ECET 310-001
Chapter 2, Part 3 of 3

W. Barnes, 9/2006, rev’d. 10/07
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

\[
\text{[<label>] brclr (opr),(msk),(rel) \ [<comment>]} \\
\text{[<label>] brset (opr),(msk),(rel) \ [<comment>]} \\
\]

where

- \textbf{opr} specifies the memory location to be checked and must be specified using either the direct, extended, or index addressing mode.

- \textbf{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.

- \textbf{rel} the branch offset specified in the 8-bit relative mode, usually with a label

For example:

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

**BOTTOM LINE:**

for \texttt{brclr}, put 1’s in bits where you are looking for 0’s and for \texttt{brset}, put 1’s where you are looking for 1’s
**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 \textit{repeat S until C} looping construct.

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

```assembly
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

array db 2,3,4,8,12,13,19,24,33,32
end
```

Discuss: is this really \textit{repeat S until C} looping?
Shift and Rotate Instructions

• Three 8-bit arithmetic shift left instructions:

- \texttt{asl opr} -- memory location \texttt{opr} is shifted left one place
- \texttt{asla} -- accumulator A is shifted left one place
- \texttt{aslb} -- accumulator B is shifted left one place

\[ C \leftarrow \begin{array}{c} \text{b7} \rightarrow \ldots \rightarrow \text{b0} \end{array} \rightarrow 0 \]

• One 16-bit arithmetic shift left instruction:

- \texttt{asld}

\[ C \leftarrow \begin{array}{c} \text{b7} \rightarrow \ldots \rightarrow \text{b0} \end{array} \leftarrow 0 \]

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

- \texttt{arl opr} -- memory location \texttt{opr} is shifted right one place
- \texttt{asra} -- accumulator A is shifted right one place
- \texttt{asrb} -- accumulator B is shifted right one place

\[ \begin{array}{c} \text{b7} \rightarrow \ldots \rightarrow \text{b0} \end{array} \leftarrow C \]
Shift and Rotate Instructions Cont’d.

• Logical shift instructions
  – Shift left instructions (lsl opr, Isla, lslb, lsld) perform identical operation as arithmetic shifts left
  – Shift right instructions (lsr opr, Isra, 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

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Instruction</th>
<th>Initial values</th>
<th>Final Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2.18)</td>
<td>asla</td>
<td>[A] = $95, C = 1</td>
<td></td>
</tr>
<tr>
<td>(2.19)</td>
<td>asr $800</td>
<td>m[$800] = $ED, C = 0</td>
<td></td>
</tr>
<tr>
<td>(2.20)</td>
<td>lsr $1000</td>
<td>m[$1000] = $E7, C = 1</td>
<td></td>
</tr>
<tr>
<td>(2.21)</td>
<td>rolb</td>
<td>[B] = $BD, C = 1</td>
<td></td>
</tr>
<tr>
<td>(2.22)</td>
<td>rora</td>
<td>[A] = $BE and C = 1</td>
<td></td>
</tr>
</tbody>
</table>
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 Boolean logic instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDA &lt;opr&gt;</td>
<td>AND A with memory</td>
<td>$A \leftarrow (A) \cdot (M)$</td>
</tr>
<tr>
<td>ANDB &lt;opr&gt;</td>
<td>AND B with memory</td>
<td>$B \leftarrow (B) \cdot (M)$</td>
</tr>
<tr>
<td>ANDCC &lt;opr&gt;</td>
<td>AND CCR with memory (clear CCR bits)</td>
<td>$CCR \leftarrow (CCR) \cdot (M)$</td>
</tr>
<tr>
<td>EORA &lt;opr&gt;</td>
<td>Exclusive OR A with memory</td>
<td>$A \leftarrow (A) \oplus (M)$</td>
</tr>
<tr>
<td>EORB &lt;opr&gt;</td>
<td>Exclusive OR B with memory</td>
<td>$B \leftarrow (B) \oplus (M)$</td>
</tr>
<tr>
<td>ORAA &lt;opr&gt;</td>
<td>OR A with memory</td>
<td>$A \leftarrow (A) + (M)$</td>
</tr>
<tr>
<td>ORAB &lt;opr&gt;</td>
<td>OR B with memory</td>
<td>$B \leftarrow (B) + (M)$</td>
</tr>
<tr>
<td>ORCC &lt;opr&gt;</td>
<td>OR CCR with memory</td>
<td>$CCR \leftarrow (CCR) + (M)$</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear C bit in CCR</td>
<td>$C \leftarrow 0$</td>
</tr>
<tr>
<td>CLI</td>
<td>Clear I bit in CCR</td>
<td>$I \leftarrow 0$</td>
</tr>
<tr>
<td>CLV</td>
<td>Clear V bit in CCR</td>
<td>$V \leftarrow 0$</td>
</tr>
<tr>
<td>COM &lt;opr&gt;</td>
<td>One's complement memory</td>
<td>$M \leftarrow $\text{FF} - (M)$</td>
</tr>
<tr>
<td>COMA</td>
<td>One's complement A</td>
<td>$A \leftarrow $\text{FF} - (A)$</td>
</tr>
<tr>
<td>COMB</td>
<td>One's complement B</td>
<td>$B \leftarrow $\text{FF} - (B)$</td>
</tr>
<tr>
<td>NEG &lt;opr&gt;</td>
<td>Two's complement memory</td>
<td>$M \leftarrow $\text{00} - (M)$</td>
</tr>
<tr>
<td>NEGA</td>
<td>Two's complement A</td>
<td>$A \leftarrow $\text{00} - (A)$</td>
</tr>
<tr>
<td>NEGB</td>
<td>Two's complement B</td>
<td>$B \leftarrow $\text{00} - (B)$</td>
</tr>
</tbody>
</table>
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 \textit{ldy} instruction also take time but is relatively insignificant.

\begin{verbatim}
    ldy  #N
    dly  dey     ; 1 cycle to execute the decrement
    bne  dly     ; 3 cycles to execute the conditional branch
\end{verbatim}

Example A

If the HCS12 has a crystal oscillator with a frequency of 20 MHz, then
\[ f(E) = \frac{20}{2} = 10 \text{ MHz} \] and \[ T = \frac{1}{f} = 0.1 \mu s = 100 \text{ ns}. \] Therefore the delay created will be:

\[ (100 \text{ ns/E cycle})(4 \text{ E cycles}) = 400 \text{ ns} \text{ or } .4 \mu s \]

If N equated to 1000 the total delay is: 1000\( \cdot .4 \mu s = .4 \text{ ms} \)
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 \mu s = 62,500. Here’s our delay snippet:

```
ldx #40
ldy #62500 ; outer loop executes 40 times
dey ; inner loop executes 40 \cdot 62,500 times
bne inner
bne 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.