Representing Data with Bits

bits, bytes, numbers, and notation

conversion and arithmetic

19\textsubscript{10} = ?\textsubscript{2}

1001\textsubscript{2} = ?\textsubscript{10}

240\textsubscript{10} = ?\textsubscript{2}

11010011\textsubscript{2} = ?\textsubscript{10}

101\textsubscript{2} + 1011\textsubscript{2} = ?\textsubscript{2}

1001011\textsubscript{2} \times 2\textsubscript{10} = ?\textsubscript{2}

\textit{bitwise operators}

\textbf{Bitwise operators} on fixed-width \textbf{bit vectors}.

\begin{align*}
\text{AND} & \quad \text{OR} & \quad \text{XOR} & \quad \text{NOT} \\
01101001 & | 01010101 & 01101001 & ^01010101 & \sim01010101
\end{align*}

Laws of Boolean algebra apply bitwise.

\textit{e.g.}, DeMorgan’s Law: \(\sim(A \lor B) = \sim A \land \sim B\)

\textit{bitwise operators in C}

\& \quad | \quad ^ \quad \sim \quad \text{apply to any} \ integral \text{ data type}

\begin{align*}
\text{long, int, short, char, unsigned}
\end{align*}

\textbf{Examples} (\texttt{char})

\begin{align*}
\sim0x41 & = \hfill \\
\sim0x00 & = \\
0x69 \& 0x55 & = \\
0x69 \mid 0x55 & =
\end{align*}

Many bit-twiddling puzzles in upcoming assignment
**logical operations in C**

`&&` `||` `!` apply to any "integral" data type `long`, `int`, `short`, `char`, `unsigned`

0 is false  nonzero is true  result always 0 or 1

early termination  a.k.a.  short-circuit evaluation

Examples (char)

```c
!0x41 =
!0x00 =
!!0x41 =
0x69 && 0x55 =
0x69 || 0x55 =
```

**Compare Card Suits**

`mask`: a bit vector that, when bitwise ANDed with another bit vector `v`, turns all but the bits of interest in `v` to 0

```c
#define SUIT_MASK 0x30
```

```c
int sameSuit(char card1, char card2) {
    return !(((card1 & SUIT_MASK) ^ (card2 & SUIT_MASK));
    //same as (card1 & SUIT_MASK) == (card2 & SUIT_MASK);
}
```

```c
char hand[5];  // represents a 5-card hand
char card1, card2;  // two cards to compare
...  
if ( sameSuit(hand[0], hand[1]) ) { ... }
```

**Compare Card Values**

`mask`: a bit vector that, when bitwise ANDed with another bit vector `v`, turns all but the bits of interest in `v` to 0

```c
#define VALUE_MASK
```

```c
int greaterValue(char card1, char card2) {

}
```

```c
char hand[5];  // represents a 5-card hand
char card1, card2;  // two cards to compare
...  
if ( greaterValue(hand[0], hand[1]) ) { ... }
```

**Shift and Mask: extract a bit field**

Write C code:

extract 2\textsuperscript{nd} most significant byte from a 32-bit integer.

```c
given \quad x = \quad \text{expected} = \quad \text{desired bits in least significant byte.}
```

All other bits are zero.