# ECE 3120
# Computer Systems
# Shift and Rotate

Manjeera Jeedigunta

http://blogs.cae.tntech.edu/msjeedigun21

Email: msjeedigun21@tntech.edu

Tel: 931-372-6181, Prescott Hall 120

- Prev:
  - Loops
- Today:
  - Shift and Rotation

# Shift and Rotate Instructions

The 68HCS12 has shift and rotate instructions that apply to a memory location, accumulators A, B and D.  A memory operand must be specified using the extended or index addressing modes.

**Logical Shift**

**Shift Left (Memory,A,B,D): LSL,LSLA,LSLB,LSLD**

**Shift Right (Memory,A,B,D): LSR,LSRA,LSRB,LSRD**

**Arithmetic Shift,** **Similar to a Logical shift, but the sign bit remains unchanged.**

**Shift Left (Memory,A,B,D): ASL,ASLA,ASLB,ASLD**

**Shift Right (Memory,A,B,D): ASR,ASRA,ASRB**

**Cyclic Shift (or Rotation)**

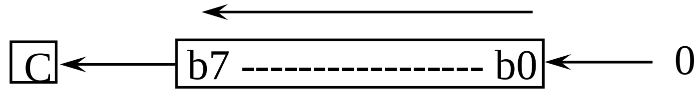
**Left (Memory,A,B): ROL, ROLA,ROLB**

**Right (Memory,A,B): ROR, RORA,RORB**

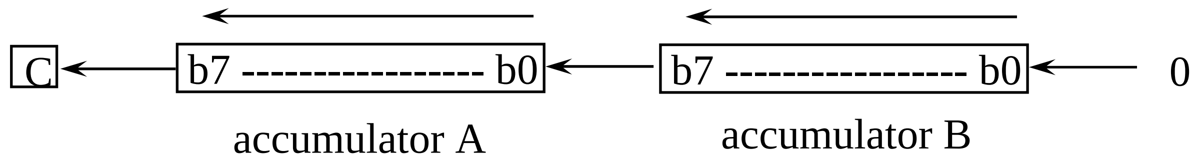
# Logical Shift

One bit falls off and a '0' is shifted in!

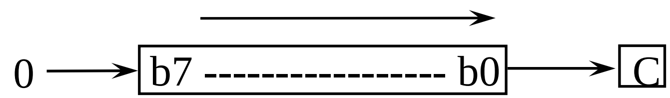
**LSL,LSLA,LSLB**



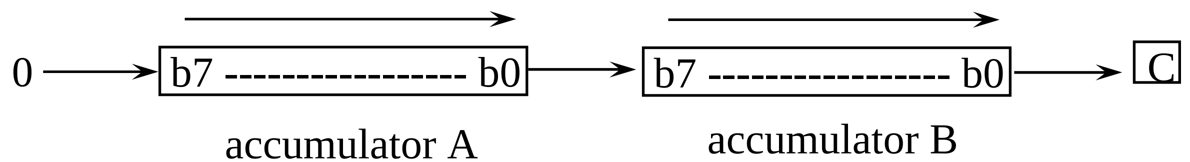

**LSLD**




**LSR,LSRA,LSRB**



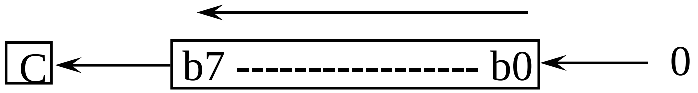

**LSRD**



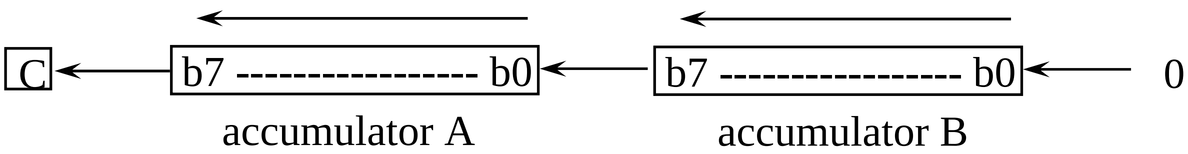
# Arithmetic Shift

:Similar to Logical shift but here the sign bit remains the same

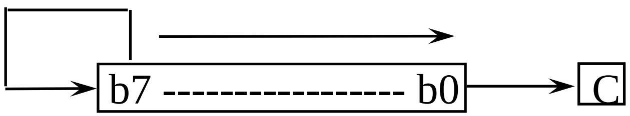
**ASL,ASLA,ASLB**




**ASLD**

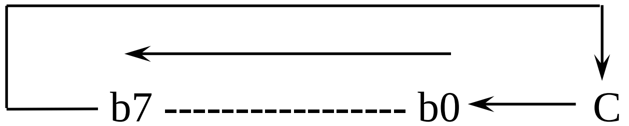



**ASR,ASRA,ASRB**



# Rotation (Cyclic Shift)

**ROL, ROLA,ROLB**




**ROR, RORA,RORB**


A useful link from [The Teacher]@ website.

**Example 2.18** Suppose that [A] = $95 and C = 1. Compute the new values of A and C after the execution of the instruction ASLA.

**Solution:**

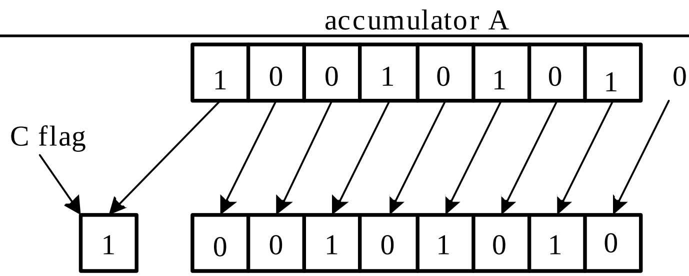



Figure 2.11a Operation of the ASLA instruction

| Original value | New value |
|---|---|
| [A] = 10010101<br>C = 1 | [A] = 00101010<br>C = 1 |

Figure 2.11b Execution result of the ASLA instruction

**Example 2.19** Suppose that m[$800] = $ED and C = 0. Compute the new values of m[$800] and the C flag after the execution of the instruction ASR $800.

**Solution:**

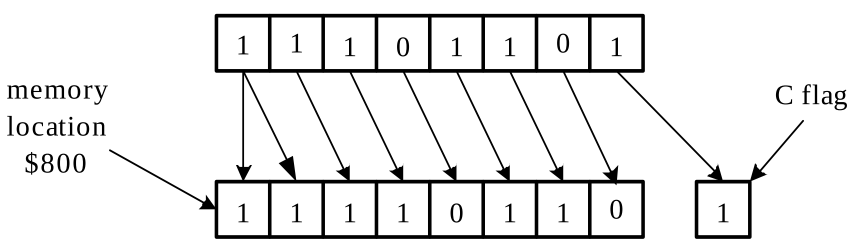



Figure 2.12a Operation of the ASR $800 instruction

| Original value | New value |
|---|---|
| [$800]=11101101<br>C = 0 | [$800]=11110110<br>C = 1 |

Figure 2.12b Result of the ASR $800 instruction

**Example 2.20** Suppose that m[$800] = $E7 and C = 1. Compute the new contents of m[$800] and the C flag after the execution of the instruction LSR $800.

**Solution:**

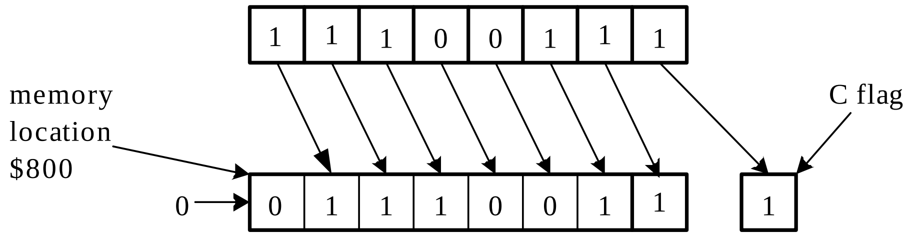



Figure 2.13a Operation of the LSR $800 instruction

| Original value | New value |
|---|---|
| [$800] = 11100111<br>C = 1 | [$800] = 01110011<br>C = 1 |

Figure 2.13b Execution result of LSR $800

**Example 2.21** Suppose that [B] = $BD and C = 1. Compute the new values of B and the C flag after the execution of the instruction ROLB.

**Solution:**

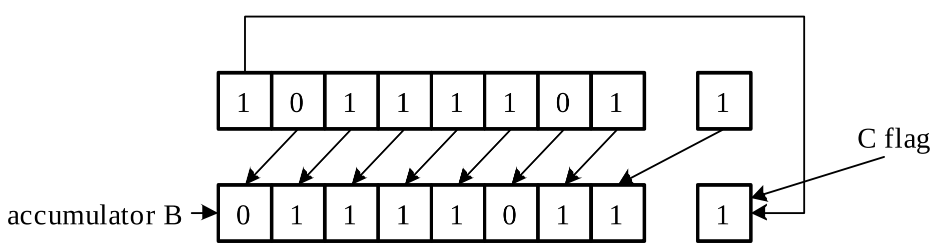



Figure 2.14a Operation of the instruction ROLB

| Original value | New value |
|---|---|
| [B] = 10111101<br>C = 1 | [B] = 01111011<br>C = 1 |

Figure 14b. Execution result of ROLB

# Shift a Multi-byte Number

## *For shifting right*

1. The bit 7 of each byte will receive the bit 0 of its immediate left byte with the exception of the most significant byte which will receive a 0.
2. Each byte will be shifted to the right by 1 bit. The bit 0 of the least significant byte will be lost.

Suppose there is a k-byte number that is stored at *loc* to *loc+k-1*.

## *Method for shifting right*

Step 1: Shift the byte at *loc* to the right one place.
Step 2: Rotate the byte at *loc+1* to the right one place.
Step 3: Repeat Step 2 for the remaining bytes.

## *For shifting left*

1. The bit 0 of each byte will receive the bit 7 of its immediate right byte with the exception of the least significant byte which will receive a 0.
2. Each byte will be shifted to the left by 1 bit. The bit 7 of the most significant byte will be lost.

Suppose there is a k-byte number that is stored at *loc* to *loc+k-1*.

## *Method for shifting left*

Step 1: Shift the byte at *loc+k-1* to the left one place.
Step 2: Rotate the byte at *loc+K-2* to the left one place.
Step 3: Repeat Step 2 for the remaining bytes.

**Example 2.24** Write a program to shift the 32-bit number stored at $1000-$1003 to the right four places.

```
            ldab    #4          ;set up the loop count
            ldx     #$1000
again       lsr     0,x
            ror     1,x
            ror     2,x
            ror     3,x
            dbne    b,again
```

# Boolean Logic Instructions

- Changing a few bits are often done in I/O applications.
- Boolean logic operation can be used to change a few I/O port pins easily.

Table 2.8 Summary of Booleran logic instructions

| Mnemonic | Function | Operation |
|---|---|---|
| ANDA <opr> | AND A with memory | $A \leftarrow (A) \bullet (M)$ |
| ANDB <opr> | AND B with memory | $B \leftarrow (B) \bullet (M)$ |
| ANDCC <opr> | AND CCR with memory (clear CCR bits) | $CCR \leftarrow (CCR) \bullet (M)$ |
| EORA <opr> | Exclusive OR A with memroy | $A \leftarrow (A) \oplus (M)$ |
| EORB <opr> | Exclusive OR B with memory | $B \leftarrow (B) \oplus (M)$ |
| ORAA <opr> | OR A with memory | $A \leftarrow (A) + (M)$ |
| ORAB <opr> | OR B with memory | $B \leftarrow (B) + (M)$ |
| ORCC <opr> | OR CCR with memory | $CCR \leftarrow (CCR) + (M)$ |
| CLC | Clear C bit in CCR | $C \leftarrow 0$ |
| CLI | Clear I bit in CCR | $I \leftarrow 0$ |
| CLV | Clear V bit in CCR | $V \leftarrow 0$ |
| COM <opr> | One's complement memory | $M \leftarrow \$FF - (M)$ |
| COMA | One's complement A | $A \leftarrow \$FF - (A)$ |
| COMB | One's complement B | $B \leftarrow \$FF - (B)$ |
| NEG <opr> | Two's complement memory | $M \leftarrow \$00 - (M)$ |
| NEGA | Two's complement A | $A \leftarrow \$00 - (A)$ |
| NEGB | Two's complement B | $B \leftarrow \$00 - (B)$ |

**<u>Example AND</u>**

```
Ldaa   $56
Anda   #$0F
Staa   $56
```

# Bit Test and Manipulate Instruction

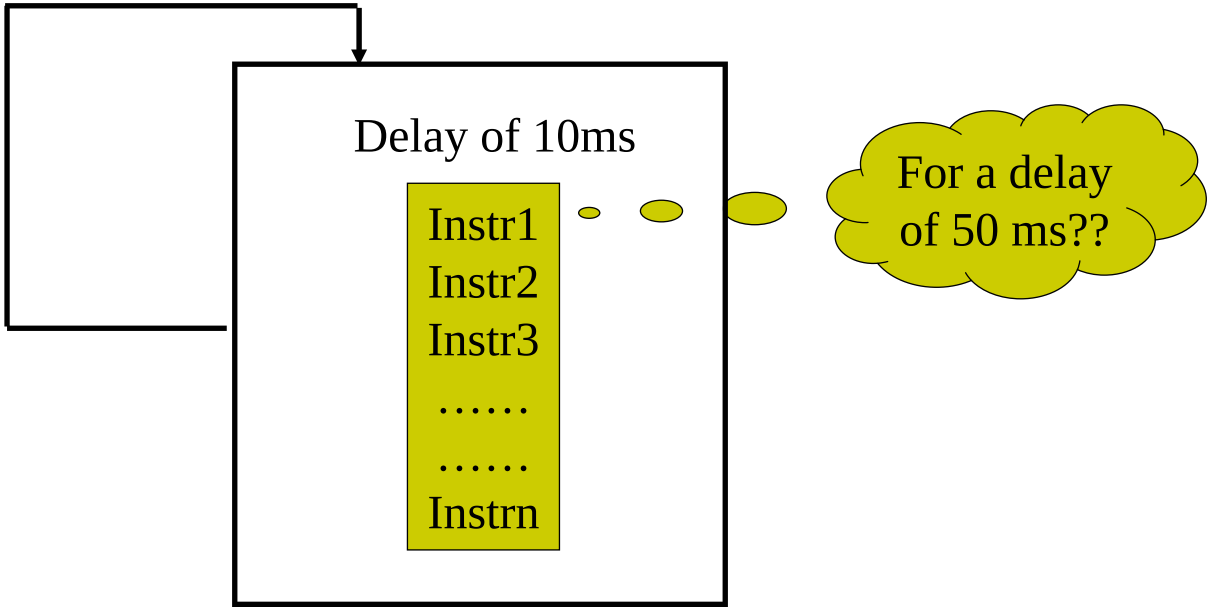
- Used to either test or change the values of certain bits in a given number
- Bclr,bita,bitb,bset
- Examples:
  - Bclr 0,x,$81 → (10000001)
    - Clears MSB and LSB, pointed by memory location 0,x
  - Bita #$44 → (01000100)
    - Tests bit 6 & 2 of A and updates Z,N flags accordingly
  - Bitb #$22 → (00100010)
    - Tests bit 5 & 1 of B and updates Z,N flags accordingly
  - Bset 0,y,$33 (00110011)
    - Sets bits 5,4,1,0 of the memory location pointed by 0,y

# Program Execution Time

- The 68HCS12 uses the E clock (**ECLK**) as a timing reference.
- There are many applications that require the generation of time delays.

The creation of a time delay involves two steps:

1. Select a sequence of instructions that takes a certain amount of time to execute.
2. Repeat the selected instruction sequence for an appropriate number of times.



For example, the instruction sequence to the right takes 40 E cycles to execute. By repeating this instruction sequence certain number of times, different time delay can be created.

Assume that the E frequency of 68HCS12 is 24 MHz and hence its **clock period is 41.25 ns**. Therefore the instruction sequence to the right will take **1.667 μs** to execute.

```
loop    psha            ; 2 E cycles
        pula            ; 3 E cycles
        psha
        pula
        psha
        pula
        psha
        pula
        psha
        pula
        psha
        pula
        psha
        pula
        nop             ; 1 E cycle
        nop             ; 1 E cycle
        dbne   x,loop   ; 3 E cycles
```

**Example 2.25** Write a program loop to create a delay of 100 ms.

---

**Solution:** A delay of 100 ms can be created by repeating the previous loop 60000 times.

```
        ldx     #60000;
loop    psha              ; 2 E cycles
        pula              ; 3 E cycles
        psha
        pula
        psha
        pula
        psha
        pula
        psha
        pula
        psha
        pula
        psha
        pula
        nop               ; 1 E cycle
        nop               ; 1 E cycle
        dbne    x,loop    ; 3 E cycles
```

# Chapter Review

- Assembly Language Program Structure:
  - Label, operation, operand, comment
- Directives: end,org,db,ds,fill…
- Flow chart
- Arithmetic
- Loops, branch instructions
- Shift and rotate
- Boolean logic
- Bit test and manipulate
- Program execution time

# Now, you should be able to:

- Allocate memory blocks, define constants, and create a message using assembler directives
- Write assembly programs to perform simple arithmetic operations
- Write loops to perform repetitive operations
- Use loops to creat time delays
- Use boolean and bit manipulation instructions to perform bit field operations.

- Next:
  - Chapter 3