Electrical and Computer Engineering
EE3612Microprocessor Systems Laboratory

Lab 4: Assembly Language Programming – IO Ports and Directives

Fall 2019

OBJECTIVES:

- To learn how to use IO registers and IO ports
- To learn how to use the directives
- To learn how to draw and use flowchart in programming

REFERENCES:

Mazidi and Naimi, “The AVR Microcontroller and Embedded Systems,” , 2nd Ed. Chapters 1 and 2.

MATERIALS:

Atmel Studio 7

MAP OF THIS LAB:

- Activity 1
  1.1 Write a code to load values to IO registers and show in IO ports
  1.2 Write a code to load values to the memory locations using directives
- Activity 2
  Draw a flowchart and use them to write the corresponding code

LAB REPORT INSTRUCTIONS:

This lab consists of three activities. Use the given report frame to write your report and submit to Canvas assignment. **No full report is required.**

Submission Type: short assignment report in doc and pdf (use the given frame)
Due: section 01 – 10/01/2019 2:59pm, section 02 – 10/03/2019 8:59am

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 complementary of $C1 to PORTA
(6) Sum of $54 and $1F to PORTB

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

Activity 2

Write assembly codes to perform (40 pts).

We are going to 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.

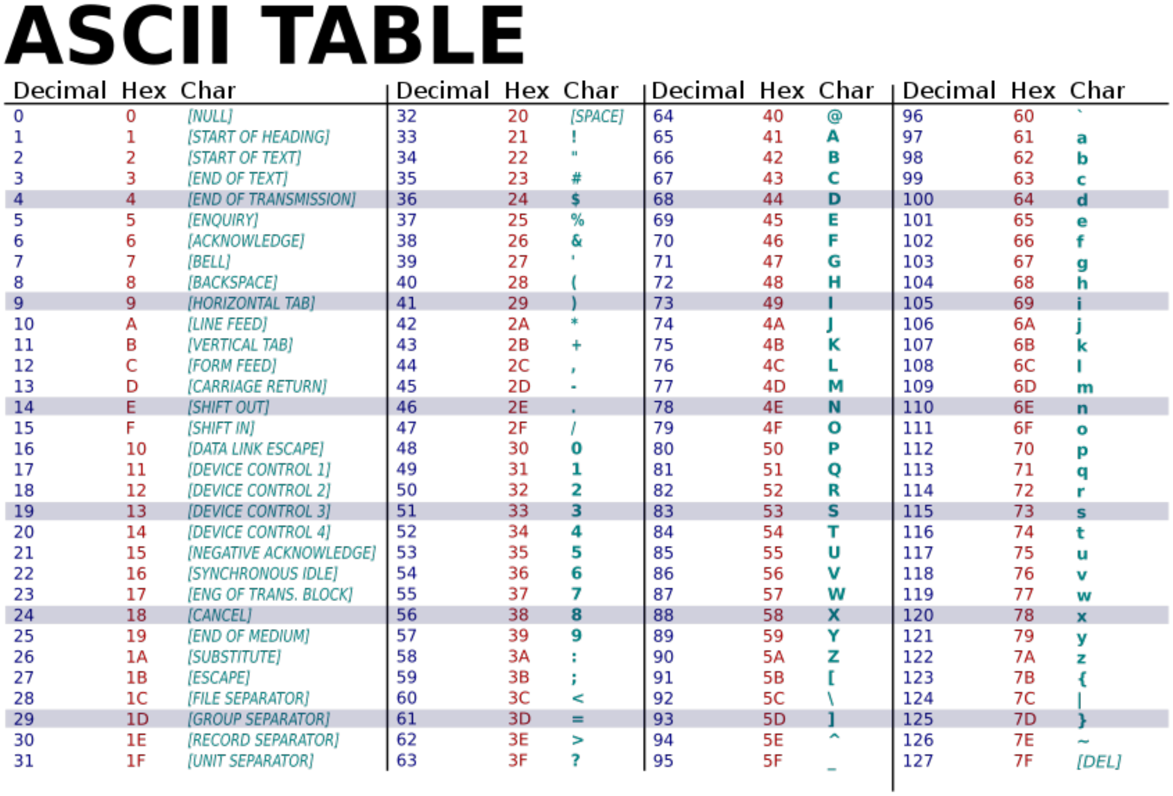
(4) Count the total number of the lower-case letter (a-z) in the string and show the counted number using PORTB.

(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) after load the value to R20 at the initial location of X pointer
        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
```