Temple University

College of Engineering

Department of Electrical and Computer Engineering (ECE)

Student Lab Report Cover Page

Course Number	**:**	3613
Course Section	**:**	001 / 002
Experiment #	**:**	Lab # 4

Student Name (print)	**:**	Von Kaukeano
TUid#	**:**	915596703
Date	**:**	9/26/19

Grade	**:**	______ **/100**
TA Name	**:**	Sung Choi

ACTIVITIES:

Activity 1

Write assembly codes to load values to the IO registers/ports and the memory locations (60 pts).

1.1 Load the given numbers to the assigned IO ports.
Show your code and the resulted values of the ports (each port screenshot after you run each port output line in the code, simulation only): (30 points)

Requirements:

(1) $4A to PORTA
(2) 10011111 to PORTB
(3) 96 to PORTC
(4) ‘P’ (ASCII value) to PORTD
(5) 2’s complement of $C1 to PORTA
(6) Sum of $54 and $1F to PORTB

CODE (copy and paste)

```
start:
            ;initiate the stack pointer
            ldi r16, LOW(RAMEND) ;
            out spl, r16
            ldi r17, HIGH(RAMEND);
            out sph, r17
```

; YOUR CODE HERE

```
1-1
a.
start:
    LDI R16, LOW(RAMEND)
            OUT SPL, R16
            LDI R17, HIGH(RAMEND)
            OUT SPH, R17

            LDI R16, 0XFF //LOAD R16
            OUT DDRA,R16 // SET TO OUTPUT
            LDI R17,$4A // LOAD R17
            OUT PORTA, R17

1-1
b.
start:
    LDI R16, LOW(RAMEND)
            OUT SPL, R16
            LDI R17, HIGH(RAMEND)
            OUT SPH, R17
```

```
        LDI R16, 0XFF //LOAD R16
        OUT DDRB,R16 // SET TO OUTPUT
        LDI R17,0b10011111 // LOAD R17
        OUT PORTB, R17
```

1-1
c.

```
start:
   LDI R16, LOW(RAMEND)
        OUT SPL, R16
        LDI R17, HIGH(RAMEND)
        OUT SPH, R17

        LDI R16, 0XFF //LOAD R16
        OUT DDRC,R16 // SET TO OUTPUT
        LDI R17,96 // LOAD R17
        OUT PORTC, R17
```

1-1
d.

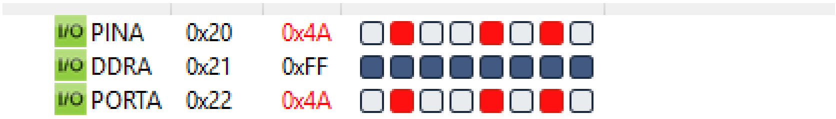
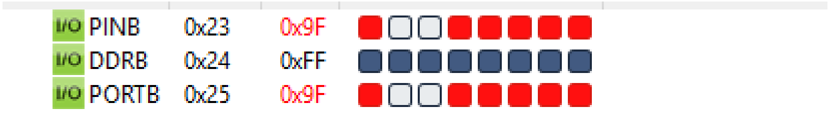
```
start:
   LDI R16, LOW(RAMEND)
        OUT SPL, R16
        LDI R17, HIGH(RAMEND)
        OUT SPH, R17

        LDI R16, 0XFF //LOAD R16
        OUT DDRD,R16 // SET TO OUTPUT
        LDI R17,'P' // LOAD R17
        OUT PORTD, R17
```

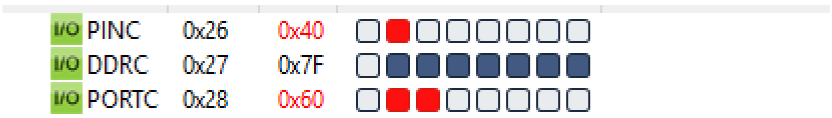
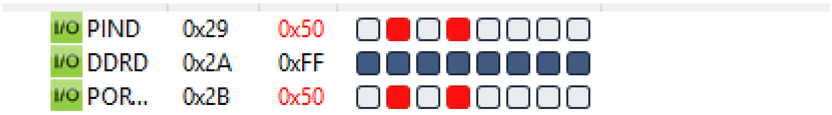
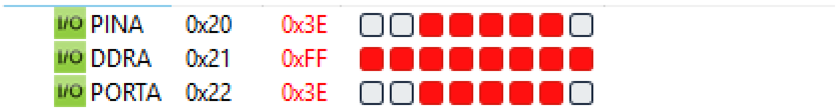
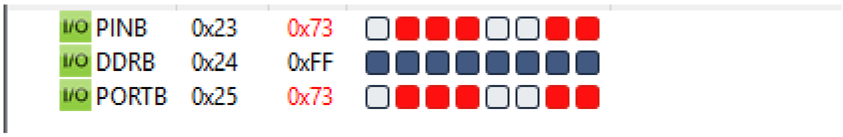
1-1
e.

```
start:
   LDI R16, LOW(RAMEND)
        OUT SPL, R16
        LDI R17, HIGH(RAMEND)
        OUT SPH, R17

        LDI R16, 0XFF //LOAD R16
        OUT DDRA,R16 // SET TO OUTPUT
        LDI R17,$C1 // LOAD R17
        COM R17     // 2's Complement
        OUT PORTA, R17
```

```
1-1
f.
start:
    LDI R16, LOW(RAMEND)
            OUT SPL, R16
            LDI R17, HIGH(RAMEND)
            OUT SPH, R17

            LDI R16, 0XFF //LOAD R16
            OUT DDRB,R16 // SET TO OUTPUT
            LDI R17,$54 // LOAD R17
            LDI R18,$1F // LOAD R18\
            ADD R17,R18
            OUT PORTB, R17
```

RESULT(Screenshots of IO ports)

Index	Value	PORT	Result Screenshot of IO PORT
(1)	$4A	A	 I/O PINA 0x20 0x4A □🟥□□🟥□🟥□ I/O DDRA 0x21 0xFF ■■■■■■■■ I/O PORTA 0x22 0x4A □🟥□□🟥□🟥□
(2)	10011111	B	 I/O PINB 0x23 0x9F 🟥□□🟥🟥🟥🟥🟥 I/O DDRB 0x24 0xFF ■■■■■■■■ I/O PORTB 0x25 0x9F 🟥□□🟥🟥🟥🟥🟥

(3)	96	C	 I/O PINC 0x26 0x40 □🟥□□□□□□ I/O DDRC 0x27 0x7F □🟦🟦🟦🟦🟦🟦🟦 I/O PORTC 0x28 0x60 □🟥🟥□□□□□
(4)	‘p’	D	 I/O PIND 0x29 0x50 □🟥□🟥□□□□ I/O DDRD 0x2A 0xFF 🟦🟦🟦🟦🟦🟦🟦🟦 I/O POR... 0x2B 0x50 □🟥□🟥□□□□
(5)	Negative of $C1	A	 I/O PINA 0x20 0x3E □□🟥🟥🟥🟥🟥□ I/O DDRA 0x21 0xFF 🟥🟥🟥🟥🟥🟥🟥🟥 I/O PORTA 0x22 0x3E □□🟥🟥🟥🟥🟥□
(6)	Sum of $54 and $1F	B	 I/O PINB 0x23 0x73 □🟥🟥🟥□□🟥🟥 I/O DDRB 0x24 0xFF 🟦🟦🟦🟦🟦🟦🟦🟦 I/O PORTB 0x25 0x73 □🟥🟥🟥□□🟥🟥

1.2 Load the values from section 1.1 (1)-(6) to the specified location in the memory. To store the values in the location, use the specified directives for each location.

Requirements:

(1) Set the six directives for the memory locations, 0x100, 0x101, 0x102, 0x103, 0x104, and 0x105.

Index	Directive Names	Memory Address
1	HexNum	0x0100
2	BinNum	0x0101
3	DecNum	0x0102
4	ASCIINum	0x0103
5	TwoComp	0x0104
6	SumNum	0x0105

(2) Store the numbers from the section 1.1 to the memory locations of the directives.

Index	Numbers from Sec 1.1	Directives
1	$4A	HexNum
2	10011111	BinNum
3	96	DecNum
4	‘P’	ASCIINum
5	2’s complementary of $C1	TwoComp
6	Sum of $54 and $1F	SumNum

```
start:
        ;initiate the stack pointer
        ldi r16, LOW(RAMEND) ;
        out spl, r16
        ldi r17, HIGH(RAMEND);
        out sph, r17
```

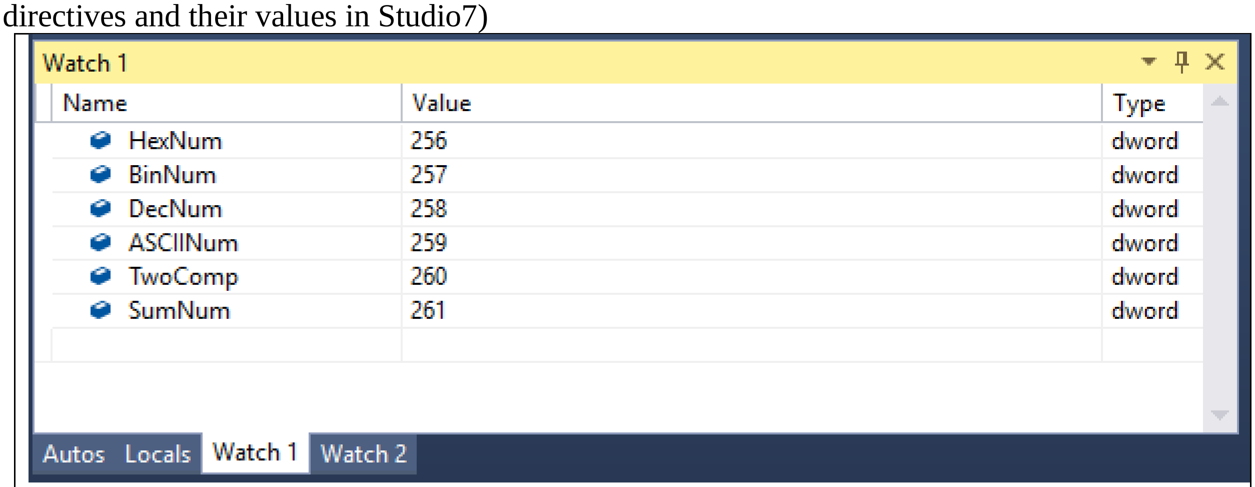
; YOUR CODE HERE

```
start:
   LDI R16, LOW(RAMEND)
        OUT SPL, R16
        LDI R17, HIGH(RAMEND)
        OUT SPH, R17

        .equ HexNum = 0x0100
        .equ BinNum = 0x0101
        .equ DecNum = 0x0102
        .equ ASCIINum = 0x0103
        .equ TwoComp = 0x0104
        .equ SumNum = 0x0105

        LDI R17,$54 // LOAD R17
        LDI R18,0b010011111 // LOAD R18
        LDI R19,96 //LOAD R19
        LDI R20,'P' //LOAD R20
        LDI R21,$C1 //LOAD R21
        COM R21
        LDI R22,$54 // LOAD R22
        LDI R23,$1F // LOAD R23
        ADD R22,R23

        STS HexNum, R17
        STS BinNum, R18
        STS DecNum, R19
        STS ASCIINum, R20
        STS TwoComp, R21
        STS SumNum, R22
```

CODE (copy and paste)

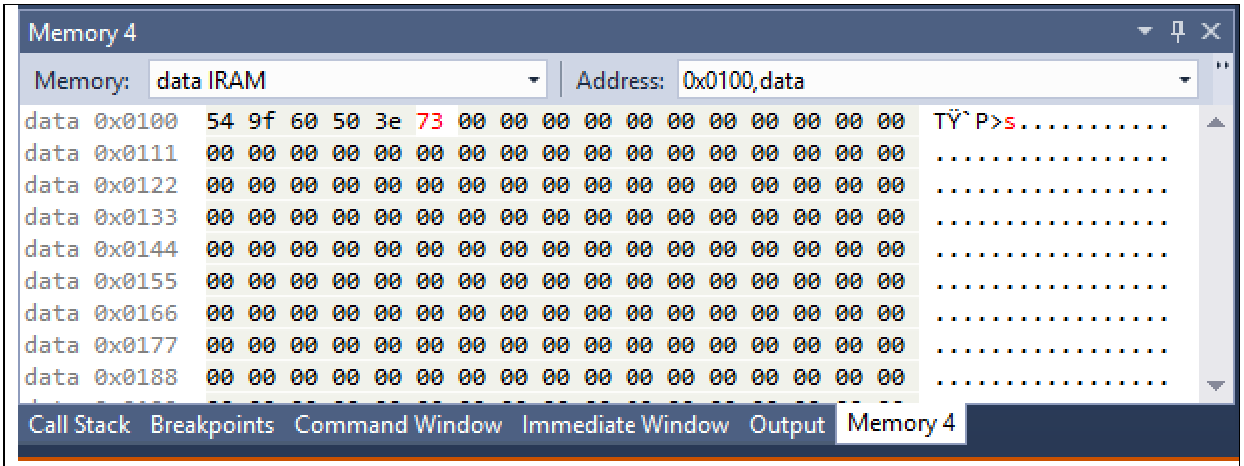
RESULT

(Screenshots of the stored values in the memory locations and the watch view showing the directives and their values in Studio7)



Watch 1

Name	Value	Type
HexNum	256	dword
BinNum	257	dword
DecNum	258	dword
ASCIINum	259	dword
TwoComp	260	dword
SumNum	261	dword



Memory: data IRAM

Address: 0x0100,data

```
data 0x0100  54 9f 60 50 3e 73 00 00 00 00 00 00 00 00 00 00 00  TŸ`P>s...........
data 0x0111  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0122  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0133  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0144  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0155  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0166  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0177  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0188  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
```

Activity 2

Write assembly codes to perform (40 pts).

Read a given ASCII string and read each value of the string to load into the memory locations. Also, show the total count of the types of ASCII letters in the specified ports. Use the ASCII values' corresponding hex values to find that each value is in the range of capital letter, lower-case letter, or the numerical value.

Requirements:

(1) You must draw a flowchart to do the operation.
(2) You must use a branching instruction.
(2) Read each value from the ASCII string, **'Fall2019PS'** and store them into the locations:

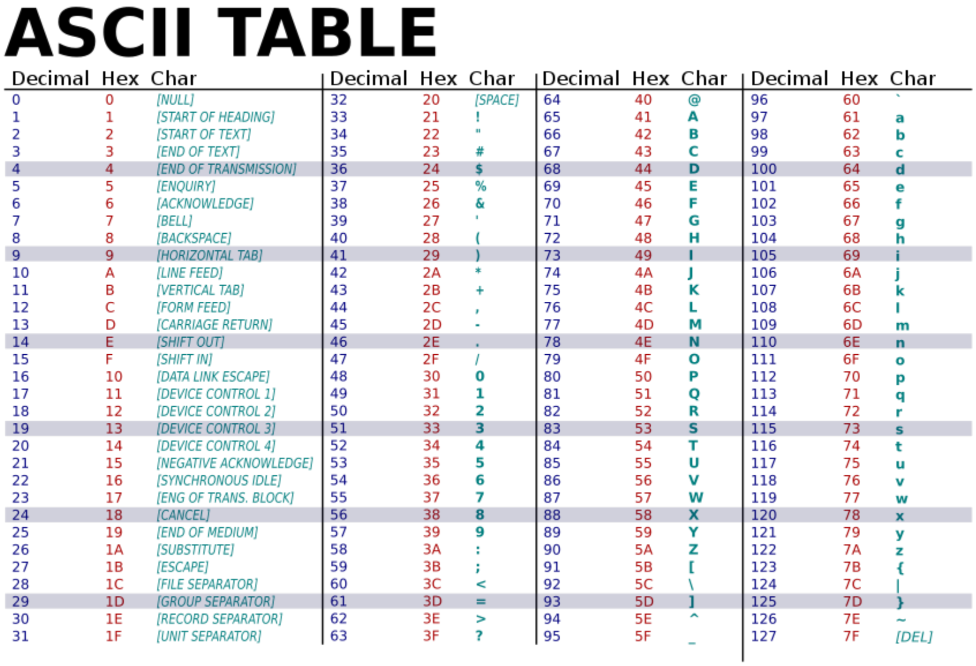
Index	ASCII	Memory Locations
1	F	0x0200
2	a	0x0201
3	l	0x0202
4	l	0x0203
5	2	0x0204
6	0	0x0205
7	1	0x0206
8	9	0x0207
9	P	0x0208
10	S	0x0209

(3) Count the total number of the capital letter (A-Z) in the string and show the counted number using PORTA.
(4) Count the total number of the lower-case letter (a-z) in the string and show the counted number using PORTB.
(5) Count the numbers (0-9) and show the counted number using PORTC.

About the ASCII:

- ASCII (ˈæski) abbreviated from, American Standard Code for Information Interchange, is a character encoding standard for electronic communication.

- ASCII codes represent text in computers, telecommunications equipment, and other devices. Most modern character-encoding schemes are based on ASCII, although they support many additional characters.



ASCII TABLE

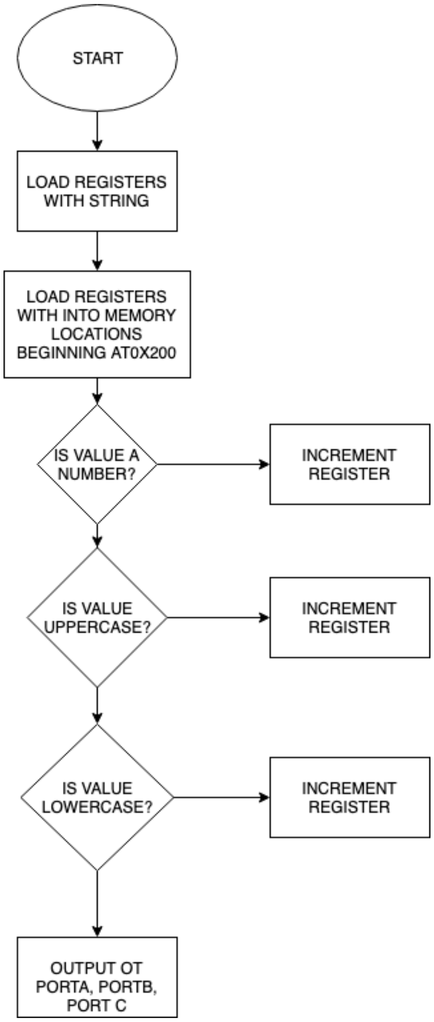
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Table 1.1: ASCII Table

Sample code to use X pointer for incrementing the data memory location from 0x0200:

```
start:
        ;initiate the stack pointer
        ldi r16, LOW(RAMEND) ;
        out spl, r16
        ldi r17, HIGH(RAMEND);
        out sph, r17

        ;load an ASCII value to R20 and load the number 5 to R21 (control loop)
        ldi r20, ‘F’
        ldi r21, 6

        ;set the X pointer to the memory location at 0x0200
        ldi xl, 0x00                ;assign lower byte of the address (16 bits) to the x pointer
                                      lower byte location
        ldi xh, 0x02                ;assign higher byte of the address (16 bits) to the x pointer
                                      higher byte location
        ;store the value or R20 in 0x200, 0x201, 0x202, 0x203, 0x204, and 0x205
op:     st x+, r20                  ;increase the pointer value (address where the pointer is
                                      pointing) and load value to R20
        dec r21                     ;decrement the value of R21
        brne op                     ;branch back to op if Z=0, otherwise go to the next line

here:   jmp here                    ;stay here forever
```

Flowchart


START
LOAD REGISTERS WITH STRING
LOAD REGISTERS WITH INTO MEMORY LOCATIONS BEGINNING AT0X200
IS VALUE A NUMBER?
INCREMENT REGISTER
IS VALUE UPPERCASE?
INCREMENT REGISTER
IS VALUE LOWERCASE?
INCREMENT REGISTER
OUTPUT OT PORTA, PORTB, PORT C

CODE (copy and paste)

```
start:
        ;initiate the stack pointer
        ldi r16, LOW(RAMEND) ;
        out spl, r16
        ldi r17, HIGH(RAMEND);
        out sph, r17
```

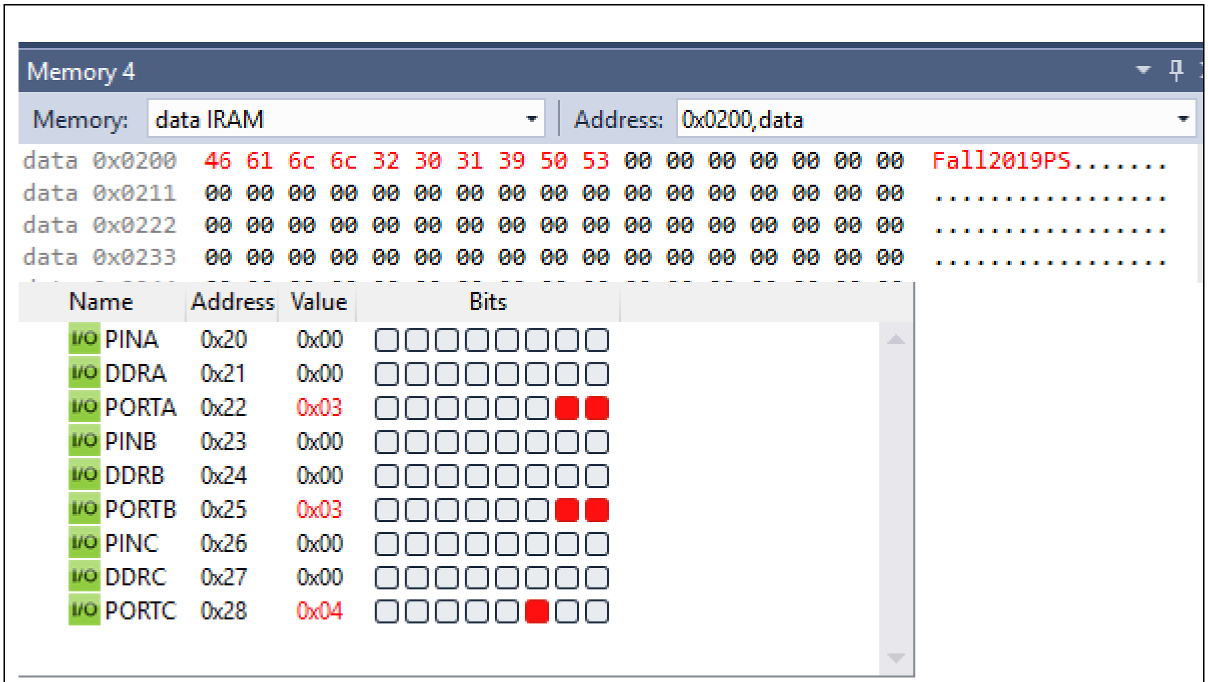
; YOUR CODE HERE

```
start:
        // INITIATE STACK POINTER
        LDI R16, LOW(RAMEND)
        OUT SPL, R16
        LDI R17, HIGH(RAMEND)
        OUT SPH, R17

        LDI R18,'F' // LOAD R18
        LDI R19,'a' // LOAD R19
        LDI R20,'l' //LOAD R20
        LDI R21,'l' //LOAD R21
        LDI R22,'2' //LOAD R22
        LDI R23,'0' // LOAD R23
        LDI R24,'1' // LOAD R24
        LDI R25,'9' // LOAD R25
        LDI R26,'P' // LOAD R26
        LDI R27,'S' // LOAD R27
        LDI R28,0 //LOAD R28
        LDI R29,0 //LOAD R28
        LDI R30,0 //LOAD R28
        LDI R31,0 //LOAD R28

        //STORE INTO MEMORY LOCATION
        STS 0x0200, R18
        STS 0x0201, R19
        STS 0x0202, R20
        STS 0x0203, R21
        STS 0x0204, R22
        STS 0x0205, R23
        STS 0x0206, R24
        STS 0x0207, R25
        STS 0x0208, R26
        STS 0x0209, R27

        //SET X POINTER TO THE MEMORY LOCATION AT 0X200
        LDI XL,0x00 // ASSIGN LOWER BYTE OF THE ADDRESS  TO THE X
POINTER LOWER BYTE LOCATION
        LDI XH,0x02 // ASSIGN HIGHER BYTE OF THE ADDRESS  TO THE X
POINTER HIGHER BYTE LOCATION

OP: ld R28,x+ //LOAD R28 WITH LOOPING MEMORY LOCATIONS  // LOOP
THROUGH MEMORY LOCATIONS AND LOAD IN R28
        CPI R28,0 // COMPARE R28 TO 0
        BREQ OUTPUT // IF R28 IS 0,BRANCH TO OUTPUT

        CPI R28,0X3A // COMPARE R28 WITH 0X3A. THE HIGHEST VALUE TO BE
A NUMBER
        BRLO NUMBER_COUNT // BRANCH TO NUMBER_COUNT IF R28 IS
LOWER

        CPI R28,0X5B // COMPARE R28 WITH 0X5B. THE HIGHEST VALUE TO BE
UPPER CASE
        BRLO UPCASE_COUNT // BRANCH TO NUMBER_COUNT IF R28 IS
LOWER

        CPI R28,0X7B // COMPARE R28 WITH 0X5B. THE HIGHEST VALUE TO BE
UPPER CASE
```

```
        BRLO LOWCASE_COUNT // BRANCH TO NUMBER_COUNT IF R28 IS LOWER

NUMBER_COUNT:
        INC R31 //COUNTER FOR NUMBER
        RJMP OP //BACK TO LOOP

UPCASE_COUNT:
        INC R29 //COUNTER FOR UPPERCASE
        RJMP OP //BACK TO LOOP

LOWCASE_COUNT:
        INC R30 //COUNTER FOR LOWERCASE
        RJMP OP //BACK TO LOOP

OUTPUT:
        OUT PORTA,R29 //OUTPUT PORTA
        OUT PORTB,R30 //OUTPUT PORTB
        OUT PORTC,R31 //OUTPUT PORTC

HERE: RJMP HERE
```

Result

(Screenshots of the memory locations and the IO ports)



Memory: data IRAM

Address: 0x0200,data

```
data 0x0200  46 61 6c 6c 32 30 31 39 50 53 00 00 00 00 00 00 00  Fall2019PS.......
data 0x0211  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0222  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
data 0x0233  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .................
```

Name	Address	Value	Bits
I/O PINA	0x20	0x00	□□□□□□□□
I/O DDRA	0x21	0x00	□□□□□□□□
I/O PORTA	0x22	0x03	□□□□□□🟥🟥
I/O PINB	0x23	0x00	□□□□□□□□
I/O DDRB	0x24	0x00	□□□□□□□□
I/O PORTB	0x25	0x03	□□□□□□🟥🟥
I/O PINC	0x26	0x00	□□□□□□□□
I/O DDRC	0x27	0x00	□□□□□□□□
I/O PORTC	0x28	0x04	□□□□□🟥□□

ECE3613 Processor System Laboratory Rubric
Lab #: 4
Section: 001 / 002
Name: ________________________

Activity	Section	Task	Full Points	Earned Points	Comment
1	1.1	Code	15		
		Result	15		
Subtotal			30		
	1.2	Code	10		
		Result (memory values)	10		
		Result (directive values)	10		
Subtotal			30		
Total for Activity 1			60		
2		Flowchart	10		
		Code	10		
	Result	Memory values	10		
		Port values	10		
Total for Activity 2			40		
Total			**100**		