Electrical and Computer Engineering

EE3613 Processor Systems Laboratory

LAB 8

AVR Advanced Assembly Language Programming –

Rotating, Shifting Operation and Look Up Table

Fall 2019

In lab 8, we will experience a more advanced level of assembly language programming that uses the commonly used techniques: bitwise rotating and shifting operation, and utilizing lookup table programmed in ROM. The rotating and shifting operation is widely used in the data serializing process in the data transfer. The operations allow transferring a bit at a time through the serial port. Shifting each bit of data to left or right results the data value changes to the same result of the arithmetic operations: multiplication and division by 2^n. In this lab, we are going to observe the result of left and right shifting operation.

The lookup table contains the allocated data in flash memory (ROM). The directives .ORG and .DB (.DB: define byte, .DW: define word) allows defining data in the flash memory to be used as a lookup table in the process. The data can be accessed by the Z pointer to be loaded to the GPRs. In Activity 1, we will learn how to operate shifting. In Activity 2, we will write data in the flash memory location and access the data location to make the output using the 7-segment display (see Lab 7 for 7-segment display specification). Each activity requires to develop the flowchart and code to a based on the requirements. Follow the steps to complete the activities.

OBJECTIVES:

- To program AVR microprocessor using assembly language
- To learn the rotating and shifting operation
- To learn how to program the look up table in ROM and access to the data
- To learn how to access the program memory using pointers
- To practice using the Switch, LEDs, and 7-sigment display

REFERENCES:

Mazidi and Naimi, "The AVR Microcontroller and Embedded Systems," 2nd Ed. Chapters 5&6.

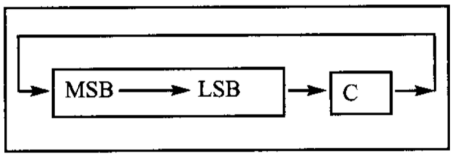
MATERIALS:

Atmel Studio 7, Atmel 324PB XPlained PRO Board (PORTA: 7-Segment display. The detailed board information is available in the lab7), servo.

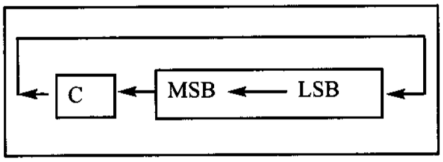
BACKGROUND INFORMATION:

(1) Rotating and shifting instructions

Rotating operation and shifting operation both are used in the data serialization in data transfer process. The rotation is rotating data bit to left or right including carry bit.

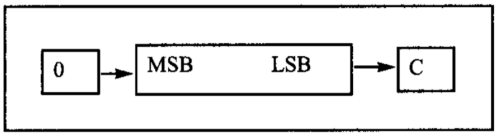


ROR Operation

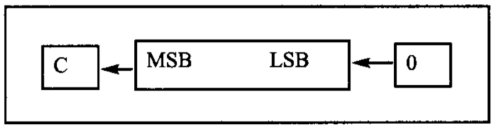


ROL Operation

The shifting operation shift data bit to left or right with carry bit. The result of a single bit left shifting causes the value multiply by 2^n and right shifting causes the value divided by 2^n . See Example 1.



LSR Operation



LSL Operation

Table 1. Syntax and function description for the rotating and shifting instructions

Instruction	Syntax	Function
ROR	ROR Rn	Rotate Rn right through Carry
ROL	ROL Rn	Rotate Rn left through Carry
LSR	LSR Rn	Logical shift right
>>	Rn >> k	shift Rn right by k bits, shift operator
LSL	LSL Rn	Logical shift right
<<	Rn << k	shift Rn left by k bits, shift operator
ASR	ASR Rn	Arithmetic shift right

Example 1:
The code stores the hexadecimal value 0x22 into R20 and does bit by bit shifting three time. The shifting result can be compared with the multiplication of 2^n**.**

```
LDI R20,0x22        ;load 0x22 to R20, R20=0010 0010
LSL R20             ;shift R20 by 1 bit, R20=0100 0100 = 0x44, R20*2, C=0
LSL R20             ;shift R20 by 1 bit, R20=1000 1000 = 0x88, R20*2, C=0
LSL R20             ;shift R20 by 1 bit, R20=0001 0000 = 0x10, C=1 (no multiplication by 2)
```

Using a shift operator for the same operations in Example 1:
Shifting can be done using the shift operators, << (shift to left side) and >> (shift to right side)

```
LDI R20,0b00100010          ;load 0x22 to R20, R20=0010 0010
LDI R20,0b00100010<<1       ;shift R20 by 1 bit, R20=0100 0100 = 0x44, R20*2, C=0
LSL R20,0b00100010<<2       ;shift R20 by 2 bit, R20=1000 1000 = 0x88, R20*2, C=0
LSL R20,0b00100010<<3       ;shift R20 by 3 bit, R20=0001 0000 = 0x10, C=1
```

Using a shift operator to make the port outputs:
To make the output according to the value of R20, we can use the shift operators and the predefined numbers of PORT pins. Ports pins (PA0 ~ PA7) are mapped already in the header files of the microcontrollers as 0 to 7. For instance, PA0 indicates 0 and PA1 indicates 1. Therefore, we can use PA1, PA2, and PA3 instead of 1,2, and 3 for the bitwise shifting operation.

```
LDI R20, (1<<PA5)|(1<<PA1)      ;load 0x22 to R20 to make the output of PA5 and PA1 set
OUT PORTA,R20
LDI R20, (1<<PA6)|(1<<PA2)      ;load 0x44 to R20 to make the output of PA6 and PA2 set
OUT PORTA,R20
LDI R20, (1<<PA7)|(1<<PA3)      ;load 0x88 to R20, make the output of PA7 and PA3 set
OUT PORTA,R20
```

(2) Storing and retrieving data from the program memory location using pointers

Indirect addressing – Pointers
The pointers, X, Y, and Z are consisted of two registers as shown in Figure 1. They are 16 bits registers (8bit x 2) and represent the address of the data space which is 16bits.

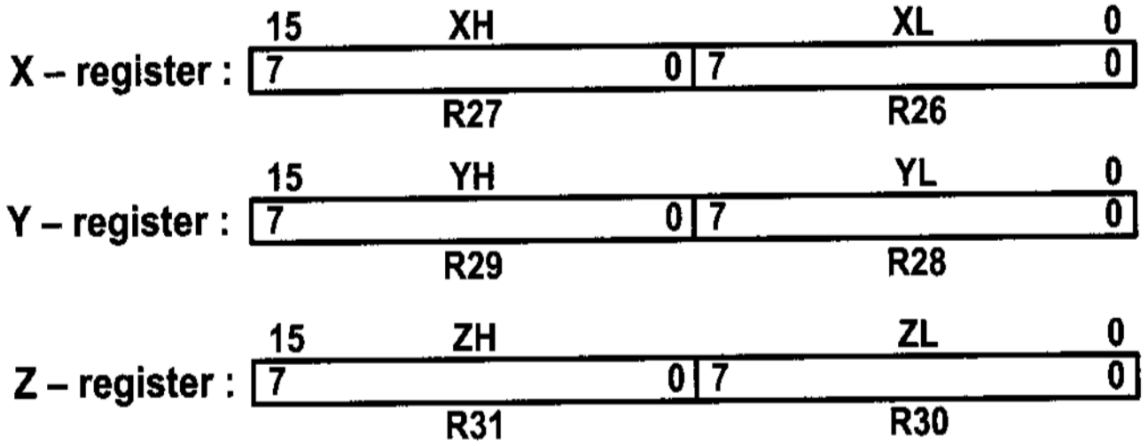


Figure 1. Pointers and their registers for each pointer (textbook)

The assembly instructions to use the pointers are listed in Table 6-5 (from the textbook) and you can use the auto increment in the program to access data in the desired memory location.

Table2. The instructions for pointers (textbook)

Table 6-5: AVR Auto-Increment/Decrement of Pointer Registers for LD Instruction

Instruction		Function
LD	Rn,X	After loading location pointed to by X, the X stays the same.
LD	Rn,X+	After loading location pointed to by X, the X is incremented.
LD	Rn,-X	The X is decremented, then the location pointed to by X is loaded.
LD	Rn,Y	After loading location pointed to by Y, the Y stays the same.
LD	Rn,Y+	After loading location pointed to by Y, the Y is incremented.
LD	Rn,-Y	The Y is decremented, then the location pointed to by Y is loaded.
LDD	Rn,Y+q	After loading location pointed to by Y+q, the Y stays the same.
LD	Rn,Z	After loading location pointed to by Z, the Z stays the same.
LD	Rn,Z+	After loading location pointed to by Z, the Z is incremented.
LD	Rn,-Z	The Z is decremented, then the location pointed to by Z is loaded.
LDD	Rn,Z+q	After loading location pointed to by Z+q, the Z stays the same.

Note: This table shows the syntax for the LD instruction, but it works for all such instructions. The auto-decrement or auto-increment affects the entire 16 bits of the pointer register and has no effect on the status register. This means that pointer register going from FFFF to 0000 will not raise any flag.

(3) Creating look up table in the program memory (ROM) using .DB (.DB: define byte, and .DW: define word) and .ORG

.DB

To define the byte size data in the program memory (ROM), use .DB (define byte) directive. The examples are shown in Table 3.

Table 3. Define byte in the program memory and example instruction lines

Directive	Example instruction line to define data	Data Type
.DB	Data1: .DB 1,2,3	Decimal number
.DB	Data2: .DB 0x24, 0x67	Hexadecimal number
.DB	Data3: .DB 0b10111011	Binary number
.DB	Data4: .DB ‘a’,’b’,’c’	ASCII letters
.DB	Data5: .DB “Task Complete”	ASCII String

.ORG

To define the initial data location of the data set, use the directive .ORG that shows the first location of the data in program memory. However, the address used for .ORG is a half of the physical memory address in the data location due to the flash memory addressing mode is in 32 bits base. As shown in Figure 6-13a and Figure 6-13b (from the textbook), the address is based on 32 bits.

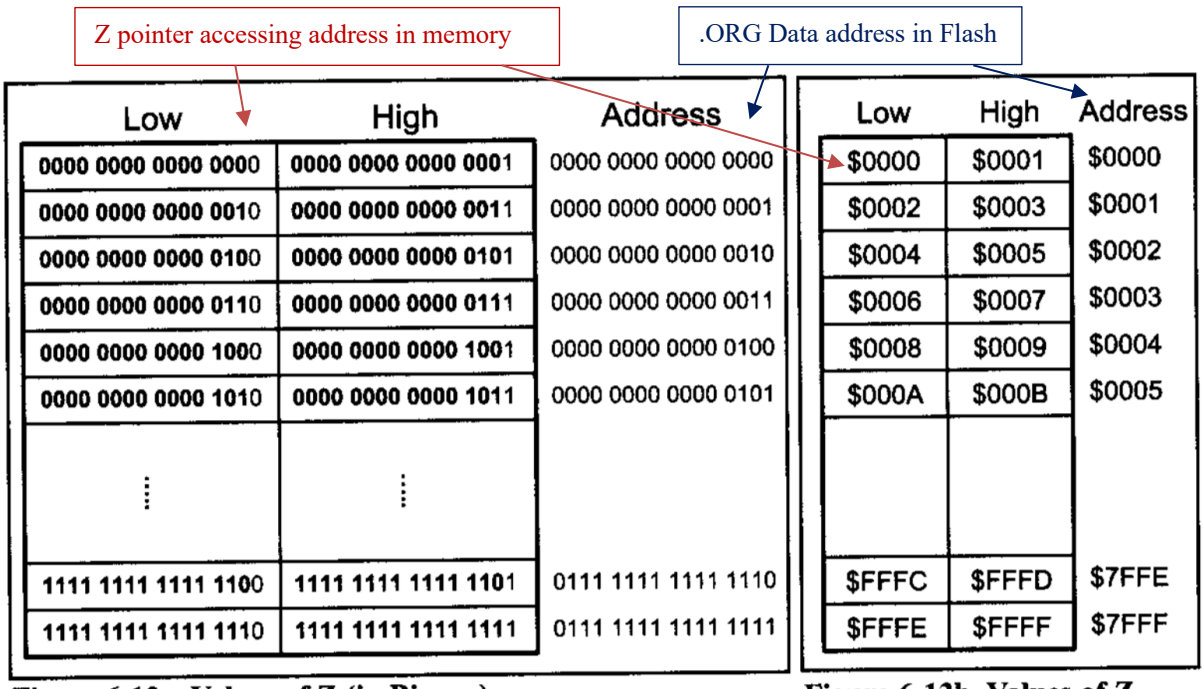


Figure 6-13a. Values of Z (in Binary) **Figure 6-13b. Values of Z**

For example, the data location starts at 0x20 (.ORG 0x10) that can be accessed by the pointer with the memory address 0x20 (Z=0x0020). The actual memory location pointed by Z pointer that is assigned by .ORG is computed by Equation (1) and (2).

$$Z\ address = low\ location = .ORG\ address \ll 1 + 0 \quad (1)$$

$$Z\ address = \ high\ location = .ORG\ address \ll 1 + 1 \quad (2)$$

Example 2:

.ORG address = 0x0002, find Z pointer value in the lower location
Z = 0x0002<<1+0 = 0x0004

.ORG address = 0x0003, find Z pointer value in the higher location
Z = 0x0002<<1+3 = 0x0005

NOTE: The lower and higher locations of Z pointer are not the same as ZL and ZH. Each location is a 16-bit address that is pointed by Z.

To load a value to a GPR from the program memory, use LPM instruction (LPM Rn, Z: load from program memory to Rn, LPM Rn, Z+: load from program memory to Rn and increment the pointer Z by 1). Run the example code below to see how it works.

Example 3:
The code assigns the pointer to 0x0200 memory location in order to access the data in ROM. The data has been defined at the location 0x0200 in memory location (.ORG 0x100) using .DB directive.

```
//Initializing the Z pointer
LDI ZH,HIGH(0x0200) ;higher byte of the address
LDI ZL,LOW(0x0200)  ;lower byte of the address

//Retrieve the data from the program memory
Main:  lpm r16,Z+
       rjmp main

//ASCII letter A, B, C in the program memory location 0x100 (physical address: 0x200)
.ORG 0x0100
Data: .DB 'A','B','C'
```

ACTIVITIES:

ACTIVITY 1

Write an assembly code to use the decimal values, 16, -8, and 128. Then, make the required output using the LEDs outputs (PORTA) and 7-segment display based on the result of the shifting operations and arithmetic operations.

Required operation in the code:

1. Load the decimal numbers, 16, -8, and 128
2. Do arithmetic operations using the assembly instructions for each value 16 and -8. And load the results in GPRs: **R20 = 16 * 4 and R21 = -8*8**

PART 1.

3. Display the first multiplication result (R20 value) on LEDs for 1 second ON (don't care about the 7-seg for now)
4. Turn off LEDs
5. Apply the first shifting operation to the number, 16
6. Compare the first shifting result with the R20 value
7. Use 7-segment display NOW. Display the result as '0' if they are not equal and '1' if they are equal on 7-seg display (see Table 1.2) for 0.5 second ON.
8. Turn off for the 7-segment display for 0.5 second
9. Apply shifting operation continuously until the result of shifting and the arithmetic operation result are equal (7-seg display shows '1')

10. Display each shifting result with 0.5second time delay. Do not care about the LEDs outputs when 7-segment display is running.

PART 2.

11. Display the value of R21 on LEDs for 1 second
12. Turn off LEDs after the 1 second display
13. Apply the first shifting operation to the signed number -8 and compare the result to the value of R21
14. Use 7-segment display NOW. Display the result as '0' if they are not equal and '1' if they are equal on 7-seg display (see Table 1.2) for 0.5 second ON.
15. Turn off for the 7-segment display for 0.5 second
16. Apply shifting operation continuously until the result of shifting and the arithmetic operation result are equal (7-seg display shows '1')
17. Display each shifting result with 0.5second time delay. Do not care about the LEDs outputs when 7-segment display is running.

PART 3.

18. Load the number 128 to R16
19. Display R16 on LEDs for 0.5 second (Do not care about the 7-segment display for NOW)
20. Apply the shifting operation to the number 128 until the shifting result reaches to the number 8 and update R16 with the shifted value. Count the number of shifting (how many times shifting applied until the value reaches the number 8) . Display each shifting result of R16 on LEDs until the last value reaches the number 8
21. Once the value reaches 8, LED display is done. Turn off LEDs.
22. Display the shifting operation counter value on the 7-segment display for 1 second and turn off (Do not care about the LEDs outputs while 7-segment display is running)
    Note: Each display of the shifting result must be shown with 0.5 second time delay
23. Make the code run infinite loop to show the results of the PART 1, 2, and 3 continuously

Table 1.2. Activity 2 Operation Description

Number, N	LEDs	Arithmetic operation, R	7-segment Display
16	Binary of 16	R = N * 4	if R is not equal to shift (N), 0 if R is equal to shift (N), 1
-8	Binary of 80	R = N * 8	
128	Binary of 128 and shifted values until it reaches the number 8	$R = N / 2^n$ (n: count of shifting)	Counted value of shifting operations

Activity 1 Result for the Lab Report:

1. Draw a flowchart for Activity 1. (Hand drawing is not acceptable.)
2. Write the code to complete the operation
3. Make the videos of the result of Activity 1. Attached the links in the report.

ACTIVITY 2

Write a code to access the flash memory allocated data and display the data based on the inputs from the PORTB.

Required operation in the code:

1. Define the given data to the program memory using the .ORG and .DB directives (see Table 2.1)

Data Type	Values	.ORG Program Memory Address	Flash Memory Address
Number	3, 6, 7, 1, 2	0x0100	0x0200
ASCII String	"PS2019"	0x0200	0x0400

2. Access the number data if the PB0 reading is clear. Display each data in binary using LEDs with 0.5second time delay until the input value is changed. (You can use the 7-segment display instead of using LEDs.)
3. Access the ASCII String if the PB0 reading is set. Display each data in binary using LEDs with 0.5second time delay until the input value is changed. (You can use the 7-segment display instead of using LEDs.)

Activity 2 Result for the Lab Report:

1. Draw a flowchart for Activity 2. (Hand drawing is not acceptable.)
2. Write the code to complete the operation
3. Show the result of memory location 0x100 (.ORG 0x0050) and 0x200 (.ORG 0x0100) (screenshot)
4. Make the video of the result of Activity 2. Attached the links in the report.

Extra Credit Activity (10 points)

Do Activity 1 using SERVO (PORTD) instead of using 7-segment display.

SERVO: Use PD2 of the Atmel 324PB XPlained PRO Board

Conditions:

1. Turn SERVO counterclockwise if the shifted value is not matching (PWM 2ms in 20ms total period)
2. Turn SERVO clockwise if the shifted value is matching (PWM 1ms in 20ms total period)

   NOTE: Use time delay to adjust the PWM signal for SERVO (servo control PWM can be slightly varying)

Number, N	LEDs	Arithmetic operation, R	SERVO if R is not equal to shift (N)	SERVO if R is equal to shift (N)
16	Binary of 16	R = N * 4	Counterclockwise (PWM 2ms in 20ms period)	Clockwise(PWM 1ms in 20ms period)
-8	Binary of 80	R = N * 8	Counterclockwise (PWM 2ms in 20ms period)	Clockwise(PWM 1ms in 20ms period)
128	Binary of 128 and shifted values until it reaches the number 8	$R = N / 2^n$ (n: count of shifting)	Counterclockwise (PWM 2ms in 20ms period)	Clockwise(PWM 1ms in 20ms period)

1. No flowchart is required
2. Write the code to complete the operation
3. Make the videos of the result of Extra credit activity. Attached the links in the report.

Lab 8 Report Submission:

Lab 8 consists of two activities. You must include the report cover page, flowcharts, codes, and results (activity 1: video link, activity 2: memory location result and video). **No full lab report is required.**

Total points for this lab: 100 points (Activity 2 - 50 points, Activity 3 – 50 points) and Extra credit activity 10 points

Due date: Tuesday (10/29/2019) 2:59pm, Thursday (1031/2019) 8:59pm

ECE3613 Processor System Laboratory Rubric
Lab #: 8
Section: 001 / 002
Name: _____

Activity	Task	Full Points	Earned Points	Comment
1	Flowchart	10		
	Code	20		Shifting for 16 (10pts), and 80 (10pts)
	Result	20		Video Link
2	Flowchart	10		
	Code	20		Pointer (10pts), program memory data storing (10pts)
	Result – memory values	10		Screenshot
	Result – video	10		Video Link
Extra	Code	5		
	Result	5		Video Link
Total		**100**		