Derek Nhan

CPE301 – SPRING 2016

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)
0.	COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS		
1.	CODE SEGMENT OF TASK 1/A		
2.	CODE SEGMENT OF TASK 2/B/C		
3.	CODE SEGMENT OF TASK 3/D		
4.	COMPLETE CODE		
5.	SCREENSHOTS OF EACH TASK OUTPUT		
6.	FLOW CHART OF ALGORITHM		
7.	VIDEO LINKS OF EACH DEMO		
8.	GOOGLECODE LINK OF THE DA		

1.	INITIAL CODE OF TASK 1/A		

```
;Code segment that puts 25 numbers on to the stack
.def COUNT=r25                      ;counter
.def dividend=r22                   ;dividend register
.def number=r12                     ;number to be added is divided
.def SUM_7H=r20                     ;high of sum of 7
.def SUM_7L=r21                     ;low of sum of 7
.def SUM_3H=r23                     ;high of sum of 3
.def SUM_3L=r24                     ;low of sum of 3
.def OVERFLOW=r7                    ;overflow register for sum

.macro STACK
       ldi @0, high(@1)
       out SPH, @0
       ldi @0, low(@1)
       out SPL, @0
.endmacro

STACK r16, RAMEND

ldi XH, high(RAMEND/2)              ;set X pointer to high bits of middle of ramend
ldi XL, low(RAMEND/2)               ;set X pointer to low bits of middle of ramend
ldi COUNT, 0                        ;set counter to 0

loop:
;loop to store numbers in to RAMEND/2 location
       ldi r17, low(RAMEND/2)
       add r17, COUNT
       st X+, r17
       inc COUNT
       cpi COUNT, 25
       brne loop
```

2.	INITIAL CODE OF TASK 1/B		

```
;Code segment that parses the numbers and check division by 7 and 3 and adds the
;corresponding values
       ldi XH, high(RAMEND/2)
       ldi XL, low(RAMEND/2)
       ldi YH, high(RAMEND/2)
       ldi YL, low(RAMEND/2)
       ldi ZH, high(RAMEND/2)
       ldi ZL, low(RAMEND/2)

again:
       ld number, Z+                ;loads number in to the number var
       ld dividend, X+                     ;loads number to the dividend to be divided
division7:
;loop to divide number by 7
       subi dividend, 7
       cpi dividend, 7
       brsh division7
       cpi dividend, 0
       ld dividend, Y+                     ;if remainder is 0, then the number is divisible
by 7
       breq sum_7
division3:
```

```
;loop to divide number by 3
       subi dividend, 3
       cpi dividend, 3
       brsh division3
       cpi dividend, 0                   ;if remainder is 0, then the number is divisible
by 3
       breq sum_3
div_lp:
       dec COUNT
       cpi COUNT, 0               ;count of the numbers already used
       brne again
       jmp done

sum_7:
;calculates the sum for division by 7
       add SUM_7L, number
       brvs ovr_flw7
       jmp division3
sum_3:
;calculates the sum for division by 3
       add SUM_3L, number
       brvs ovr_flw3
       jmp div_lp
```

3.	INITIAL CODE OF TASK 1/D		

```
;Code segment to set overflow register
ovr_flw7:
;both labels will set overflow register is the sum is greater that 8 bits
       ldi r17, 0x08
       mov OVERFLOW, r17          ;copies r17 to OVERFLOW(r7) register and set bit 3
       subi SUM_7H, -1
       jmp division3
ovr_flw3:
       ldi r17, 0x08
       mov OVERFLOW, r17
       subi SUM_3H, -1
       jmp div_lp
```

4.	Complete Code		

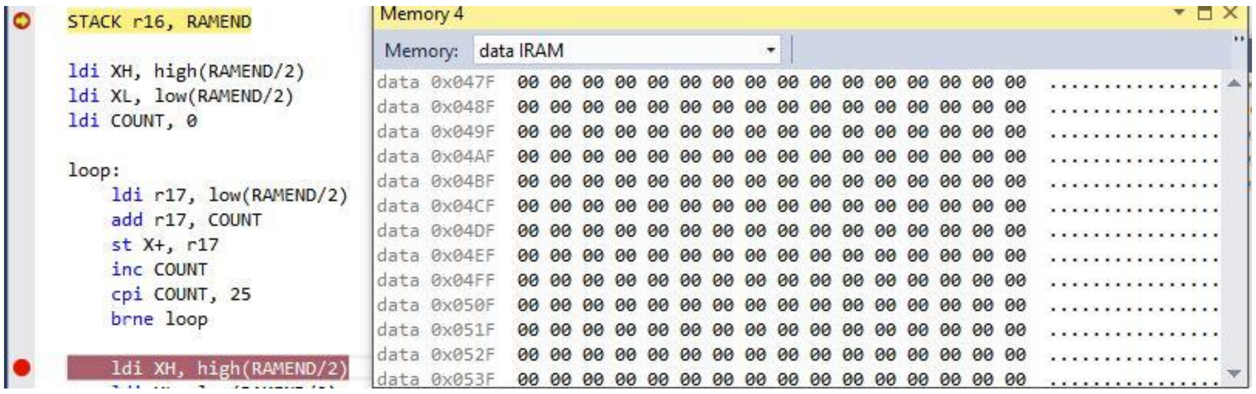
```
.def COUNT=r25                    ;counter
.def dividend=r22          ;dividend register
.def number=r12                   ;number to be added is divided
.def SUM_7H=r20                   ;high of sum of 7
.def SUM_7L=r21                   ;low of sum of 7
.def SUM_3H=r23                   ;high of sum of 3
.def SUM_3L=r24                   ;low of sum of 3
.def OVERFLOW=r7           ;overflow register for sum

.macro STACK
       ldi @0, high(@1)
       out SPH, @0
       ldi @0, low(@1)
       out SPL, @0
.endmacro

STACK r16, RAMEND
```

```
ldi XH, high(RAMEND/2)            ;set X pointer to high bits of middle of ramend
ldi XL, low(RAMEND/2)             ;set X pointer to low bits of middle of ramend
ldi COUNT, 0                      ;set counter to 0

loop:
;loop to store numbers in to RAMEND/2 location
       ldi r17, low(RAMEND/2)
       add r17, COUNT
       st X+, r17
       inc COUNT
       cpi COUNT, 25
       brne loop

;set the X,Y,Z pointers to the first number on the stack
       ldi XH, high(RAMEND/2)
       ldi XL, low(RAMEND/2)
       ldi YH, high(RAMEND/2)
       ldi YL, low(RAMEND/2)
       ldi ZH, high(RAMEND/2)
       ldi ZL, low(RAMEND/2)

again:
       ld number, Z+              ;loads number in to the number var
       ld dividend, X+                   ;loads number to the dividend to be divided
division7:
;loop to divide number by 7
       subi dividend, 7
       cpi dividend, 7
       brsh division7
       cpi dividend, 0
       ld dividend, Y+                   ;if remainder is 0, then the number is divisible
by 7
       breq sum_7
division3:
;loop to divide number by 3
       subi dividend, 3
       cpi dividend, 3
       brsh division3
       cpi dividend, 0                   ;if remainder is 0, then the number is divisible
by 3
       breq sum_3
div_lp:
       dec COUNT
       cpi COUNT, 0               ;count of the numbers already used
       brne again
       jmp done

sum_7:
;calculates the sum for division by 7
       add SUM_7L, number
       brvs ovr_flw7
       jmp division3
sum_3:
;calculates the sum for division by 3
       add SUM_3L, number
       brvs ovr_flw3
       jmp div_lp
```

```
ovr_flw7:
;both labels will set overflow register is the sum is greater that 8 bits
       ldi r17, 0x08
       mov OVERFLOW, r17          ;copies r17 to OVERFLOW(r7) register and set bit 3
       subi SUM_7H, -1
       jmp division3
ovr_flw3:
       ldi r17, 0x08
       mov OVERFLOW, r17
       subi SUM_3H, -1
       jmp div_lp

done:
```

5.	SCREENSHOTS OF EACH TASK OUTPUT		

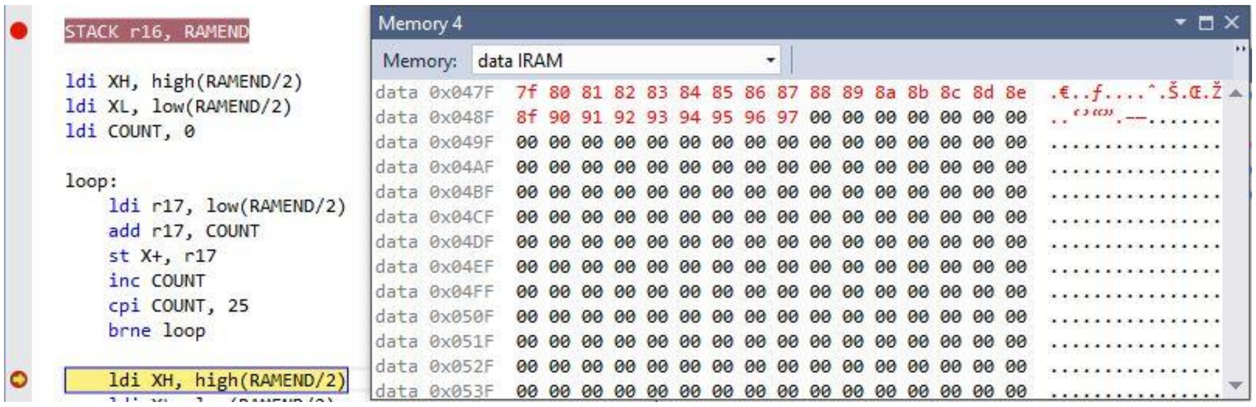
Task 1/A



```
STACK r16, RAMEND

ldi XH, high(RAMEND/2)
ldi XL, low(RAMEND/2)
ldi COUNT, 0

loop:
    ldi r17, low(RAMEND/2)
    add r17, COUNT
    st X+, r17
    inc COUNT
    cpi COUNT, 25
    brne loop

    ldi XH, high(RAMEND/2)
```

```
Memory 4
Memory: data IRAM
data 0x047F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x048F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x049F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04AF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04BF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04CF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04DF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04EF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04FF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x050F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x051F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x052F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x053F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 1: Before storing values

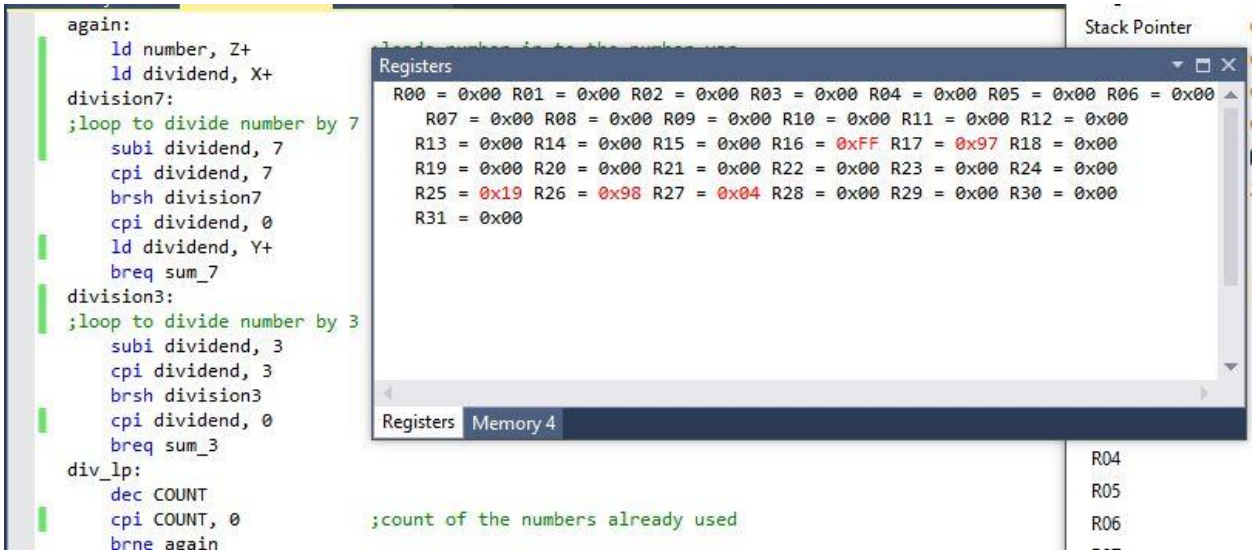


```
STACK r16, RAMEND

ldi XH, high(RAMEND/2)
ldi XL, low(RAMEND/2)
ldi COUNT, 0

loop:
    ldi r17, low(RAMEND/2)
    add r17, COUNT
    st X+, r17
    inc COUNT
    cpi COUNT, 25
    brne loop

    ldi XH, high(RAMEND/2)
```

```
Memory 4
Memory: data IRAM
data 0x047F  7f 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e  .€..ƒ....ˆ.Š.Œ.Ž
data 0x048F  8f 90 91 92 93 94 95 96 97 00 00 00 00 00 00 00  ..‘’“”.–—.......
data 0x049F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04AF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04BF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04CF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04DF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04EF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x04FF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x050F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x051F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x052F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
data 0x053F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 2: After storing values

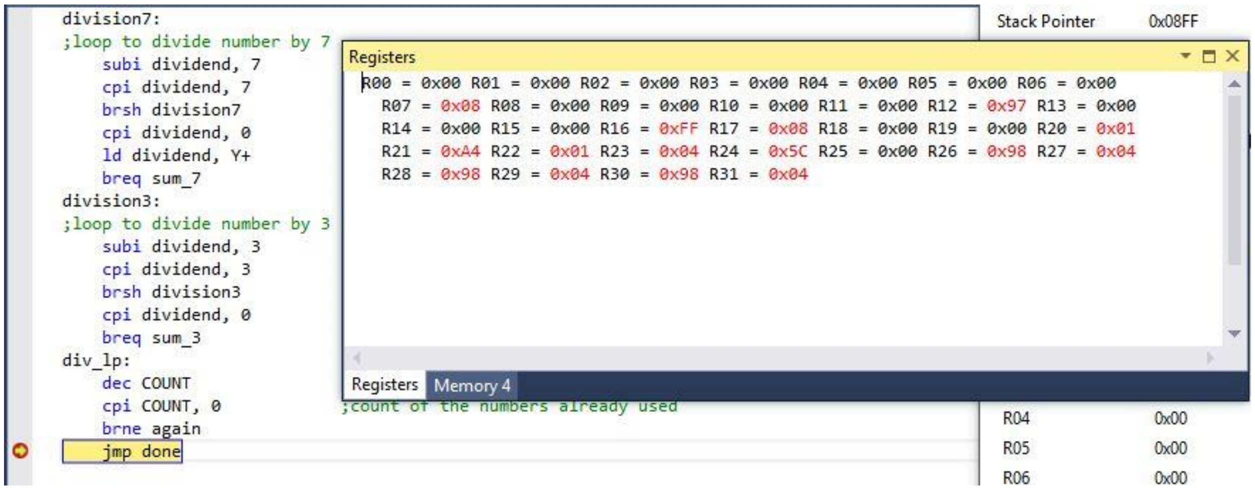
Task 1/B/C/D



```
again:
    ld number, Z+
    ld dividend, X+
division7:
;loop to divide number by 7
    subi dividend, 7
    cpi dividend, 7
    brsh division7
    cpi dividend, 0
    ld dividend, Y+
    breq sum_7
division3:
;loop to divide number by 3
    subi dividend, 3
    cpi dividend, 3
    brsh division3
    cpi dividend, 0
    breq sum_3
div_lp:
    dec COUNT
    cpi COUNT, 0            ;count of the numbers already used
    brne again
```

```
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
   R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00
  R13 = 0x00 R14 = 0x00 R15 = 0x00 R16 = 0xFF R17 = 0x97 R18 = 0x00
  R19 = 0x00 R20 = 0x00 R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00
  R25 = 0x19 R26 = 0x98 R27 = 0x04 R28 = 0x00 R29 = 0x00 R30 = 0x00
  R31 = 0x00
```

Figure 3: Task B,C,D before performing arithmetic/parse



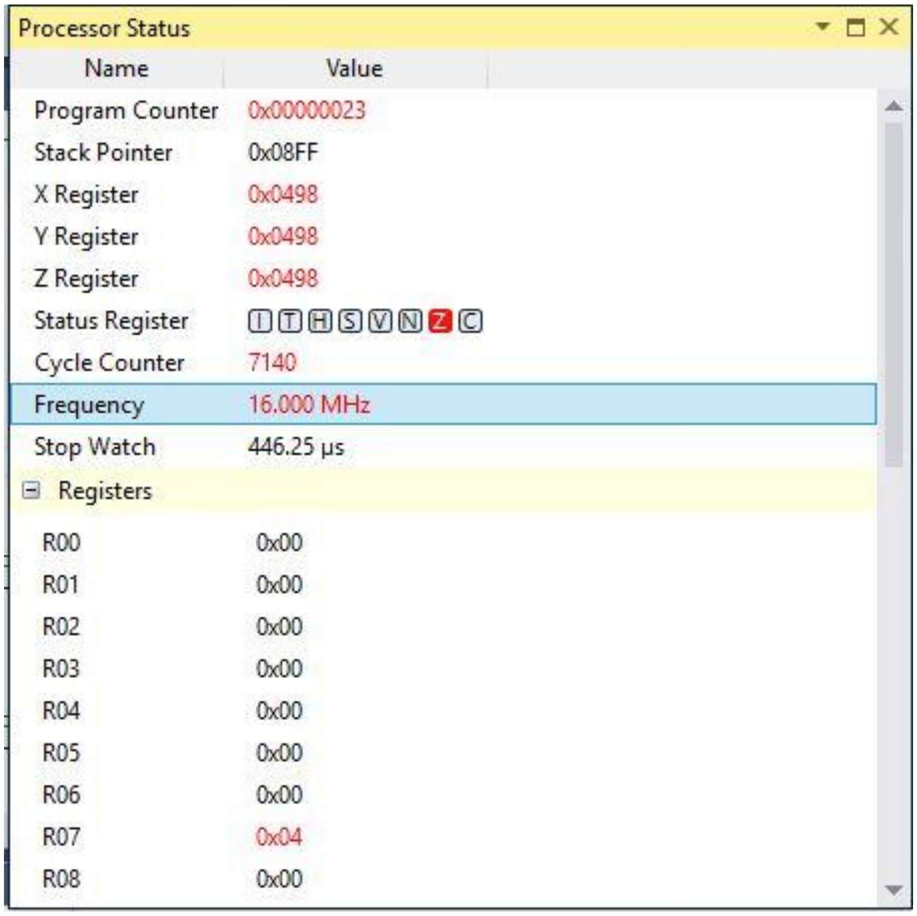
```
division7:
;loop to divide number by 7
    subi dividend, 7
    cpi dividend, 7
    brsh division7
    cpi dividend, 0
    ld dividend, Y+
    breq sum_7
division3:
;loop to divide number by 3
    subi dividend, 3
    cpi dividend, 3
    brsh division3
    cpi dividend, 0
    breq sum_3
div_lp:
    dec COUNT
    cpi COUNT, 0            ;count of the numbers already used
    brne again
    jmp done
```

```
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
  R07 = 0x08 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x97 R13 = 0x00
  R14 = 0x00 R15 = 0x00 R16 = 0xFF R17 = 0x08 R18 = 0x00 R19 = 0x00 R20 = 0x01
  R21 = 0xA4 R22 = 0x01 R23 = 0x04 R24 = 0x5C R25 = 0x00 R26 = 0x98 R27 = 0x04
  R28 = 0x98 R29 = 0x04 R30 = 0x98 R31 = 0x04
```

Stack Pointer	0x08FF
R04	0x00
R05	0x00
R06	0x00

Figure 4: Task B,C,D after performing arithmetic/parse

Task 1/E

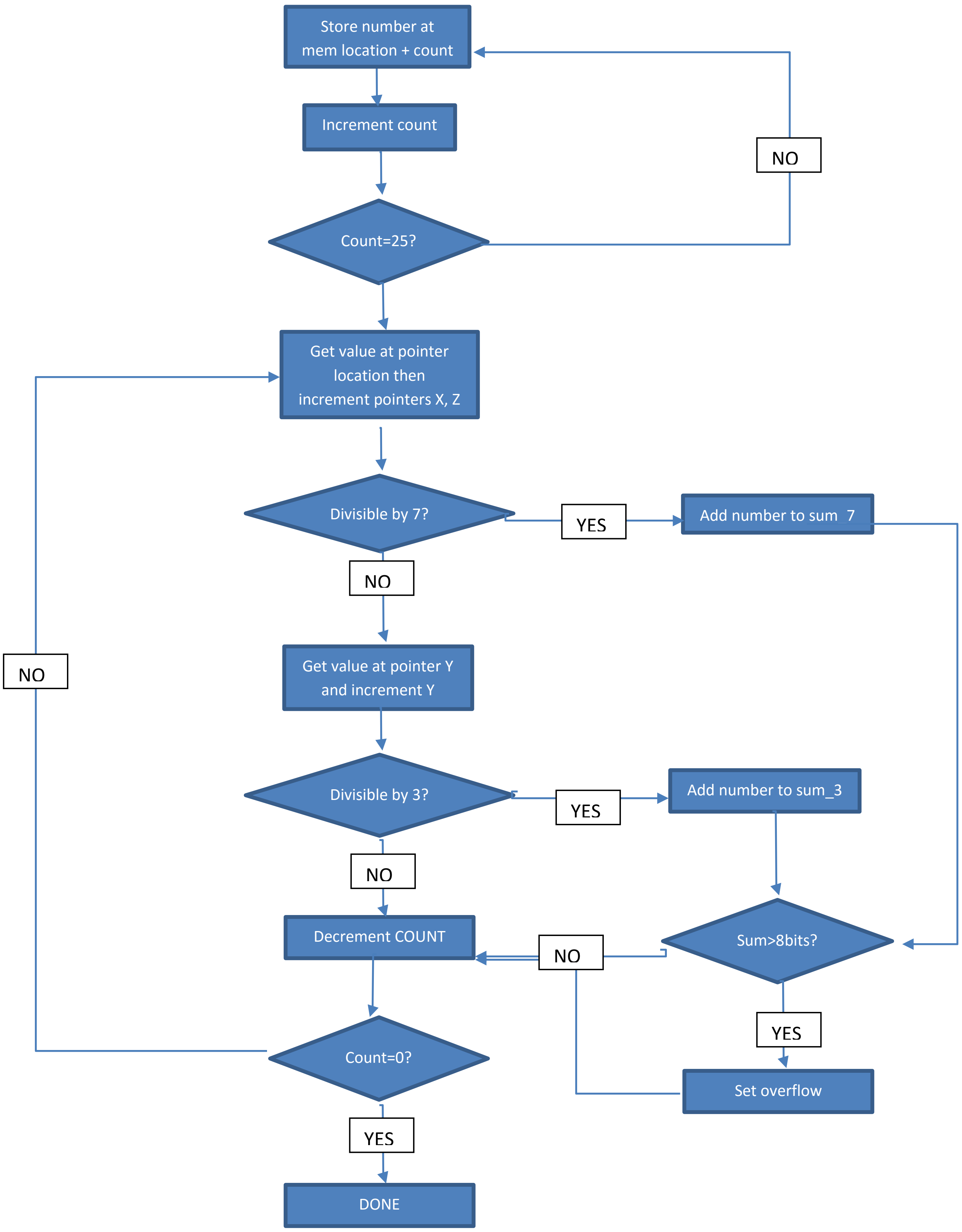


Processor Status

Name	Value
Program Counter	0x00000023
Stack Pointer	0x08FF
X Register	0x0498
Y Register	0x0498
Z Register	0x0498
Status Register	I T H S V N Z C
Cycle Counter	7140
Frequency	16.000 MHz
Stop Watch	446.25 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x04
R08	0x00

Figure 5: Clock at 16MHz, execution done in 446.25 microseconds

7.	FLOW CHART OF ALGORITHM		



9.	VIDEO LINKS OF EACH DEMO		
http:// @youtube			
10.	GOOGLECODE LINK OF THE DA		
hhttps://github.com/nhand2/CPE301S16.git			

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

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