Arrays

- In Lab Task 1, we used array that stores only 1 or 0 on each element.
- Using an integer array is efficient?
  - sizeof(int) returns 4 or 8.
Bitwise operators

- With a 4 bytes memory space, we can store $8 \times 4 = 32$ Boolean data.
- Therefore, we can save at least $1/32$ memory space using bitwise operators.

Revisiting printf() function

- Revisiting printf() function
  - `%d` for Decimal (Base 10)
  - `%x`, `%X` for hexadecimal (Base 16)
  - `%o` for octal (Base 8)
  - Do we have a binary (Base 2) format specifier? No
N-bit integer system

Most significant bit (left-most bit) denotes the sign bit
- 0 indicates positive
- 1 indicates negative

2's Complement for signed representation

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal (unsigned)</th>
<th>Decimal (signed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>-4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>-3</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>-2</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>-1</td>
</tr>
</tbody>
</table>

Example:

2'sComplement(x) = −x

2'sComplement(5) = −5
2'sComplement(−5) = 5
### 2’s Complement for signed representation

- **N-bit integer system**
- 2’s complement is an operation of changing the sign of the given integer.

#### 2's Complement for signed representation

<table>
<thead>
<tr>
<th>Binary</th>
<th>(unsigned)</th>
<th>(signed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>-4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>-3</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>-2</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>-1</td>
</tr>
</tbody>
</table>

2'\(s\) Complement (x) = 1'\(s\) Complement (x) + 1

Then what is 1’s Complement of x?

### 2’s Complement for signed representation

- **N-bit integer system**
- **1’s Complement of x:**
  - The number produced after you subtract each bits of x from 1
  - 1'sComplement (011\(_2\)) = 1'sComplement (3) = 100\(_2\), where N=3
  - 1'Complement (1100\(_2\)) = 0011\(_2\), where N=4

2'\(s\) Complement (x) = 1'\(s\) Complement (x) + 1
2’s Complement for signed representation

- **N-bit integer system**
- 2’s Complement of \( x \):
  - \( 2'\text{sc}omplement(011_2) = 2'\text{c}omplement(3) = 1'\text{sc}omplement(011_2) + 1 = 100_2 + 1 = 101_2 \)

\[ 2'\text{c}omplement(x) = 1'\text{c}omplement(x) + 1 \]

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal (unsigned)</th>
<th>Decimal (signed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>-4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>-3</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>-2</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>-1</td>
</tr>
</tbody>
</table>

So, this negative number is the minimum in the range of number using \( N \)-bits

\[ 2'\text{c}omplement(x) = 1'\text{c}omplement(x) + 1 \]
N-bit integer system

The range of representable integers:

When N=3:

-4 to 3

When N=4:

-8 to 7

When N:

\((-2^{N-1}) \text{ to } (2^{N-1} - 1)\)

2’s Complement for signed representation

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal (unsigned)</th>
<th>Decimal (signed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>-4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>-3</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>-2</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>-1</td>
</tr>
</tbody>
</table>

2’s Complement for signed representation

Now you can replace any subtraction, division, multiplication operation using only the addition operation !! (A Giant Step towards circuit minimization)

\[ 5 - 2 = ? \]
\[ = 5 + (-2) \]

(Let, N=4 bits)

\[ 2\text{'s Complement}(2) = -2 = 1\text{'s Complement}(2)+1 \]
\[ = 1\text{'s Complement}(0010_2)+1 \]
\[ = 1101_2 + 1 = 1110_2 \]
2’s Complement for signed representation

- N-bit integer system
- Now you can replace any subtraction, division, multiplication operation using only the addition operation !! (A Giant Step towards circuit minimization)

\[ 5 - 2 = ? \]
\[ = 5 + (-2) \]
\[ = 0101_2 + 1110_2 = ? \ (N=4\text{bits}) \]

\[
\begin{array}{c}
0101_2 \\
1110_2 \\
\hline
10011_2 = 3
\end{array}
\]

What is Bitwise Operator?

- Allows you to get, set, shift bits in an integer type variable declared in C

<table>
<thead>
<tr>
<th>Tools (operators) to be used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitwise NOT (~): Is the 1’s complement operator</td>
</tr>
<tr>
<td>Bitwise AND (&amp;):</td>
</tr>
<tr>
<td>0 &amp; 0 = 0</td>
</tr>
<tr>
<td>0 &amp; 1 = 0</td>
</tr>
<tr>
<td>1 &amp; 0 = 0</td>
</tr>
<tr>
<td>1 &amp; 1 = 1</td>
</tr>
</tbody>
</table>
What is Bitwise Operator?

- **Tools (operators) to be used:**
  - Bitwise NOT (~) : Is the 1’s complement operator
  - Bitwise AND (&)
  - Bitwise OR ( | ):
    - 0 | 0 = 0
    - 0 | 1 = 1
    - 1 | 0 = 1
    - 1 | 1 = 1

What is Bitwise Operator?

- **Tools (operators) to be used:**
  - Bitwise NOT (~) : Is the 1’s complement operator
  - Bitwise AND (&)
  - Bitwise OR ( | )
  - Bitwise XOR (^):
    - 0 ^ 0 = 0
    - 0 ^ 1 = 1
    - 1 ^ 0 = 1
    - 1 ^ 1 = 0
What is Bitwise Operator?

Tools (operators) to be used:

- Bitwise NOT (~) : Is the 1’s complement operator
- Bitwise AND (&)
- Bitwise OR ( | )
- Bitwise XOR (^)
- Left Shift Operator (<<):
  - \( x = 01001101_2 \)
  - \( x << 1 \)
  - Value of \( x = 10011010_2 \)
  - \( x << 1 \)
  - Value of \( x = 00110100_2 \)
- Right Shift Operator (>>):
  - \( x = 01001101_2 \)
  - \( x >> 1 \)
  - Value of \( x = 00110100_2 \)
  - \( x >> 1 \)
  - Value of \( x = 00010011_2 \)
  - \( y = 10110100_2 \)
  - \( y >> 3 \)
  - Value of \( y = 10100000_2 \)

(sign extension if \( y \) is a signed integer, although it is architecture dependent)
What is Bitwise Operator?

- **Tools (operators) to be used:**
  - [refer to bitwise_operators_example1.c]
  - Bitwise NOT (~)
  - Bitwise AND (&)
  - Bitwise OR ( | )
  - Bitwise XOR (^)
  - Left Shift Operator (<<)
  - Right Shift Operator (>>)

Why bother learning it?

- **Packing of (Boolean) Variables**
  - An N-bit integer variable can pack N Boolean variables in it
  - How can you access 3\textsuperscript{rd} from the right variable from the packed variable?
  - How can you change of value to the 5\textsuperscript{th} from the right variable in the packed variable?
Accessing 3\textsuperscript{rd} bit from the right

Suppose, N = 8 bit integer system (unsigned)

\[
x = \begin{array}{cccccccc}
    x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\end{array}
\]

\[
1 \& x_2 = x_2 \\
0 \& x_2 = 0
\]

\[
\begin{array}{cccccccc}
    & & & & & & & \\
\end{array}
\]

Result = \begin{cases} 
1 & \text{if } x_2 = 1 \\
0 & \text{if } x_2 = 0
\end{cases}

\text{if } ( \text{x} \& (1\ll2) ) \neq 0 \text{ printf("1"); else printf("0");}
Set 5th bit from the Right to 1

Suppose, N = 8 bit integer system (unsigned)

\[ x = \begin{array}{cccccccc}
    x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\end{array} \]

Set 5th bit from the Right to 1

Suppose, N = 8 bit integer system (unsigned)

\[ x = \begin{array}{cccccccc}
    x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\end{array} \]

\[ \text{mask} = \begin{array}{cccccccc}
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array} \ll 4 \]

\[ \text{mask} = \begin{array}{cccccccc}
    0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array} \]

\[ \text{Result} = \begin{array}{cccccccc}
    x_7 & x_6 & x_5 & 1 & x_3 & x_2 & x_1 & x_0 \\
\end{array} \]

\[ x = x \text{ | } (1 \ll 4) ; \]
Set 5\textsuperscript{th} bit from the Right to 0

Suppose, \( N = 8 \) bit integer system (unsigned)

\[ x = \begin{array}{cccccccc} x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \end{array} \]

\[ \text{mask} = \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \ll 4 \]

\[ x = x \& \left( \lnot (1 \ll 4) \right); \]

---

Toggle (Complement) 5\textsuperscript{th} bit from the right

Suppose, \( N = 8 \) bit integer system (unsigned)

\[ x = \begin{array}{cccccccc} x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \end{array} \]

\[ 1 \uparrow x_4 = \overline{x_4} \]

\[ 0 \uparrow x_4 = x_4 \]
Toggle (Complement) 5th bit from the right

Suppose, N = 8 bit integer system (unsigned)

\[ x = \begin{array}{cccccccc}
  x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\end{array} \]

\[ \text{mask} = \begin{array}{cccccccc}
  0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array} \]

\[ \text{Result} = \begin{array}{cccccccc}
  x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\end{array} \]

\[ x = x \oplus (1 \ll 4); \]

Alternatively,

\[ x = x \oplus 0x10; \]

What is the integer equivalent value represented by the bits \( x_5 x_4 x_3 \)?

Suppose, N = 8 bit integer system (unsigned)

\[ x = \begin{array}{cccccccc}
  x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\end{array} \]

\[ \text{mask} = \begin{array}{cccccccc}
  0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array} \]

\[ \text{We get} = \begin{array}{cccccccc}
  0 & 0 & x_5 & x_4 & x_3 & 0 & 0 & 0 \\
\end{array} \]

\( \gg 3 \)

We right shift this to align the three bits to the right, so that \( x_3 \) becomes the least-significant bit.

\[ \text{printf("%d", (x & 0x38) >> 3 }); \]
Other Advantages?

- Bitwise operators provide efficient ways to perform some computations.

Multiplying $x$ by $2^k$

- $x \ll k$
Divide \( x \) by \( 2^k \)

- \( x \gg k \)

Check if \( x \) is an integer power of 2

```c
if( (x & (x-1)) == 0 )
    printf("YES!!");
else
    printf("NO");
```

<table>
<thead>
<tr>
<th>( x )</th>
<th>Binary</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 4 )</td>
<td>00100</td>
<td>( 00000 == 0 ) (YES)</td>
</tr>
<tr>
<td>( x = 8 )</td>
<td>01000</td>
<td>( 00000 == 0 ) (YES)</td>
</tr>
<tr>
<td>( x = 12 )</td>
<td>01100</td>
<td>( 01000 != 0 ) (No)</td>
</tr>
</tbody>
</table>
Even or ODD?

```c
if( x & 1 == 1 )
    printf(“ODD!!”);
else
    printf(“EVEN”);
```

Multi-options as a single parameter

- In Windows Programming,
  - Multiple options can be passed as a single parameter by using bitwise OR operator

```c
HWND WINAPI CreateWindowEx(
    _In_    DWORD dwExStyle,
    _In_opt_ LPCTSTR lpClassName,
    _In_opt_ LPCTSTR lpWindowName,
    _In_    DWORD dwStyle,
    _In_    int x,
    _In_    int y,
    _In_    int nWidth,
    _In_    int nHeight,
    _In_opt_ HWND hWndParent,
    _In_opt_ HMENU hMenu,
    _In_opt_ HINSTANCE hInstance,
    _In_opt_ LPVOID lpParam
);
```
Multi-options as a single parameter

- `WS_EX_APPWINDOW | WS_EX_CLIENTEDGE | WS_EX_RIGHTSCROLLBAR | ..`


References

- Bitwise Operators in C
Exercise

- Let's implement Lab 1 again by using bitwise operators.