

Since it has 16 bit to store the sum you need to get the 16 bit value.

Design Assignment 01

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		

Task 1/A: Store 300 numbers starting from the STARTADD=0x0222 location. Populate the value of the memory location by adding high(STARTADD) and low(STARTADD) . Use the X/Y/Z registers as pointers to fill up 300 numbers.

a) Declaring X/Y/Z registers as pointers

```
start:
; Declare POINTERS
LDI XL, LOW(0x0222)
LDI XH, HIGH(0x0222)
LDI YL, LOW(0x0400)
LDI YH, HIGH(0x0400)
LDI ZL, LOW(0x0600)
LDI ZH, HIGH(0x0600)
```

b) Populating 300 numbers and storing them (counter checks for exactly 300 numbers)

```
CLR R22          ; R22 = 0
LDI R22, LOW(0x0222) ; R22 = 0x22
ADD R25, R22      ; R25 = R25 + R22
LDI R22, HIGH(0x0222) ; R22 = 0x02

; Populate and store 300 numbers
popLOOP:
ADD R25, R22      ; R25 += R22
ST X+, R25        ; Store number
INC R21           ; increment counter (low)
JMP check300first

; Increment higher bits
changeHighfirst:
INC R20           ; increment higher bit
CLR R21           ; restarts lower bits
JMP popLOOP

; Checks if counter == 300
check300first:
CPI R21, 0x00     ; Check to increment high bit
BREQ changeHighfirst
CP R24, R21        ; Compare LOW(300) and R21
BRNE popLOOP
CP R23, R20        ; Compare HIGH(300) and R20
BRNE popLOOP
```

c) Memory spaces

Memory space 0x222, first number placed is 0x24 (36) and last number is 0x7a (122)

Memory 4																																								
Memory:	prog BOOT_SECTION_1 ▾																												Address:	0x01DA_data										
data 0x01DA	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
data 0x01F2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
data 0x020A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
data 0x0222	24	26	28	2a	2c	2e	30	32	34	36	38	3a	3c	3e	40	42	44	46	48	4a	4c	4e	50	52																
data 0x023A	54	56	58	5a	5c	5e	60	62	64	66	68	6a	6c	6e	70	72	74	76	78	7a	7c	7e	80	82																
data 0x0252	84	86	88	8a	8c	8e	90	92	94	96	98	9a	9c	9e	a0	a2	a4	a6	a8	aa	ac	ae	b0	b2																
data 0x026A	b4	b6	b8	ba	bc	be	c0	c2	c4	c6	c8	ca	cc	ce	d0	d2	d4	d6	d8	da	dc	de	e0	e2																
data 0x0282	e4	e6	e8	ea	ec	ee	f0	f2	f4	f6	f8	fa	fc	fe	00	02	04	06	08	0a	0c	0e	10	12																
data 0x029A	14	16	18	1a	1c	1e	20	22	24	26	28	2a	2c	2e	30	32	34	36	38	3a	3c	3e	40	42																
data 0x02B2	44	46	48	4a	4c	4e	50	52	54	56	58	5a	5c	5e	60	62	64	66	68	6a	6c	6e	70	72																
data 0x02CA	74	76	78	7a	7c	7e	80	82	84	86	88	8a	8c	8e	90	92	94	96	98	9a	9c	9e	a0	a2																
data 0x02E2	a4	a6	a8	aa	ac	ae	b0	b2	b4	b6	ba	bb	bc	bd	be	c0	c2	c4	c6	c8	ca	cc	ce	d0	d2															
data 0x02FA	d4	d6	d8	da	dc	de	e0	e2	e4	e6	e8	ea	ec	ee	f0	f2	f4	f6	f8	fa	fc	fe	00	02																
data 0x0312	04	06	08	0a	0c	0e	10	12	14	16	18	1a	1c	1e	20	22	24	26	28	2a	2c	2e	30	32																
data 0x032A	34	36	38	3a	3c	3e	40	42	44	46	48	4a	4c	4e	50	52	54	56	58	5a	5c	5e	60	62																
data 0x0342	64	66	68	6a	6c	6e	70	72	74	76	78	7a	00	00	00	00	00	00	00	00	00	00	00	00																

Memory space 0x400, numbers divisible by 5

[illegible]

Memory space 0x600, numbers not divisible by 5

Memory 4																																																												
Memory:	prog BOOT_SECTION_1																												Address:	0x0582_data																														
data 0x0582	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00								
data 0x059A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00						
data 0x05B2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00						
data 0x05CA	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00						
data 0x05E2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00						
data 0x05FA	00	00	00	00	00	00	7a	76	74	72	70	6c	6a	68	66	62	60	5e	5c	58	56	54	52	4e																																				
data 0x0612	4c	4a	48	44	42	40	3e	3a	38	36	34	30	2e	2c	2a	26	24	22	20	1c	1a	18	16	12																																				
data 0x062A	10	e0	0c	08	06	04	02	00	f2	c0	fc	f8	f6	f4	f2	ee	ec	ea	e8	e4	e2	e0	d8																																					
data 0x0642	d6	d4	d0	ce	cc	c6	c4	c2	c0	bc	ba	b8	b6	b2	b0	ae	ac	a8	a6	a4	a2	9e	9c																																					
data 0x065A	9a	98	94	92	90	8e	8a	88	86	84	80	7e	7c	7a	76	74	72	70	6c	6a	68	66	62	60																																				
data 0x0672	5e	5c	58	56	54	52	4e	4c	4a	48	44	42	40	3e	3a	38	36	34	30	2e	2c	2a	26	24																																				
data 0x068A	22	20	1c	1a	18	16	12	10	0e	0c	08	06	04	02	00	fe	fc	f8	f6	f4	f2	ee	ec	ea																																				
data 0x06A2	e8	e4	e2	e0	de	da	d8	d6	d4	d0	ce	cc	ca	c6	c4	c2	c0	bc	ba	b8	b6	b2	b0	ae																																				
data 0x06BA	ac	a8	a6	a4	a2	9e	9c	9a	98	94	92	90	8e	8a	88	86	84	80	7e	7c	7a	76	74	72																																				
data 0x06D2	70	6c	6a	68	66	62	60	5e	5c	58	56	54	52	4e	4c	4a	48	44	42	40	3e	3a	38	36																																				
data 0x06EA	34	30	2e	2c	2a	26	24	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00																																				

Task 2/B: Use X/Y/Z register addressing to parse through the 300 numbers, if the number is divisible by 5 store the number starting from memory location 0x0400, else store at location starting at 0x0600.

Task 3/C: Use X/Y/Z register addressing to simultaneously add numbers from memory location 0x0400 and 0x0600 and store the sums at R16:R17 and R18:R19 respectively. Do not worry about the overflow.

a) checking if each number is divisible by 5, goes to registers R16 and R17 if divisible by 5 and goes to registers R18 and R19 if not

```

loader:
    LD R15, -X          ; Load number to R20 downwards the stack
    MOV R14, R15        ; Moves value in R16 to check if divisible by 5

; checks if %5=0
D5:
    CP R15, R22          ; Compare R15 and R22 (5)
    BRLO ISNOT5          ; Go to ISNOT5 back if R15 < R22
    CP R15, R22          ; Compare R15 and R22 (5)
    BREQ IS5            ; Go to IS5 if R15 == R22
    SUB R15, R22         ; R15 = R15 - 5
    CP R15, R22          ; Compare R15 and R22 again
    BRSH D5             ; Loop back if R15 >= R22

```

b) running the program, R16:R17 (divisible by 5) is 7220 (0x1C34) and R18:R19 (not divisible by 5) is 28768 (0x7060)

```

; when %5 != 0 (not divisible)
ISNOT5:
    ST Z+, R14          ; Stores number
    MOV R13, R14        ; Copies stored number
    ADD R19, R14         ; Simultaneously add number (LOW BITS)
    CP R19, R13          ; Compare LOW BIT with loaded number
    BRLO oNot5          ; goes to oIs5 if R19 < R13
    INC R21             ; Increment counter
    JMP check300second

oNot5:
    INC R18             ; Increment HIGH BIT
    INC R21             ; Increment counter
    JMP check300second

; when %5 == 0 (divisible)
IS5:
    ST Y+, R14          ; Stores number
    MOV R13, R14        ; Copies stored number
    ADD R17, R14         ; Simultaneously add number (LOW BITS)
    CP R17, R13          ; Compare LOW BIT with loaded number
    BRLO oIs5          ; goes to oIs5 if R17 < R13
    INC R21             ; Increment counter
    JMP check300second

oIs5:
    INC R16             ; Increment HIGH BIT
    INC R21             ; Increment counter
    JMP check300second

```

Processor Status	
Name	Value
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x24
R14	0x24
R15	0x01
R16	0x1C
R17	0x34
R18	0x70
R19	0x60
R20	0x01
R21	0x2C
R22	0x05
R23	0x01
R24	0x2C
R25	0x7A
R26	0x22

Task 4/D: Verify your algorithm and answers using C programming

a) The Code

```
#include <stdio.h>

int main()
{
    int rem, fivecount = 0, notfivecount = 0;
    int fivesum = 0, notfivesum = 0;
    int array[300], i, j;

    //stores 300 numbers
    array[0] = 36; //FIRST ELEMENT (0x22 + 0x02)
    for(i = 1 ; i < 300 ; i++)
    {
        if(array[i-1] == 254)
            array[i] = 0;
        else
            array[i] = array[i-1] + 2;
    }

    //check if divisible by 5
    for(j = 0 ; j < 300 ; j++) //TRAVERSE
    {
        rem = array[j] % 5;
        if(rem == 0 && array[j] != 0) //IF DIVISIBLE BY 5
        {
            fivesum += array[j];
            fivecount++;
        }
        else //IF NOT DIVISIBLE BY 5
        {
            notfivesum += array[j];
            notfivecount++;
        }
    }

    //print results
    printf("#s divisible by 5: %d\n", fivecount);
    printf("The result for numbers div. by 5: %d\n", fivesum);
    printf("#s not divisible by 5: %d\n", notfivecount);
    printf("The result for numbers not div. by 5: %d\n", notfivesum);
}
```

b) running the code, the outputs matches the hexadecimal results from Task 3/C, 7220 being 0x1C34 and 28768 being 0x7060

```
[razonjl@cardiac CPE301]$ gcc DA01.c
[razonjl@cardiac CPE301]$ ./a.out
#s divisible by 5: 59
The result for numbers div. by 5: 7220
#s not divisible by 5: 241
The result for numbers not div. by 5: 28768
[razonjl@cardiac CPE301]$
```

Task 5/E: Determine the execution time @ 16MHz/#cycles of your algorithm using the simulation.
 Execution time (in microseconds) = # of cycles (65543 cycles) / frequency (16 MHz) = 4096.44 us

a) calculations on online scientific calculator

$\frac{65543}{16}$	= 4096.4375
--------------------	-------------

b) by changing the frequency on Atmel Studios, the stop watch's execution time changed accordingly

Processor Status	
Name	Value
Program Counter	0x00000050
Stack Pointer	0x08FF
X Register	0x0222
Y Register	0x043B
Z Register	0x06F1
Status Register	I T H S V N Z C
Cycle Counter	65543
Frequency	16.000 MHz
Stop Watch	4,096.44 µs

FULL CODE

```
start:
; Declare POINTERS
LDI XL, LOW(0x0222)
LDI XH, HIGH(0x0222)
LDI YL, LOW(0x0400)
LDI YH, HIGH(0x0400)
LDI ZL, LOW(0x0600)
LDI ZH, HIGH(0x0600)

; Declare variables
CLR R17          ; Stores total sum of numbers that are divisible by 5, set to 0
CLR R16          ; R17 overflow (don't worry about overflow)
CLR R19          ; Stores total sum of numbers that are not divisible by 5, set to 0
CLR R18          ; R18 overflow (don't worry about overflow)

CLR R20          ; Counter register (HIGH), set to 0
CLR R21          ; Counter register (LOW), set to 0
LDI R23, HIGH(300) ; Immediate value 300 H (0x01), count checker
LDI R24, LOW(300)  ; Immediate value 300 L (0x2c), count checker

CLR R22          ; R22 = 0
LDI R22, LOW(0x0222) ; R22 = 0x22
ADD R25, R22      ; R25 = R25 + R22
LDI R22, HIGH(0x0222) ; R22 = 0x02

; Populate and store 300 numbers
popLOOP:
ADD R25, R22      ; R25 += R22
ST X+, R25        ; Store number
INC R21           ; increment counter (low)
JMP check300first
```

```

; Increment higher bits
changeHighfirst:
    INC R20          ; increment higher bit
    CLR R21          ; restarts lower bits
    JMP popLOOP

; Checks if counter == 300
check300first:
    CPI R21, 0x00    ; Check to increment high bit
    BREQ changeHighfirst
    CP R24, R21       ; Compare LOW(300) and R21
    BRNE popLOOP
    CP R23, R20       ; Compare HIGH(300) and R20
    BRNE popLOOP

    CLR R20          ; clear counter(high)
    CLR R21          ; clear counter(low)
    CLR R22          ; clear R22
    LDI R22, 0x05    ; load a 5 to check if divisible

; Loads next number number in stack
loader:
    LD R15, -X        ; Load number to R20 downwards the stack
    MOV R14, R15      ; Moves value in R16 to check if divisible by 5

; checks if %5=0
D5:
    CP R15, R22       ; Compare R15 and R22 (5)
    BRLO ISNOT5       ; Go to ISNOT5 back if R15 < R22
    CP R15, R22       ; Compare R15 and R22 (5)
    BREQ IS5         ; Go to IS5 if R15 == R22
    SUB R15, R22      ; R15 = R15 - 5
    CP R15, R22       ; Compare R15 and R22 again
    BRSH D5          ; Loop back if R15 >= R22

; when %5 != 0 (not divisible)
ISNOT5:
    ST Z+, R14        ; Stores number
    MOV R13, R14      ; Copies stored number
    ADD R19, R14      ; Simultaneously add number (LOW BITS)
    CP R19, R13       ; Compare LOW BIT with loaded number
    BRLO oNot5        ; goes to oIs5 if R19 < R13
    INC R21           ; Increment counter
    JMP check300second

oNot5:
    INC R18           ; Increment HIGH BIT
    INC R21           ; Increment counter
    JMP check300second

; when %5 == 0 (divisible)
IS5:
    ST Y+, R14        ; Stores number
    MOV R13, R14      ; Copies stored number
    ADD R17, R14      ; Simultaneously add number (LOW BITS)
    CP R17, R13       ; Compare LOW BIT with loaded number
    BRLO oIs5        ; goes to oIs5 if R17 < R13
    INC R21           ; Increment counter
    JMP check300second

oIs5:
    INC R16           ; Increment HIGH BIT
    INC R21           ; Increment counter
    JMP check300second

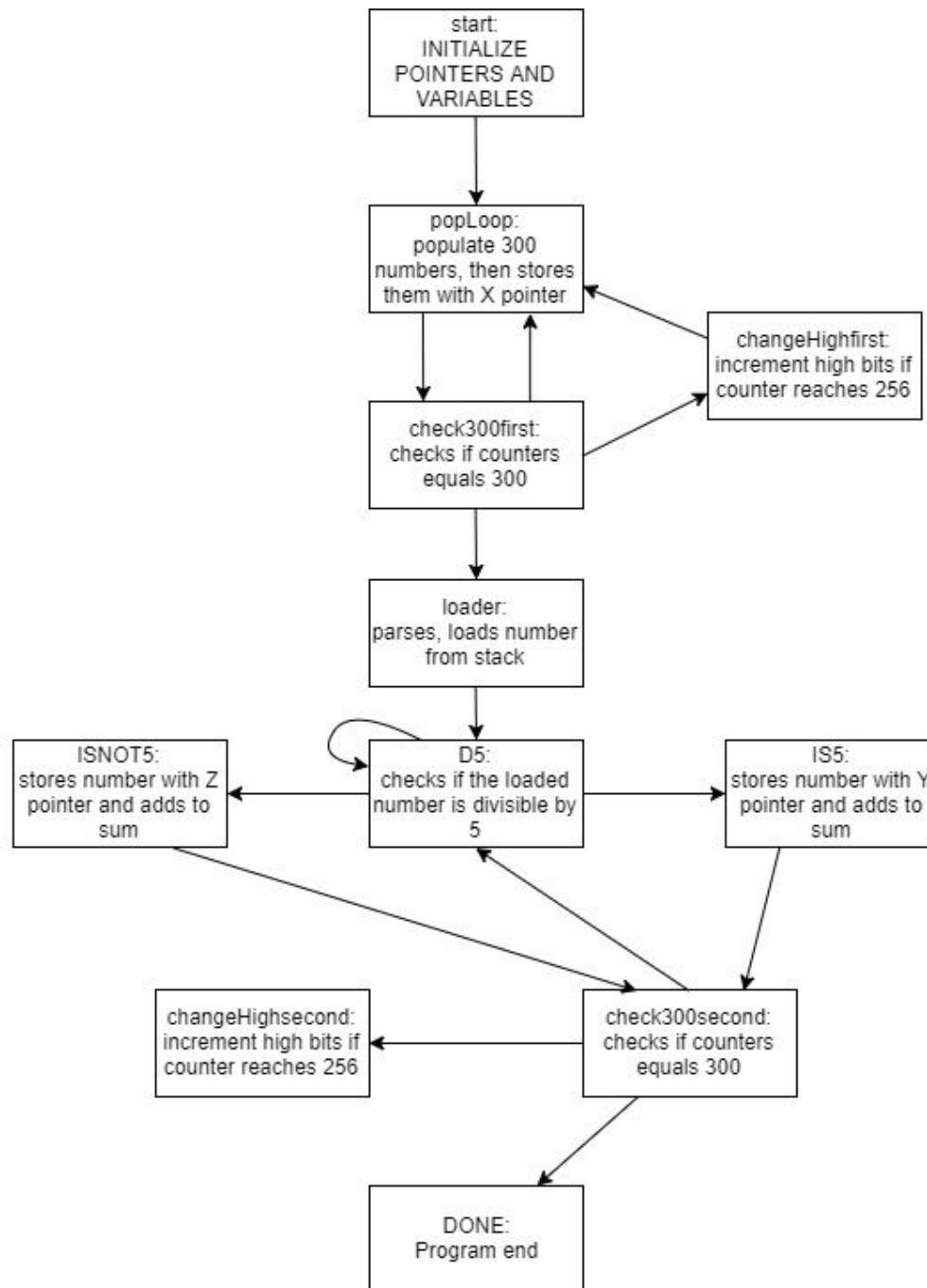
; Increments higher bits
changeHighsecond:
    INC R20          ; increment higher bit
    CLR R21          ; restarts lower bits
    JMP loader

; Checks if counter == 300
check300second:
    CPI R21, 0x00    ; Check to increment high bit
    BREQ changeHighsecond ; counter reaches 256
    CP R24, R21       ; Compare LOW(300) and R21
    BRNE loader      ; loop back if R24 != R21
    CP R23, R20       ; Compare HIGH(300) and R20
    BRNE loader      ; loop back if R23 != R20

DONE:
    NOP              ; Program end

```

FLOW CHART



GITHUB LINK: <https://github.com/JeffinVegas/EmbSys.git>

Student Academic Misconduct Policy

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

"This assignment submission is my own, original work".

Jeffrey Razon