ECET 310-001
Chapter 2, Part 3 of 3

W. Barnes, 9/2006, rev'd. 10/07

Ref. Huang, Han-Way, *The HCS12/9S12: An Introduction to Software and Hardware Interfacing,* Thomson/Delmar.

In This Set of Slides:

1. **Bit condition branch instructions**
2. **Shift and rotate instructions**
3. **Boolean logic instructions**
4. **Clocks and time delays**

Bit Condition Branch Instructions

[<label>] **brclr (opr),(msk),(rel)** [<comment>]
[<label>] **brset (opr),(msk),(rel)** [<comment>]

where

opr specifies the memory location to be checked and must be specified using either the direct, extended, 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 the branch offset specified in the 8-bit relative mode, usually with a label

For example:

```
loop    inc  count
        …
        brclr  $66,$e0,loop   ;$E0 = %1110 0000, branches if all three upper bits are 0's
        …
```

<u>**BOTTOM LINE:**</u>

for <u>brclr</u>, put 1's in bits where you are looking for 0's and for <u>brset</u>, put 1's where you are looking for 1's

Bit Condition Branch Instructions cont'd

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 two least significant bits both 0.

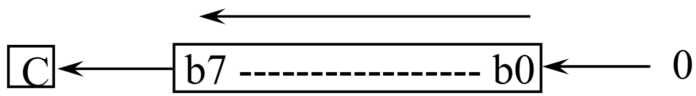
```
N       equ     10
        org     $1500
total   rmb     1
        org     $2000
        clr     total       ; initialize total to 0
        ldx     #array
        ldab    #N          ; use B as the loop count
loop    brclr   0,x,$03,yes         ; check bits 1 and 0 of M[x] for zeros
        bra     chkend      ;unconditional branch to chkend
yes     inc     total
chkend  inx
        dbne    b,loop
        swi
array   db      2,3,4,8,12,13,19,24,33,32
        end
```

Discuss: is this really **repeat S until C** looping ?

Shift and Rotate Instructions

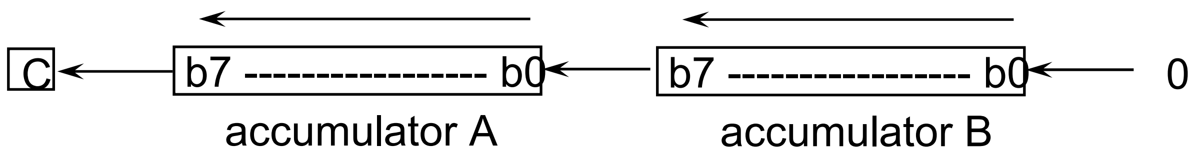
- **Three 8-bit arithmetic shift left instructions:**

[<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



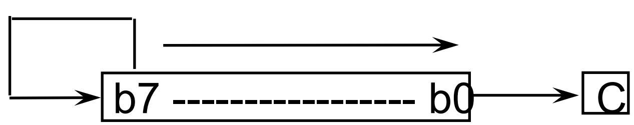
- **One 16-bit arithmetic shift left instruction:**

[<label>] **asld** [<comment>]



- **Three arithmetic shift right instructions (no 16 bit asr instruction):**

[<label>] **arl 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

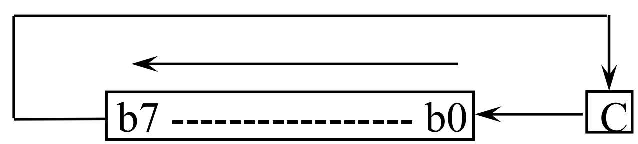


Shift and Rotate Instructions Cont'd.

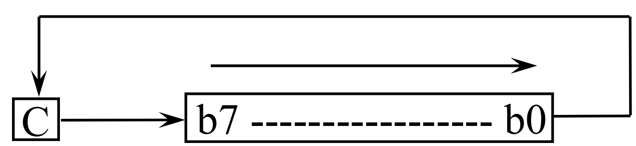
- Logical shift instructions
  - Shift left instructions (lsl opr, lsla, lslb,lsld) perform identical operation as arithmetic shifts left
  - Shift right instructions (lsr opr, lsra, lsrb, lsrd) are the same as the arithmetic shifts right EXCEPT a 0 is shifted into the msb and there is an lsrd (as opposed to asr which has no asrd)

- Note that the rotate instructions, unlike shift, form a LOOP and no bits are lost

- Rotate instructions
  - rol opr, rola, rolb



  - ror opr, rora, rorb



Shift and Rotate Instructions Cont'd.

Examples: Fill in shaded boxes

Ex.	Instruction	Initial values	Final Values
(2.18)	**asla**	**[A] = $95, C = 1**	
(2.19)	**asr $800**	**m[$800] = $ED, C = 0**	
(2.20)	**lsr $1000**	**m[$1000] = $E7, C = 1**	
(2.21)	**rolb**	**[B] = $BD, C = 1**	
(2.22)	**rora**	**[A] = $BE and C = 1**	

Shift and Rotate Instructions Cont'd.

Example 2.23 Write a program to count the number of 0s in the 16-bit number stored at $1000-$1001 and save the result in $1005.

Algorithm: The 16-bit number is shifted to the right 16 times and if the bit shifted out is a 0 then increment the 0s count by 1.

```
            org     $1000
            db      $23,$55     ; test data
            org     $1005
zero_cnt    rmb     1
lp_cnt      rmb     1
            org     $1500
            clr     zero_cnt    ; initialize the 0's count to 0
            ldaa    #16         ;initialize the
            staa    lp_cnt      ;                 loop count
            ldd     $1000       ; place the number in D
loop        lsrd                ; shift the LSB of D into the C flag
            bcs     chkend      ; branch if C flag a 1
            inc     zero_cnt    ;    otherwise inc. 0's count
chkend      dec     lp_cnt      ;
            bne     loop        ;  Done?
forever     bra     forever
            end
```

Boolean Logic Instructions

(Useful for I/O Operations)

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	$B \leftarrow \$00 - (B)$

Clocks and Time Delays

- The HCS12 uses the E clock as a timing reference.
- E clock frequency is half of that of the crystal oscillator.

- Many applications require the use of time delays.
- Two steps to create a time delay:

  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 based on the clock frequency.

Clocks and Time Delays Cont'd.

The routine below takes 4 E cycles to execute. By repeating this routine a certain number of times, any time delay can be created. The *ldy* instruction also take time but is relatively insignificant.

```
        ldy     #N
dly     dey             ; 1 cycle to execute the decrement
        bne     dly     ; 3 cycles to execute the conditional branch
```

Example A

If the HCS12 has a crystal oscillator with a frequency of 20 MHz, then f(E) = 20/2 = 10 MHz and T = 1/f = .1 μs = 100 ns. Therefore the delay created will be:

(100 ns/E cycle)(4 E cycles) = 400 ns or .4 μs

If N equated to 1000 the total delay is: 1000●.4 μs = .4 ms

Clocks and Time Delays Cont'd.

Example B. Using the same frequency as the previous slide, what is the maximum delay we can get out of this loop?

Solution: The largest number we can place in the 16 bit Y register is \$FFFF or 65,535. Rounding that off to 65,000 results in a maximum delay of $(65000)(.4\ \mu s) = 26000\ \mu s$ or 26 ms.

Example C. Based on the above, how can we get a delay of 1 s?

Solution: We will need an outer loop to multiply the basic delay. How many times will the outer loop need to execute?

1 second/26 ms = 38.46 this won't work too well. Let's come up with a better inner loop delay. How about 25 ms? Then 1s/25 ms= 40 for the outer loop. Thus, inner loop: 25 ms/.4 μs = 62,500. Here's our delay snippet:

```
          ldx       #40
outer     ldy       #62500     ; outer loop executes 40 times
inner        dey               ; inner loop executes 40 ● 62,500 times
             bne    inner
          dbne      x, outer
```

NOTE: there is some ***overhead*** in that the ldy and dbne instructions will be executed 40 times- if this is an issue that delay can be calculated and compensated for.

Clocks and Time Delays Cont’d.

- In class Exercise:

The ***Dragon12*** board runs under a crystal oscillator with a frequency of 48 MHz. Recalculate the example in the last slide and make changes in the numbers to end up with the same 1 second delay. Suggestion: make the inner loop 1 ms.