Bit Manipulation
Bit Manipulation

• We often need to test or set specific bits

• Example:
  – A signal is input to Port T, bit 0 (e.g., a switch)
  – A signal is output from Port T, bit 3 (e.g., an LED)
  – We want to test bit 0 (*and ignore other bits*)
  – We want to set bit 3 (*without changing any other bits*)

• We will use “logical” instructions, and “shift” operations
• First look at how to do in assembly, then C
Bitwise Logical Instructions in Assembly

• And
  – anda, andb, andcc
  – Examples:
    anda  N  ; A <- (A)*(N)
    andb  #1  ; B <- (B)*$01

• Or
  – oraa, orab, orcc

• Exclusive Or
  – eora, eorb

• Ones complement (just flip all bits)
  – coma, comb, com
Bitwise Logical Operators in C

- Bitwise logical operators
  - &  AND
  - |  OR
  - ^  XOR
  - ~  NOT

- AND

```c
char a, b, c;
a = 0xf0;
b = 0x1f;
c = a & b;
```
Bitwise Logical Operators (continued)

• OR

```c
char a, b, c;
a = 0x30;
b = 0x18;
c = a | b;
```

• Exclusive OR

```c
char a, b, c;
a = 0x30;
b = 0x18;
c = a ^ b;
```
Examples

• & is often used to clear one or more bits of a byte to 0
  – Example: Clear bit 1 of PTT (assume PTT is type char)
    
    ```
    PTT = PTT & 0xf0;
    ```
    
    you can also do
    
    ```
    PTT = PTT & ~0x02;
    ```

  0xfd is 11111101 in binary

  and’ing a 1 with any bit doesn’t change it;
  and’ing a 0 with any bit sets that bit to 0

• | is often used to set one or more bits to 1
  – Example: Set bits 6 and 1 of PTT to a 1
    
    ```
    PTT = PTT | 0x02;
    ```
    
    or’ing a 0 with any bit doesn’t change it;
    or’ing a 1 with any bit sets that bit to 1

• ^ can be used to toggle a bit
  – Example: Toggle upper four bits of PTT
    
    ```
    PTT = PTT ^ 0xf0;
    ```
Examples of Logical Operations

• Force bits 3,4 of M to be 1’s

• Force bits 3,4 of M to be 0’s

• Test if both bits 5,6 of M are 1’s
Examples of Logical Operations

• Force bits 3,4 of M to be 1’s
  \[ M = M \mid 0x18; \]

  or’ing a 0 with any bit doesn’t change it;
  or’ing a 1 with any bit sets that bit to 1

• Force bits 3,4 of M to be 0’s
  \[ M = M \& 0xe7; \]

  and’ing a 1 with any bit doesn’t change it;
  and’ing a 0 with any bit sets that bit to 0

  you can also do
  \[ M = M \& \sim 0x18; \]

  \(0xe7\) is the (ones) complement of \(0x18\)

• Test if both bits 5,6 of M are 1’s
  \[
  \text{if } ((M \& 0x60) == 0x60) \\
  \quad \text{// both are ones}
  \]
  \[
  \text{else} \\
  \quad \text{// both are not ones}
  \]

  and’ing M with 01100000 masks off all bits
  (ie, sets them to 0) except bits 5 and 6

  then test if that result equals 01100000; if
  so then we know both bits 5 and 6 were set
Shift Instructions

Left shift
• Shifting left one place is effectively multiplying by 2

• Assembly
  – Can apply to a register (i.e., A,B,D) or a memory location
  – Example:
    
    \[
    \begin{align*}
    & \text{ldaa} & \#83 \\
    & \text{lsla}
    \end{align*}
    \]

    \textit{result: A contains $06, carry bit = 1}

• Instructions
  – lsla, lslb, lsld, lsl
Shift Instructions (continued)

Right shift (logical)

• Shifting left one place is effectively dividing by 2

• Assembly
  – Can apply to a register (i.e., A,B,D) or a memory location
  – Example:
    ldaa #$83
    lsra

  result: A contains $41, carry bit = 1

• Instructions
  – lsra, lsrb, lsrd, lsr
Shift Instructions (continued)

Right shift (arithmetic)
- If we have a twos complement number, then we have to replicate the sign bit to correctly divide by 2

- Assembly
  - Example:
    
    \[
    \begin{align*}
    \text{ldaa} & \quad \#83 \\
    \text{asra} & \\
    \end{align*}
    \]

    \textit{result: A contains } \textit{c1, carry bit} = 1

- Instructions
  - asra, asrb, asr
Example 2.18  Suppose that [A] = $95 and C = 1. Compute the new values of A and C after the execution of the instruction \texttt{asla}.

Solution:

Example 2.20  Suppose that \texttt{m[$800]} = $E7 and C = 1. Compute the new contents of \texttt{m[$800]} and the C flag after the execution of the instruction \texttt{lsr } \texttt{$800}.

Solution:
Example 2.18 Suppose that \( [A] = \$95 \) and \( C = 1 \). Compute the new values of \( A \) and \( C \) after the execution of the instruction \texttt{asl.a}.

Solution:

![Figure 2.11a Operation of the ASLA instruction]

\[
\begin{array}{c}
\text{Original value} \\
A = 10010101 \\
C = 1 \\
\text{New value} \\
A = 00101010 \\
C = 1
\end{array}
\]

Figure 2.11b Execution result of the ASLA 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 \texttt{lsl \ $800}. \)

Solution:

![Figure 2.13a Operation of the LSR $800 instruction]

\[
\begin{array}{c}
\text{Original value} \\
[800] = 11100111 \\
C = 1 \\
\text{New value} \\
[800] = 01110011 \\
C = 1
\end{array}
\]

Figure 2.13b Execution result of LSR $800
Example 2.19 Suppose that $m[\$1000] = \$ED$ and $C = 0$. Compute the new values of $m[\$1000]$ and the C flag after the execution of the instruction \texttt{asr \$1000}.
Example 2.19 Suppose that \( m[\$1000] = $\text{ED} \) and \( C = 0 \). Compute the new values of \( m[\$1000] \) and the \( C \) flag after the execution of the instruction \textit{asr} \ $\text{1000}$.

Solution:

![Figure 2.12a Operation of the ASR $\$1000$ instruction](image)

<table>
<thead>
<tr>
<th>Original value</th>
<th>New value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( $1000 ) = 11101101</td>
<td>( $1000 ) = 11110110</td>
</tr>
<tr>
<td>( C = 0 )</td>
<td>( C = 1 )</td>
</tr>
</tbody>
</table>

Figure 2.12b Result of the \textit{asr} $\$1000$ instruction

Note:

- If the original number \( \%11101101 \) was a twos complement number, it had the value \(-19\) decimal
  \[-1(128) + 1(64) + 1(32) + 0(16) + 1(8) + 1(4) + 0(2) + 1(1) = -19\]

- The new value of \( \%11110110 \), treated as a twos complement number, has the value \(-10\) decimal
  \[-1(128) + 1(64) + 1(32) + 1(16) + 0(8) + 1(4) + 1(2) + 0(1) = -10\]

- So this implemented a division by 2 (rounding down)
Example

- Output the bits stored in byte “mydata” to PT0.

- Approach:
  - Assume we want to output the most significant bit (MSB) first.
  - We will left shift 8 times.
  - Each time, test the bit that was shifted out; if a one, output a one to PT0

```
ldab   #8   ; counter
loop   ldaa  PTT   ; get current value of PTT
        lsl  mydata  ; left shift one place
        bcc   out0   ; branch if bit shifted out was zero
        oraa  #$01  ; .. otherwise, force PT0 to 1
        bra   endbit
out0   anda  #$01  ; force PT0 to 0
endbit staa  PTT
dbne   b,loop
```
Shift Operators in C

- **Operators**
  
  \[
  \begin{align*}
  &\text{right shift} & \gg \\
  &\text{left shift} & \ll
  \end{align*}
  \]

- **In C, you can shift multiple places, not just one bit**

  ```c
  // Shift lower 4 bits of PTT to upper 4 places.
  PTT = PTT << 4;  // could also write as PTT <<= 4;
  ```

- **For right shifts, if the operand is an unsigned type, it does a logical shift; otherwise it does an arithmetic shift**

  ```c
  unsigned char a = 0x80;
  char b = 0x80;

  a >>= 2;  // shift right 2 places
  b >>= 3;  // shift right 3 places
  ```

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>
Examples

• Predict the results of the following C instructions:

```c
char a, b;
unsigned char c;

a = 0x95;
a = a << 1;

b = 0xe7;
b = b >> 2;

c = 0xed;
c = c >> 3;
```
Summary / Questions

• Shift instructions just shift all the bits either left or right. What are some possible uses of this?

• Logical operators do a bit-wise logical operation (such as AND, OR, XOR) on a pair of bytes. What are some possible uses of this?