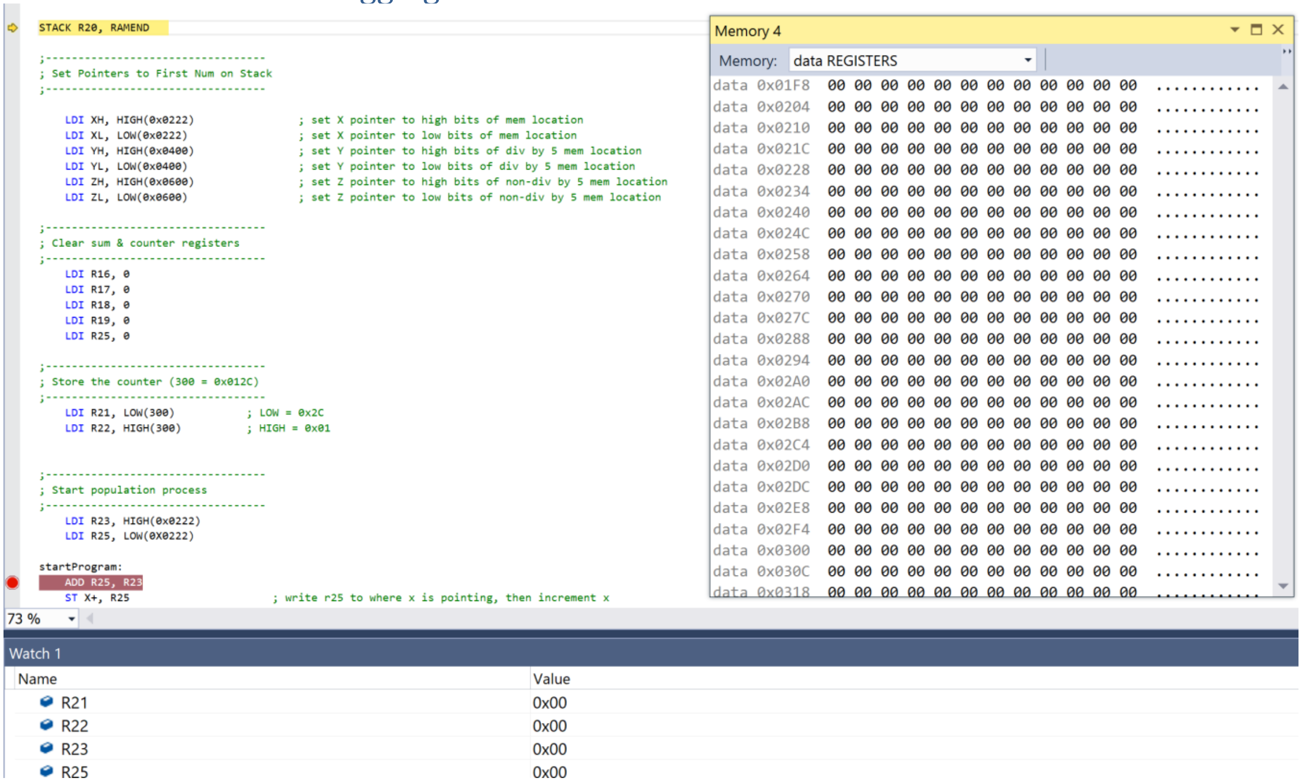
2. CODE OF TASK 1/D



```
//USING BOBBY NOT ATMEL
#include<iostream>
#include<stdio.h>
#include<cmath>

using namespace std;

int main()
{
  int num=36;
  int divisible=0;
  int nondivisible=0;

  int ten=1;

  printf("Divisible: \n");
  for (int i=0; i<300; i++)
    {
      if(num > 255)
        num = 0;

      if(num%5 == 0)
        {
          printf("%X ",num);
          divisible+=num;

          if ((ten % 10) == 0)
            printf("\n");

          ten++;
        }
      num+=2;
    }

  printf("\n");
  ten = 1;
  num=36;

  printf("Not Divisible: \n");
  for (int i=0; i<300; i++)
    {
      if(num > 255)
        num = 0;

      if(num%5 != 0)
        {
          printf("%X ",num);
          nondivisible+=num;

          if ((ten % 10) == 0)
            printf("\n");

          ten++;
        }
      num+=2;
    }

  printf("\n");
  printf("SUM of divisible: %i\n",divisible);
  printf("SUM of non-divisible: %i\n", nondivisible);

  return 0;
}
```

3. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

Task 1: Before Debugging



```
STACK R20, RAMEND

;----------------------------------
; Set Pointers to First Num on Stack
;----------------------------------

    LDI XH, HIGH(0x0222)               ; set X pointer to high bits of mem location
    LDI XL, LOW(0x0222)                ; set X pointer to low bits of mem location
    LDI YH, HIGH(0x0400)               ; set Y pointer to high bits of div by 5 mem location
    LDI YL, LOW(0x0400)                ; set Y pointer to low bits of div by 5 mem location
    LDI ZH, HIGH(0x0600)               ; set Z pointer to high bits of non-div by 5 mem location
    LDI ZL, LOW(0x0600)                ; set Z pointer to low bits of non-div by 5 mem location

;----------------------------------
; Clear sum & counter registers
;----------------------------------
    LDI R16, 0
    LDI R17, 0
    LDI R18, 0
    LDI R19, 0
    LDI R25, 0

;----------------------------------
; Store the counter (300 = 0x012C)
;----------------------------------
    LDI R21, LOW(300)           ; LOW = 0x2C
    LDI R22, HIGH(300)          ; HIGH = 0x01


;----------------------------------
; Start population process
;----------------------------------
    LDI R23, HIGH(0x0222)
    LDI R25, LOW(0X0222)

startProgram:
    ADD R25, R23
    ST X+, R25                        ; write r25 to where x is pointing, then increment x
```

73 %

Memory 4

Memory: data REGISTERS

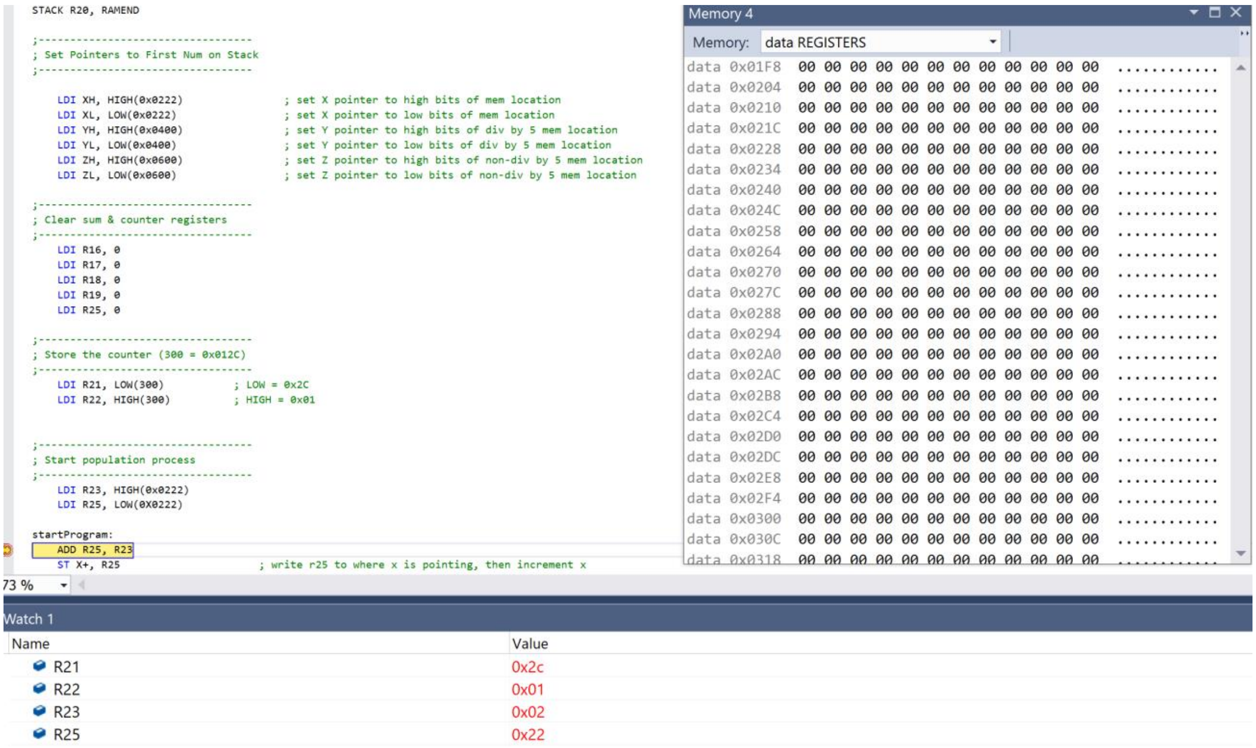
```
data 0x01F8  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0204  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0210  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x021C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0228  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0234  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0240  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x024C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0258  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0264  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0270  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x027C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0288  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0294  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02A0  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02AC  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02B8  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02C4  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02D0  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02DC  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02E8  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02F4  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0300  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x030C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0318  00 00 00 00 00 00 00 00 00 00 00 00  ............
```

Watch 1

Name	Value
R21	0x00
R22	0x00
R23	0x00
R25	0x00

Task 1: After Debugging

Values in register R21, R22, R23, and R25 have changed and loaded values from 0’s.



```
STACK R20, RAMEND

;----------------------------------
; Set Pointers to First Num on Stack
;----------------------------------

    LDI XH, HIGH(0x0222)               ; set X pointer to high bits of mem location
    LDI XL, LOW(0x0222)                ; set X pointer to low bits of mem location
    LDI YH, HIGH(0x0400)               ; set Y pointer to high bits of div by 5 mem location
    LDI YL, LOW(0x0400)                ; set Y pointer to low bits of div by 5 mem location
    LDI ZH, HIGH(0x0600)               ; set Z pointer to high bits of non-div by 5 mem location
    LDI ZL, LOW(0x0600)                ; set Z pointer to low bits of non-div by 5 mem location

;----------------------------------
; Clear sum & counter registers
;----------------------------------
    LDI R16, 0
    LDI R17, 0
    LDI R18, 0
    LDI R19, 0
    LDI R25, 0

;----------------------------------
; Store the counter (300 = 0x012C)
;----------------------------------
    LDI R21, LOW(300)           ; LOW = 0x2C
    LDI R22, HIGH(300)          ; HIGH = 0x01


;----------------------------------
; Start population process
;----------------------------------
    LDI R23, HIGH(0x0222)
    LDI R25, LOW(0X0222)

startProgram:
    ADD R25, R23
    ST X+, R25                        ; write r25 to where x is pointing, then increment x
```

73 %

Memory 4

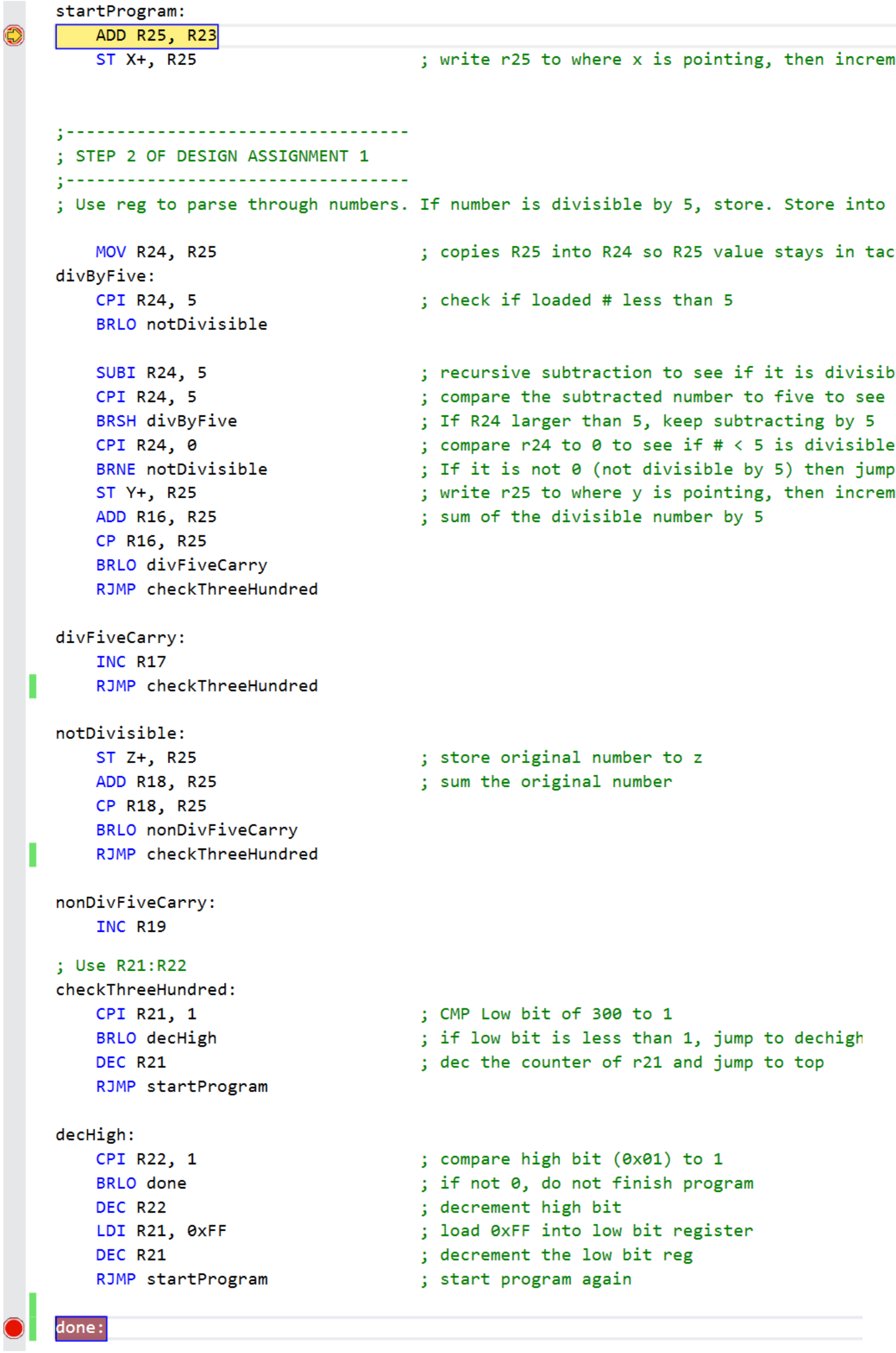
Memory: data REGISTERS

```
data 0x01F8  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0204  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0210  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x021C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0228  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0234  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0240  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x024C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0258  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0264  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0270  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x027C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0288  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0294  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02A0  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02AC  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02B8  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02C4  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02D0  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02DC  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02E8  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x02F4  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0300  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x030C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0318  00 00 00 00 00 00 00 00 00 00 00 00  ............
```

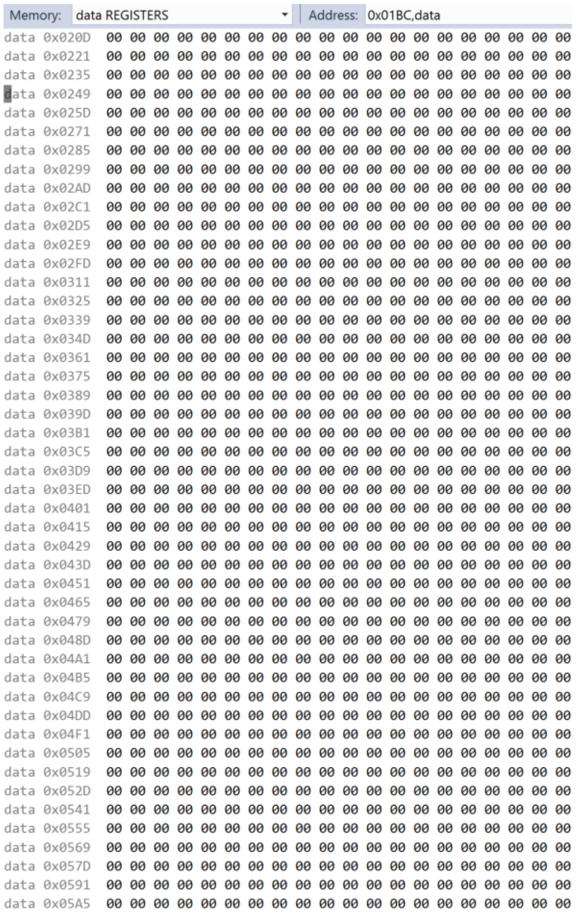
Watch 1

Name	Value
R21	0x2c
R22	0x01
R23	0x02
R25	0x22

Task 1B and 1C: Before Debugging



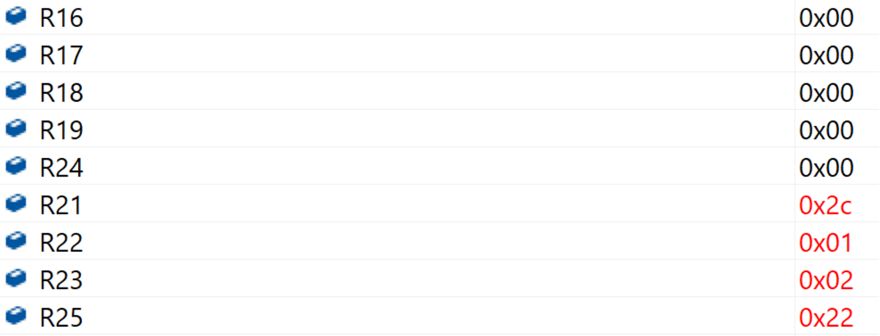
```
startProgram:
    ADD R25, R23
    ST X+, R25                      ; write r25 to where x is pointing, then increment x


;---------------------------------
; STEP 2 OF DESIGN ASSIGNMENT 1
;---------------------------------
; Use reg to parse through numbers. If number is divisible by 5, store. Store into 0x0400 else store in 0x0600

    MOV R24, R25                    ; copies R25 into R24 so R25 value stays in tact
divByFive:
    CPI R24, 5                      ; check if loaded # less than 5
    BRLO notDivisible

    SUBI R24, 5                     ; recursive subtraction to see if it is divisible by 5
    CPI R24, 5                      ; compare the subtracted number to five to see if it should keep dividing
    BRSH divByFive                  ; If R24 larger than 5, keep subtracting by 5
    CPI R24, 0                      ; compare r24 to 0 to see if # < 5 is divisible by 5
    BRNE notDivisible               ; If it is not 0 (not divisible by 5) then jump to non-divisible loop
    ST Y+, R25                      ; write r25 to where y is pointing, then increment y
    ADD R16, R25                    ; sum of the divisible number by 5
    CP R16, R25
    BRLO divFiveCarry
    RJMP checkThreeHundred

divFiveCarry:
    INC R17
    RJMP checkThreeHundred

notDivisible:
    ST Z+, R25                      ; store original number to z
    ADD R18, R25                    ; sum the original number
    CP R18, R25
    BRLO nonDivFiveCarry
    RJMP checkThreeHundred

nonDivFiveCarry:
    INC R19

; Use R21:R22
checkThreeHundred:
    CPI R21, 1                      ; CMP Low bit of 300 to 1
    BRLO decHigh                    ; if low bit is less than 1, jump to dechigh
    DEC R21                         ; dec the counter of r21 and jump to top
    RJMP startProgram

decHigh:
    CPI R22, 1                      ; compare high bit (0x01) to 1
    BRLO done                       ; if not 0, do not finish program
    DEC R22                         ; decrement high bit
    LDI R21, 0xFF                   ; load 0xFF into low bit register
    DEC R21                         ; decrement the low bit reg
    RJMP startProgram               ; start program again

done:
```



Memory: data REGISTERS | Address: 0x01BC,data

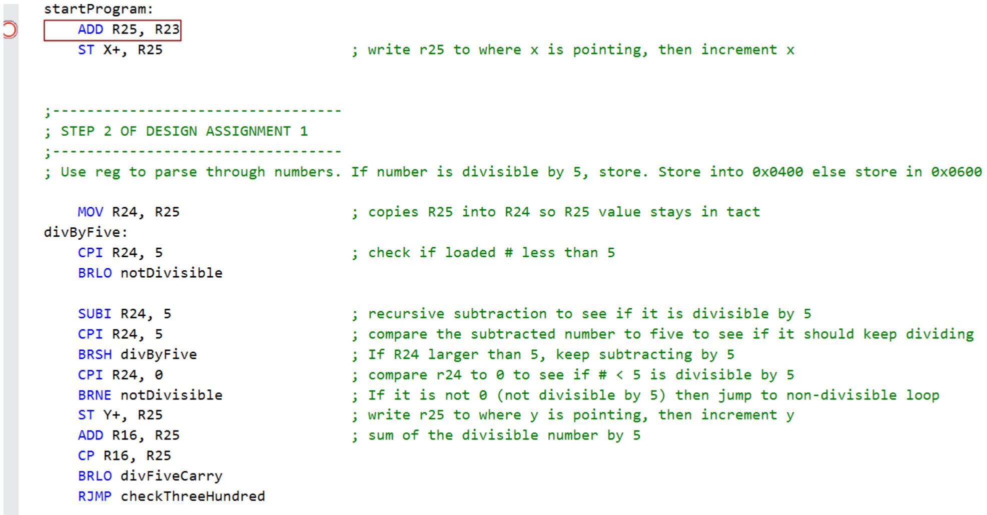
```
data 0x020D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0221  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0235  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0249  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x025D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0271  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0285  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0299  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x02AD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x02C1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x02D5  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x02E9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x02FD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0311  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0325  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0339  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x034D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0361  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0375  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0389  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x039D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x03B1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x03C5  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x03D9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x03ED  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0401  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0415  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0429  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x043D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0451  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0465  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0479  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x048D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x04A1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x04B5  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x04C9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x04DD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x04F1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0505  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0519  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x052D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0541  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0555  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0569  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x057D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0591  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x05A5  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Register values before debugging

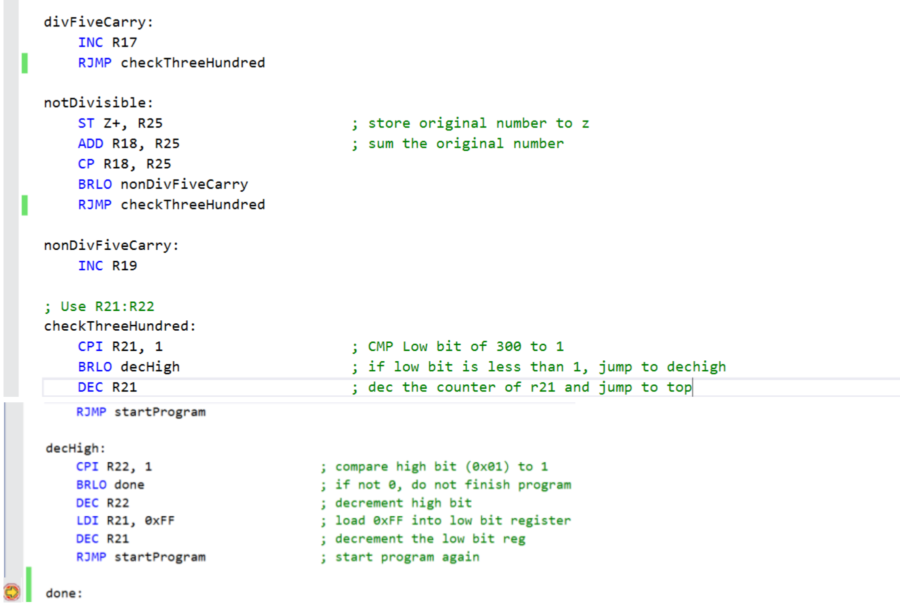


R16	0x00
R17	0x00
R18	0x00
R19	0x00
R24	0x00
R21	0x2c
R22	0x01
R23	0x02
R25	0x22

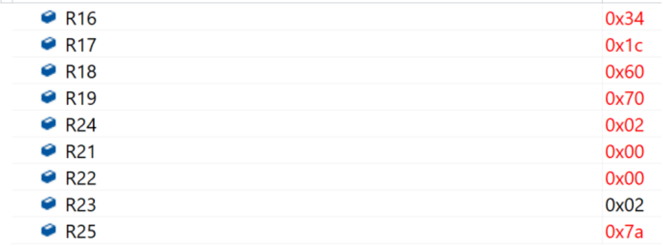
Task 1B and 1C: After Debugging



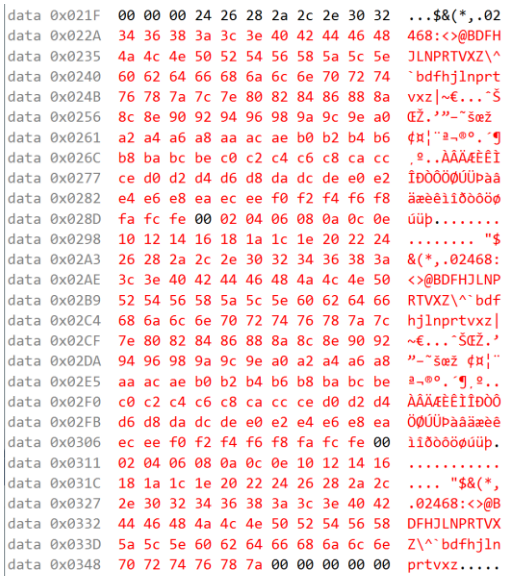
```
startProgram:
    ADD R25, R23
    ST X+, R25                    ; write r25 to where x is pointing, then increment x


;---------------------------------
; STEP 2 OF DESIGN ASSIGNMENT 1
;---------------------------------
; Use reg to parse through numbers. If number is divisible by 5, store. Store into 0x0400 else store in 0x0600

    MOV R24, R25                  ; copies R25 into R24 so R25 value stays in tact
divByFive:
    CPI R24, 5                    ; check if loaded # less than 5
    BRLO notDivisible

    SUBI R24, 5                   ; recursive subtraction to see if it is divisible by 5
    CPI R24, 5                    ; compare the subtracted number to five to see if it should keep dividing
    BRSH divByFive                ; If R24 larger than 5, keep subtracting by 5
    CPI R24, 0                    ; compare r24 to 0 to see if # < 5 is divisible by 5
    BRNE notDivisible             ; If it is not 0 (not divisible by 5) then jump to non-divisible loop
    ST Y+, R25                    ; write r25 to where y is pointing, then increment y
    ADD R16, R25                  ; sum of the divisible number by 5
    CP R16, R25
    BRLO divFiveCarry
    RJMP checkThreeHundred
```



```
divFiveCarry:
    INC R17
    RJMP checkThreeHundred

notDivisible:
    ST Z+, R25                    ; store original number to z
    ADD R18, R25                  ; sum the original number
    CP R18, R25
    BRLO nonDivFiveCarry
    RJMP checkThreeHundred

nonDivFiveCarry:
    INC R19

; Use R21:R22
checkThreeHundred:
    CPI R21, 1                    ; CMP Low bit of 300 to 1
    BRLO decHigh                  ; if low bit is less than 1, jump to dechigh
    DEC R21                       ; dec the counter of r21 and jump to top
    RJMP startProgram

decHigh:
    CPI R22, 1                    ; compare high bit (0x01) to 1
    BRLO done                     ; if not 0, do not finish program
    DEC R22                       ; decrement high bit
    LDI R21, 0xFF                 ; load 0xFF into low bit register
    DEC R21                       ; decrement the low bit reg
    RJMP startProgram             ; start program again

done:
```

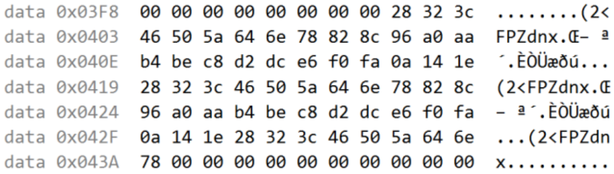


```
R16    0x34
R17    0x1c
R18    0x60
R19    0x70
R24    0x02
R21    0x00
R22    0x00
R23    0x02
R25    0x7a
```



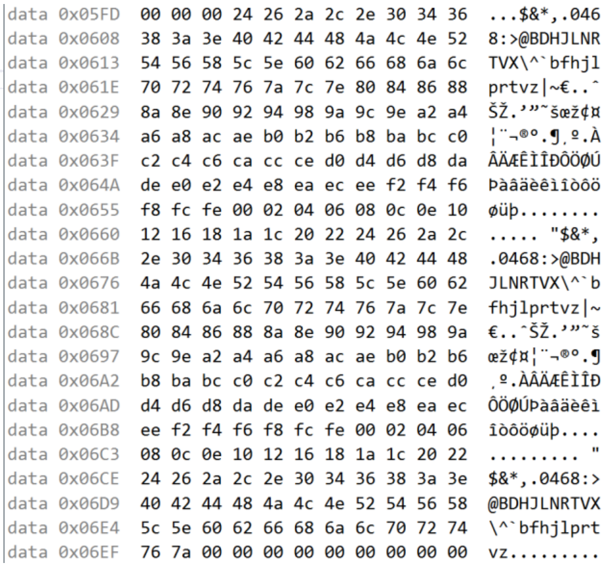
```
data 0x021F  00 00 00 24 26 28 2a 2c 2e 30 32  ...$&(*,.02
data 0x022A  34 36 38 3a 3c 3e 40 42 44 46 48  468:<>@BDFH
data 0x0235  4a 4c 4e 50 52 54 56 58 5a 5c 5e  JLNPRTVXZ\^
data 0x0240  60 62 64 66 68 6a 6c 6e 70 72 74  `bdfhjlnprt
data 0x024B  76 78 7a 7c 7e 80 82 84 86 88 8a  vxz|~€...ˆŠ
data 0x0256  8c 8e 90 92 94 96 98 9a 9c 9e a0  ŒŽ.’”–˜šœž 
data 0x0261  a2 a4 a6 a8 aa ac ae b0 b2 b4 b6  ¢¤¦¨ª¬®°.´¶
data 0x026C  b8 ba bc be c0 c2 c4 c6 c8 ca cc  ¸º..ÀÂÄÆÈÊÌ
data 0x0277  ce d0 d2 d4 d6 d8 da dc de e0 e2  ÎÐÒÔÖØÚÜÞàâ
data 0x0282  e4 e6 e8 ea ec ee f0 f2 f4 f6 f8  äæèêìîðòôöø
data 0x028D  fa fc fe 00 02 04 06 08 0a 0c 0e  úüþ........
data 0x0298  10 12 14 16 18 1a 1c 1e 20 22 24  ........ "$
data 0x02A3  26 28 2a 2c 2e 30 32 34 36 38 3a  &(*,.02468:
data 0x02AE  3c 3e 40 42 44 46 48 4a 4c 4e 50  <>@BDFHJLNP
data 0x02B9  52 54 56 58 5a 5c 5e 60 62 64 66  RTVXZ\^`bdf
data 0x02C4  68 6a 6c 6e 70 72 74 76 78 7a 7c  hjlnprtvxz|
data 0x02CF  7e 80 82 84 86 88 8a 8c 8e 90 92  ~€...ˆŠŒŽ.’
data 0x02DA  94 96 98 9a 9c 9e a0 a2 a4 a6 a8  ”–˜šœž ¢¤¦¨
data 0x02E5  aa ac ae b0 b2 b4 b6 b8 ba bc be  ª¬®°.´¶¸º..
data 0x02F0  c0 c2 c4 c6 c8 ca cc ce d0 d2 d4  ÀÂÄÆÈÊÌÎÐÒÔ
data 0x02FB  d6 d8 da dc de e0 e2 e4 e6 e8 ea  ÖØÚÜÞàâäæèê
data 0x0306  ec ee f0 f2 f4 f6 f8 fa fc fe 00  ìîðòôöøúüþ.
data 0x0311  02 04 06 08 0a 0c 0e 10 12 14 16  ...........
data 0x031C  18 1a 1c 1e 20 22 24 26 28 2a 2c  .... "$&(*,
data 0x0327  2e 30 32 34 36 38 3a 3c 3e 40 42  .02468:<>@B
data 0x0332  44 46 48 4a 4c 4e 50 52 54 56 58  DFHJLNPRTVX
data 0x033D  5a 5c 5e 60 62 64 66 68 6a 6c 6e  Z\^`bdfhjln
data 0x0348  70 72 74 76 78 7a 00 00 00 00 00  prtvxz.....
```

Figure 1 X-Pointer Array (300 numbers)



```
data 0x03F8  00 00 00 00 00 00 00 00 28 32 3c  ........(2<
data 0x0403  46 50 5a 64 6e 78 82 8c 96 a0 aa  FPZdnx.Œ– ª
data 0x040E  b4 be c8 d2 dc e6 f0 fa 0a 14 1e  ´.ÈÒÜæðú...
data 0x0419  28 32 3c 46 50 5a 64 6e 78 82 8c  (2<FPZdnx.Œ
data 0x0424  96 a0 aa b4 be c8 d2 dc e6 f0 fa  – ª´.ÈÒÜæðú
data 0x042F  0a 14 1e 28 32 3c 46 50 5a 64 6e  ...(2<FPZdn
data 0x043A  78 00 00 00 00 00 00 00 00 00 00  x..........
```

Figure 2 Y-Pointer Array (Divisible Numbers)

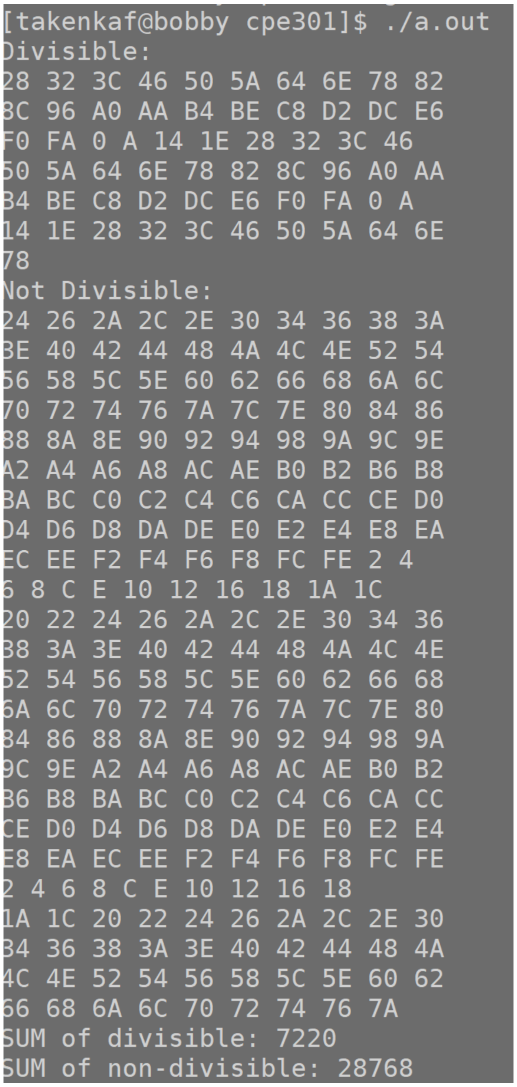


```
data 0x05FD  00 00 00 24 26 2a 2c 2e 30 34 36  ...$&*,.046
data 0x0608  38 3a 3e 40 42 44 48 4a 4c 4e 52  8:>@BDHJLNR
data 0x0613  54 56 58 5c 5e 60 62 66 68 6a 6c  TVX\^`bfhjl
data 0x061E  70 72 74 76 7a 7c 7e 80 84 86 88  prtvz|~€..ˆ
data 0x0629  8a 8e 90 92 94 98 9a 9c 9e a2 a4  ŠŽ.’”˜šœž¢¤
data 0x0634  a6 a8 ac ae b0 b2 b6 b8 ba bc c0  ¦¨¬®°.¶¸º.À
data 0x063F  c2 c4 c6 ca cc ce d0 d4 d6 d8 da  ÂÄÆÊÌÎÐÔÖØÚ
data 0x064A  de e0 e2 e4 e8 ea ec ee f2 f4 f6  Þàâäèêìîòôö
data 0x0655  f8 fc fe 00 02 04 06 08 0c 0e 10  øüþ........
data 0x0660  12 16 18 1a 1c 20 22 24 26 2a 2c  ..... "$&*,
data 0x066B  2e 30 34 36 38 3a 3e 40 42 44 48  .0468:>@BDH
data 0x0676  4a 4c 4e 52 54 56 58 5c 5e 60 62  JLNRTVX\^`b
data 0x0681  66 68 6a 6c 70 72 74 76 7a 7c 7e  fhjlprtvz|~
data 0x068C  80 84 86 88 8a 8e 90 92 94 98 9a  €..ˆŠŽ.’”˜š
data 0x0697  9c 9e a2 a4 a6 a8 ac ae b0 b2 b6  œž¢¤¦¨¬®°.¶
data 0x06A2  b8 ba bc c0 c2 c4 c6 ca cc ce d0  ¸º.ÀÂÄÆÊÌÎÐ
data 0x06AD  d4 d6 d8 da de e0 e2 e4 e8 ea ec  ÔÖØÚÞàâäèêì
data 0x06B8  ee f2 f4 f6 f8 fc fe 00 02 04 06  îòôöøüþ....
data 0x06C3  08 0c 0e 10 12 16 18 1a 1c 20 22  ......... "
data 0x06CE  24 26 2a 2c 2e 30 34 36 38 3a 3e  $&*,.0468:>
data 0x06D9  40 42 44 48 4a 4c 4e 52 54 56 58  @BDHJLNRTVX
data 0x06E4  5c 5e 60 62 66 68 6a 6c 70 72 74  \^`bfhjlprt
data 0x06EF  76 7a 00 00 00 00 00 00 00 00 00  vz.........
```

Figure 3 Z-Pointer Array (Non-Divisible #s)

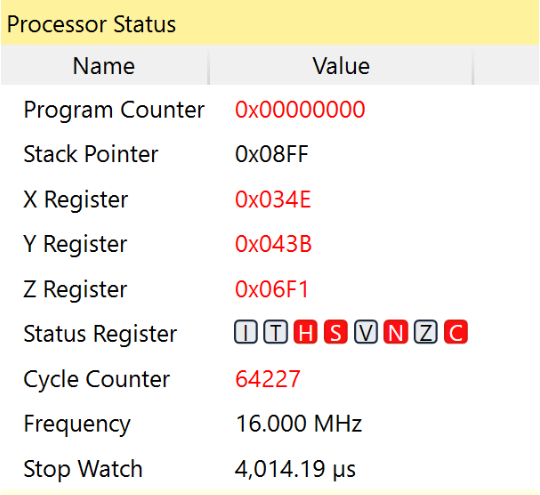
The registers R22:R21 registers are zero-d out since I placed 300 in these 2 registers and decremented both for the checkThreeHundred loop. The divisible sum is stored in R17:R16 as 0x1C34 (7,220 in decimal). The non-divisible sum is stored in R19:R18 as 0x7060 (28,768 in decimal). The arrays that x, y, and z are pointing to are shown in the Memory windows above.

Task 1D: After Debugging



```
[takenkaf@bobby cpe301]$ ./a.out
Divisible:
28 32 3C 46 50 5A 64 6E 78 82
8C 96 A0 AA B4 BE C8 D2 DC E6
F0 FA 0 A 14 1E 28 32 3C 46
50 5A 64 6E 78 82 8C 96 A0 AA
B4 BE C8 D2 DC E6 F0 FA 0 A
14 1E 28 32 3C 46 50 5A 64 6E
78
Not Divisible:
24 26 2A 2C 2E 30 34 36 38 3A
3E 40 42 44 48 4A 4C 4E 52 54
56 58 5C 5E 60 62 66 68 6A 6C
70 72 74 76 7A 7C 7E 80 84 86
88 8A 8E 90 92 94 98 9A 9C 9E
A2 A4 A6 A8 AC AE B0 B2 B6 B8
BA BC C0 C2 C4 C6 CA CC CE D0
D4 D6 D8 DA DE E0 E2 E4 E8 EA
EC EE F2 F4 F6 F8 FC FE 2 4
6 8 C E 10 12 16 18 1A 1C
20 22 24 26 2A 2C 2E 30 34 36
38 3A 3E 40 42 44 48 4A 4C 4E
52 54 56 58 5C 5E 60 62 66 68
6A 6C 70 72 74 76 7A 7C 7E 80
84 86 88 8A 8E 90 92 94 98 9A
9C 9E A2 A4 A6 A8 AC AE B0 B2
B6 B8 BA BC C0 C2 C4 C6 CA CC
CE D0 D4 D6 D8 DA DE E0 E2 E4
E8 EA EC EE F2 F4 F6 F8 FC FE
2 4 6 8 C E 10 12 16 18
1A 1C 20 22 24 26 2A 2C 2E 30
34 36 38 3A 3E 40 42 44 48 4A
4C 4E 52 54 56 58 5C 5E 60 62
66 68 6A 6C 70 72 74 76 7A
SUM of divisible: 7220
SUM of non-divisible: 28768
```

4. SCREENSHOT OF 1E



Processor Status

Name	Value
Program Counter	0x00000000
Stack Pointer	0x08FF
X Register	0x034E
Y Register	0x043B
Z Register	0x06F1
Status Register	I T H S V N Z C
Cycle Counter	64227
Frequency	16.000 MHz
Stop Watch	4,014.19 µs

Figure 4 At 16MHz, execution done in 4,014.19 microseconds

5. GITHUB LINK OF THIS DA

https://github.com/TennielTakenaka/DA1

Student Academic Misconduct Policy
http://studentconduct.unlv.edu/misconduct/policy.html

"*This assignment submission is my own, original work*".
Tenniel Takenaka-Fuller