CpE 390: Microprocessor Systems

Lecture Note 3

68HC12 Assembly Programming (2)

Program Loops

Types of program loops: *finite* and *infinite* loops

Looping mechanisms:

1. **do** *statement S* **forever**

2. **For** i = *n1* **to** *n2* **do** *statement S* or **For** i = *n2* **downto** *n1* **do** *statement S*

3. **While** *C* **do** *statement S*

4. **Repeat** *statement S* **until** *C*

Program loops are implemented by using the conditional branch instructions and the execution of these instructions depends on the contents of the CCR register.

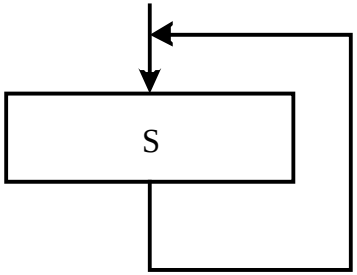


Figure 2.4 An infinite loop

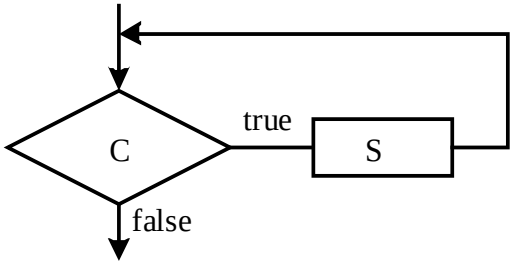
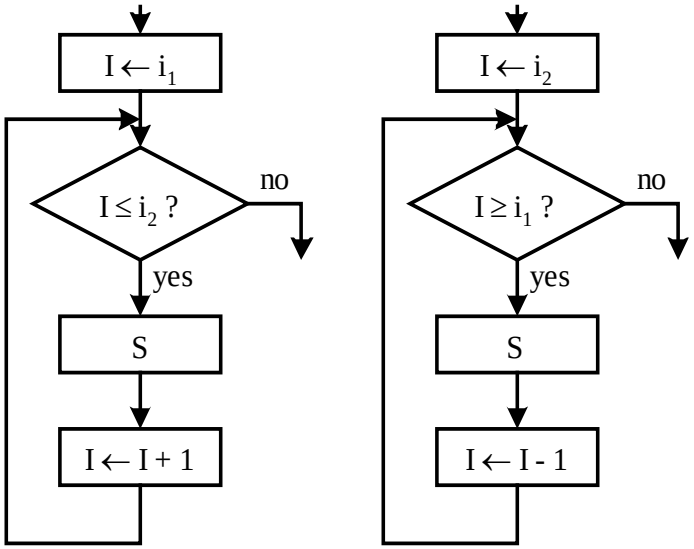


Figure 2.6 The While ... Do looping construct



(a) For $I = i_1$ to i_2 DO S (b) For $I = i_2$ downto i_1 DO S

Figure 2.5 For looping construct

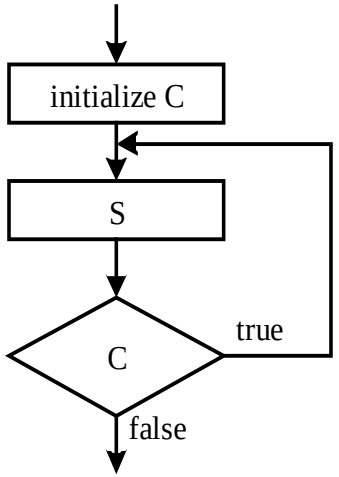


Figure 2.7 The Repeat ... Until looping construct

Condition Code Register

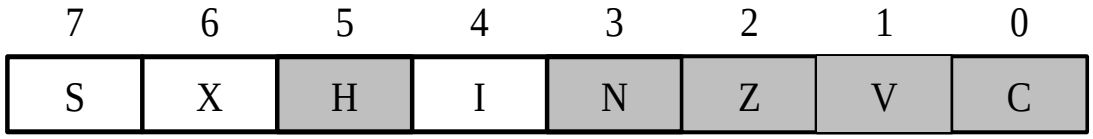


Figure 2.8 Condition code register

S – Stop disable
X – Interrupt mask
I – Interrupt mask

Example:	N	Z	V	C
CLRA	0	1	0	0
ADDA #$20	0	0	0	0
SUBA #$30	1	0	0	1
TSTA	1	0	0	0
CMPA #$F0	0	1	0	0

Branch Instructions

Four **types** of branch instructions:

- **Unary (unconditional) branch:** always execute
- **Simple branches:** branch is taken when a specific bit of CCR is in a specific status
- **Unsigned branches:** branches are taken when a comparison or test of unsigned numbers results in a specific combination of CCR bits
- **Signed branches:** branches are taken when a comparison or test of signed quantities results in a specific combination of CCR bits

Two **categories** of Branches

- **Short Branches:** in the range of -128 ~ +127 bytes
- **Long Branches:** in the range of 64KB

Table 2.2 Summary of short branch instructions

Unary Branches		
Mnemonic	Function	Equation or Operation
BRA	Branch always	1 = 1
BRN	Branch never	1 = 0
Simple Branches		
Mnemonic	Function	Equation or Operation
BCC	Branch if carry clear	C = 0
BCS	Branch if carry set	C = 1
BEQ	Branch if equal	Z = 1
BMI	Branch if minus	N = 1
BNE	Branch if not equal	Z = 0
BPL	Branch if plus	N = 0
BVC	Branch if overflow clear	V = 0
BVS	Branch if overflow set	V = 1
Unsigned Branches		
Mnemonic	Function	Equation or Operation
BHI	Branch if higher	C + Z = 0
BHS	Branch if higher or same	C = 0
BLO	Branch if lower	C = 1
BLS	Branch if lower or same	C + Z = 1
Signed Branches		
Mnemonic	Function	Equation or Operation
BGE	Branch if greater than or equal	$N \oplus V = 0$
BGT	Branch if greater than	$Z + (N \oplus V) = 0$
BLE	Branch if less than or equal	$Z + (N \oplus V) = 1$
BLT	Branch if less than	$N \oplus V = 1$

Table 2.3 Summary of long branch instructions

Unary Branches		
Mnemonic	Function	Equation or Operation
LBRA	Long branch always	1 = 1
LBRN	Long branch never	1 = 0
Simple Branches		
Mnemonic	Function	Equation or Operation
LBCC	Long branch if carry clear	C = 0
LBCS	Long branch if carry set	C = 1
LBEQ	Long branch if equal	Z = 1
LBMI	Long branch if minus	N = 1
LBNE	Long branch if not equal	Z = 0
LBPL	Long branch if plus	N = 0
LBVC	Long branch if overflow is clear	V = 0
LBVS	Long branch if overflow set	V = 1
Unsigned Branches		
Mnemonic	Function	Equation or Operation
LBHI	Long branch if higher	C + Z = 0
LBHS	Long branch if higher or same	C = 0
LBLO	Long branch if lower	C = 1
LBLS	Long branch if lower or same	C + Z = 1
Signed Branches		
Mnemonic	Function	Equation or Operation
LBGE	Long branch if greater than or equal	$N \oplus V = 0$
LBGT	Long branch if greater than	$Z + (N \oplus V) = 0$
LBLE	Long branch if less than or equal	$Z + (N \oplus V) = 1$
LBLT	Long branch if less than	$N \oplus V = 1$

Compare and Test Instructions

- Condition flags need to be set up before conditional branch instruction should be executed.
- The 68HC12 provides a group of instructions for testing the condition flags.

Table 2.4 Summary of compare and test instructions

Compare instructions		
Mnemonic	Function	Operation
CBA	Compare A to B	(A) - (B)
CMPA	Compare A to memory	(A) - (M)
CMPB	Compare B to memory	(B) - (M)
CPD	Compare D to memory	(D) - (M:M+1)
CPS	Compare SP to memory	(SP) - (M:M+1)
CPX	Compare X to memory	(X) - (M:M+1)
CPY	Compare Y to memory	(Y) - (M:M+1)
Test instructions		
Mnemonic	Function	Operation
TST	Test memory for zero or minus	(M) - $00
TSTA	Test A for zero or minus	(A) - $00
TSTB	Test B for zero or minus	(B) - $00

Loop example:

```
        LDAA  G2
        CMPA  G1
        BLS  next
        ;
        BRA continue
Next:   ;
Continue: ;
```

Loop Primitive Instructions

- 68HC12 provides a group of instructions that either decrement or increment a loop count to determine if the looping should be continued.
- The range of the branch is from $80 (-128) to $7F (+127).

Table 2.5 Summary of loop primitive instructions

Mnemonic	Function	Equation or Operation
DBEQ cntr, rel	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) - 1 If (counter) = 0, then branch else continue to next instruction
DBNE cntr, rel	Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) - 1 If (counter) ≠ 0, then branch else continue to next instruction
IBEQ cntr, rel	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) + 1 If (counter) = 0, then branch else continue to next instruction
IBNE cntr, rel	Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) + 1 If (counter) ≠ 0, then branch else continue to next instruction
TBEQ cntr, rel	Test counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	If (counter) = 0, then branch else continue to next instruction
TBNE cntr, rel	Test counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	If (counter) ≠ 0, then branch else continue to next instruction

Note. 1. **cntr** is the loop counter and can be accumulator A, B, or D and register X, Y, or SP.
2. **rel** is the relative branch offset and is usually a label

Example 2.14 Write a program to add an array of N 8-bit numbers and store the sum at memory locations $800~$801. Use the **For** i = n1 **to** n2 **do** looping construct.

Solution:

```
N       equ     20
        org     $800
sum     rmb     2
i       rmb     1
        org     $1000
        ldaa    #0
        staa    i           ; counter index <-0
        staa    sum         ; sum ← 0
        staa    sum+1       ;       “
loop    ldab    i           ; counter index
        cmpb    #N          ; is i = N?
        beq     done
        ldx     #array
        abx                 ; (b) +(x)->(x) compute the address of array[i]
        ldab    0,X         ; place array[i] in B
        ldy     sum         ; place sum in Y
        aby                 ; sum <- sum + array[i]
        sty     sum         ;  update sum
```

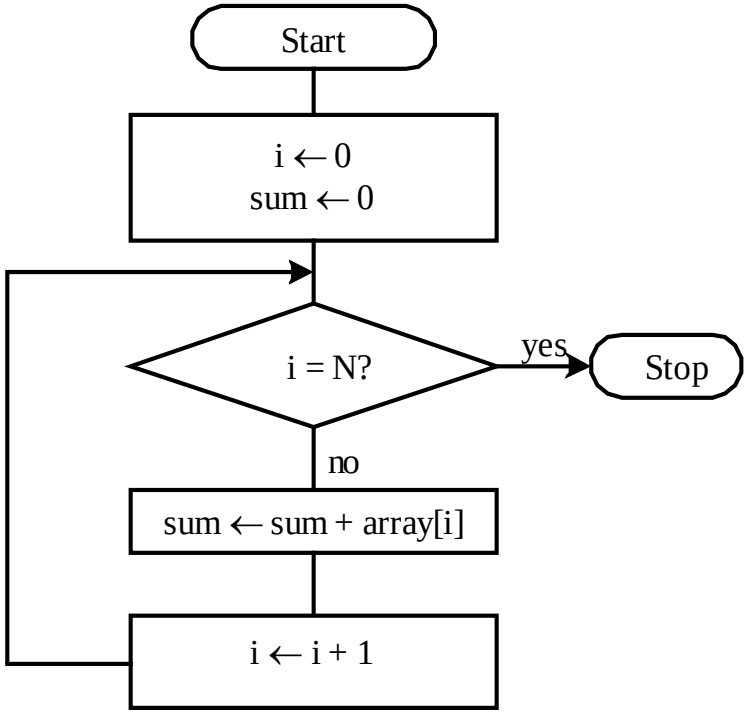


Figure 2.9 Logic flow of example 2.14

```
        inc     i               ; increment the loop count by 1
        bra     loop
done    swi                     ; software interrupt
array   dc.b    1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
        end
```

Example 2.15 Write a program to find the maximum element from an array of N 8-bit elements using the **repeat S until C** looping construct.

Solution:

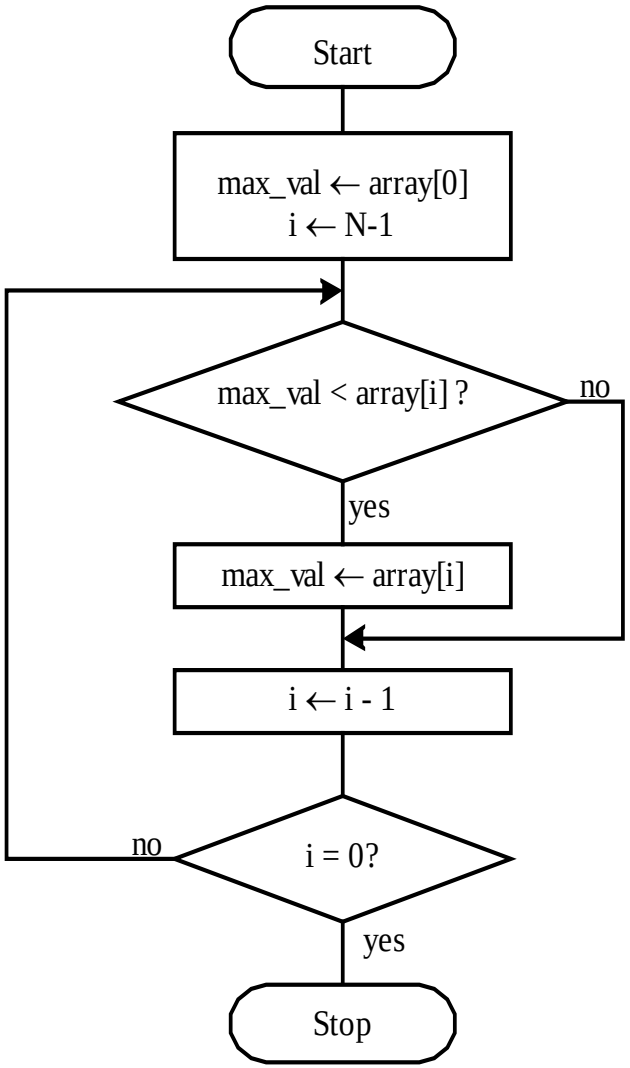


Figure 2.10 Logic flow of example 2.15

```
N         equ     20
          org     $800
max_val   ds.b    1

          org     $1000
          ldaa    array       ; set array[0] as the temporary max max
          staa    max_val     ;          “
          ldx     #array+N-1  ; start from the end of the array
          ldab    #N-1        ; set loop count to N - 1
loop      ldaa    max_val
          cmpa    0,x
          bge     chk_end
          ldaa    0,x
          staa    max_val
chk_end   dex
          dbne    b,loop      ; finish all the comparison yet?
forever   bra     forever

array     db      1,3,5,6,19,41,53,28,13,42,76,14
          db      20,54,64,74,29,33,41,45
          end
```

Special Conditional Branch Instructions

[<label>] BRCLR (opr) (msk) (rel) [<comment>]
[<label>] BRSET (opr) (msk) (rel) [<comment>]

perform bitwise logical AND of the contents of the memory location and the mask, Branch if the result is zero (for BRCLR) or one (for BRSET).

where

opr specifies the memory location to be checked and must be specified using either the direct or index addressing mode.

msk is an 8-bit mask that specifies the bits of the memory location to be checked. The bits of the memory byte to be checked correspond to those bit positions that are 1s in the mask.

rel is the branch offset and is specified in the relative mode.

Special Conditional Branch Instructions

- Example 1:

```
here  brclr   $86, $20, here
      ldd $30
```

the 68HC12 will continue to execute the first instruction if the bit 1 of the memory location at $86 is 0. otherwise, the next instruction will be executed.

- Example 2:

```
loop          inc  count
              …
              brset  $66,$f0,loop
              …
```

the branch will be taken if the most significant four bits at $66 are all ones.

Example 2.17 Write a program to compute the number of elements that are divisible by 4 in an array of N 8-bit elements. Use the **repeat S until C** looping construct.

Solution: A number divisible by 4 would have the least significant two bits equal 0s.

```
N       equ     20
        org     $800
total   ds.b    1

        org     $1000
        clr     total           ; initialize total to 0
        ldx     #array
        ldab    #N  ; use B as the loop count
loop    brclr   0,x,$03,yes     ; check bits 1 and 0
        bra     chkend
yes     inc     total
chkend  inx
        dbne    b,loop
forever bra     forever
array   db      2,3,4,8,12,13,19,24,33,32,20,18,53,52,80,82,90,94,100,102
        end
```

Instructions for Variable Initialization

We often need to initialize a variable to zero when writing a program.

1. [<label>] CLR opr [<comment>]

   where **opr** is specified using the *extended* or *index* addressing modes. The specified memory location is cleared.

2. [<label>] CLRA [<comment>]

   Accumulator A is cleared to 0

3. [<label>] CLRB [<comment>]

   Accumulator B is cleared to 0

Shift and Rotate Instructions

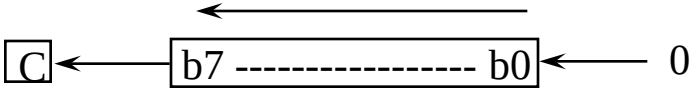
The 68HC12 has shift and rotate instructions that apply to a memory location, accumulators A, B and D. A memory operand must be specified using the extended or index addressing modes.

- are useful for bit operations
- can be used to speed up integer multiplication and division operations if one of the operands is a power of 2
- can shift/rotate the operands by 1 bit at a time

There are three 8-bit arithmetic shift left instructions:

```
[<label>]   ASL  opr   [<comment>]      -- memory location opr is shifted left one place
[<label>]   ASLA      [<comment>]      -- accumulator A is shifted left one place
[<label>]   ASLB      [<comment>]      -- accumulator B is shifted left one place
```

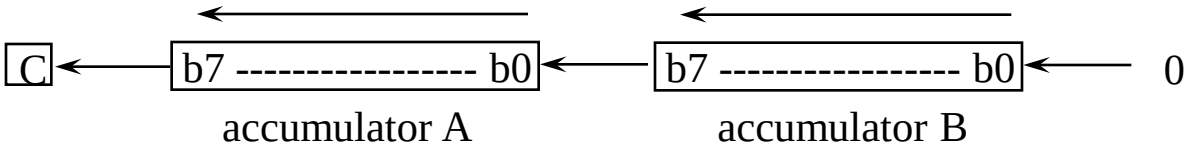
The operation is



The 68HC12 has one 16-bit arithmetic shift left instruction:

[<label>] ASLD [<comment>]

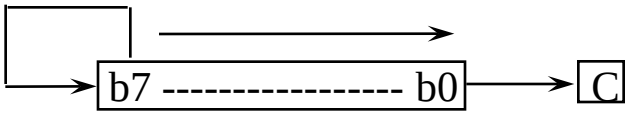
The operation is



The 68HC12 has arithmetic shift right instructions that apply to a memory location and accumulators A and B.

[<label>] ASR opr [<comment>] -- memory location **opr** is shifted right one place
[<label>] ASRA [<comment>] -- accumulator A is shifted right one place
[<label>] ASRB [<comment>] -- accumulator B is shifted right one place

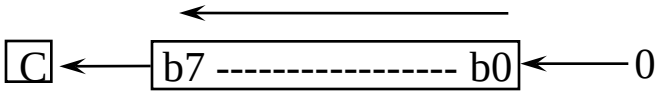
The operation is



The 68HC12 has logical shift left instructions that apply to a memory location and accumulators A and B.

[<label>]	LSL opr	[<comment>]	-- memory location **opr** is shifted left one place
[<label>]	LSLA	[<comment>]	-- accumulator A is shifted left one place
[<label>]	LSLB	[<comment>]	-- accumulator B is shifted left one place

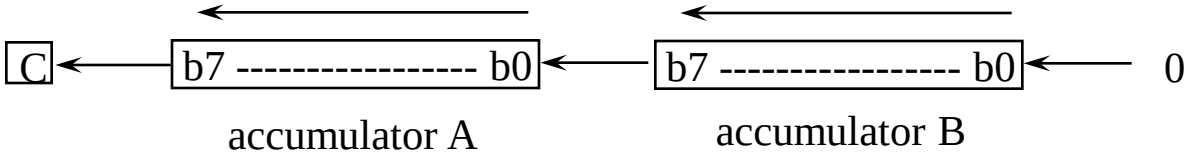
The operation is



The 68HC12 has one 16-bit logical shift left instruction:

[<label>] LSLD [<comment>]

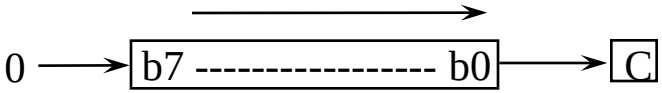
The operation is



The 68HC12 has three logical shift right instructions that apply to 8-bit operands.

[<label>]	LSR opr	[<comment>]	-- memory location **opr** is shifted right one place
[<label>]	LSRA	[<comment>]	-- accumulator A is shifted right one place
[<label>]	LSRB	[<comment>]	-- accumulator B is shifted right one place

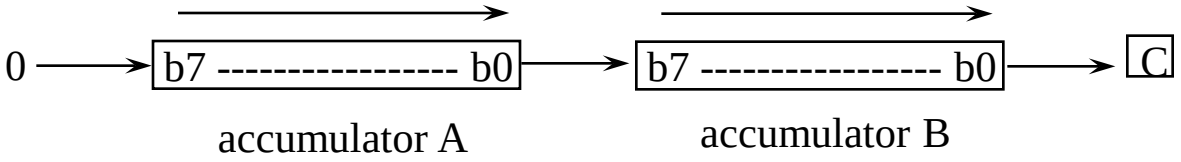
The operation is



The 68HC12 has one 16-bit logical shift right instruction:

[<label>] LSRD [<comment>]

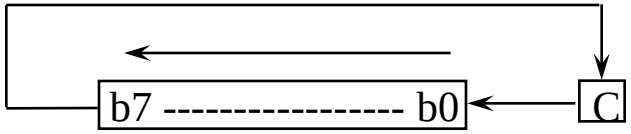
The operation is



The 68HC12 has three rotate left instructions that operate on 9-bit operands.

[<label>]	ROL opr	[<comment>]	-- memory location **opr** is rotated left one place
[<label>]	ROLA	[<comment>]	-- accumulator A is rotated left one place
[<label>]	ROLB	[<comment>]	-- accumulator B is rotated left one place

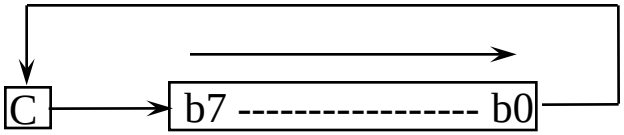
The operation is



The 68HC12 has three rotate right instructions that operate on 9-bit operands.

[<label>]	ROR opr	[<comment>]	-- memory location **opr** is rotated right one place
[<label>]	RORA	[<comment>]	-- accumulator A is rotated right one place
[<label>]	RORB	[<comment>]	-- accumulator B is rotated right one place

The operation is



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:

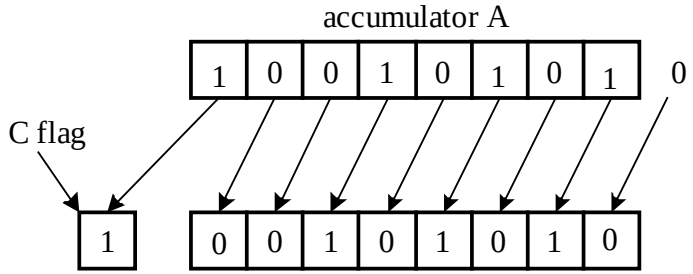


Figure 2.11a Operation of the ASLA instruction

Original value	New value
[A] = 10010101 C = 1	[A] = 00101010 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:

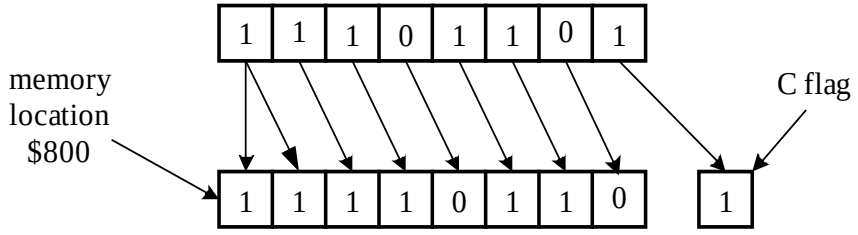


Figure 2.12a Operation of the ASR $800 instruction

Original value	New value
[$800]=11101101 C = 0	[$800]=11110110 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:

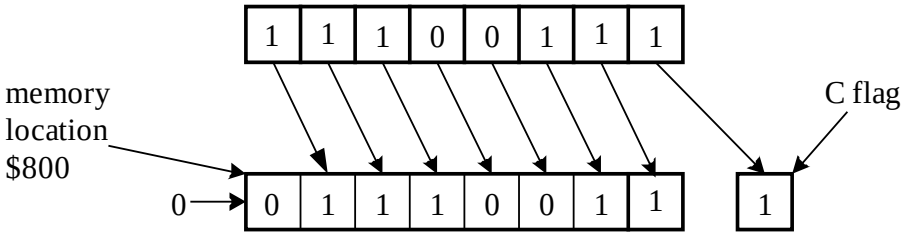


Figure 2.13a Operation of the LSR $800 instruction

Original value	New value
[$800] = 11100111 C = 1	[$800] = 01110011 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:

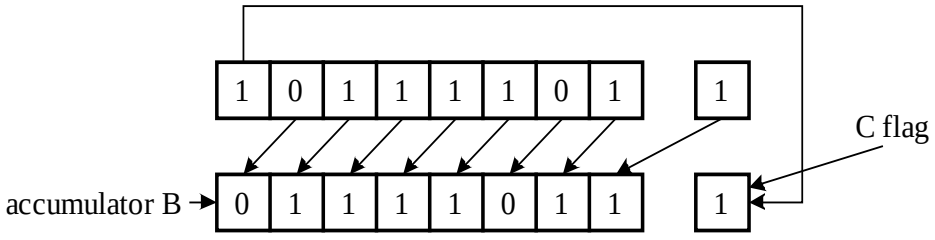


Figure 2.14a Operation of the instruction ROLB

Original value	New value
[B] = 10111101 C = 1	[B] = 01111011 C = 1

Figure 14b. Execution result of ROLB

Example 2.22 Suppose that [A] = $BE and C = 1. Compute the new values of mem[$00] after the execution of the instruction RORA.

Solution:

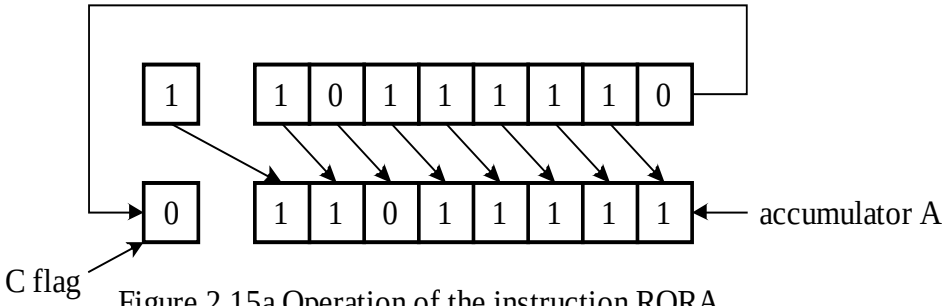


Figure 2.15a Operation of the instruction RORA

Original value	New value
[A] = 10111110 C = 1	[A] = 11011111 C = 0

Figure 2.15b Execution result of RORA

Example 2.23 Write a program to count the number of 0s in the 16-bit number stored at $800-$801 and save the result in $805.

Solution:

* The 16-bit number is shifted to the right 16 time.
* If the bit shifted out is a 0 then increment the 0s count by 1.

```
            org   $800
            db    $23,$55          ; test data
            org   $805
zero_cnt    rmb   1
lp_cnt      rmb   1
            org   $1000
            clr   zero_cnt         ; initialize the 0s count to 0
            ldaa  #16
            staa  lp_cnt
            ldd   $800             ; place the number in D
loop        lsrd                   ; shift the lsb of D to the C flag
            bcs   chkend           ; branch if C flag is 1
            inc   zero_cnt         ; increment 0s count
chkend      dec   lp_cnt           ;
            bne   loop             ; check if we tested all 16 bit.
forever     bra   forever
            end
```

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 $820-$823 to the right four places.

Solution: the most significant to the least significant bytes are stored at $800-$803.

```
        ldab    #4          ; set up the loop count
        ldx     #$820       ; use X as the pointer to the leftmost byte
again   lsr     0,X
        ror     1,X
        ror     2,X
        ror     3,X
        dbne    b,again
        end
```

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

Program Execution Time

- The 68HC12 uses the E clock as a timing reference.
- The frequency of the E clock is half of that of the crystal oscillator.
- 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 on the following slide takes 40 E cycles to execute. By repeating this instruction sequence certain number of times, any time delay can be created.

Assume that the 68HC12 runs under a crystal oscillator with a frequency of 16 MHz, then the E frequency is 8 MHz and hence its clock period is 125 ns. Therefore the instruction sequence on the next page will take 5 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 20000 times.

The following instruction sequence creates a delay of 100 ms.

```
        ldx     #20000
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 2.26 Write an instruction sequence to create a delay of 10 seconds.

Solution: By repeating the previous instruction sequence 100 times, we can create a delay of 10 seconds.

```
                ldab    #100
out_loop        ldx     #20000
in_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,in_loop ; 3 E cycles
                dbne    b,out_loop ; 3 E cycles
```

Don’t get up

- Lecture 4 now

Homework #3

- See course website: http://390.revan.us
  - click homework tab
- Please submit a hard copy