CS 33

Data Representation, Part 2
Numeric Ranges

- **Unsigned Values**
  - $U_{Min} = 0$
    - 000...0
  - $U_{Max} = 2^w - 1$
    - 111...1

- **Two’s Complement Values**
  - $T_{Min} = -2^{w-1}$
    - 100...0
  - $T_{Max} = 2^{w-1} - 1$
    - 011...1

- **Other Values**
  - Minus 1
    - 111...1

Values for $W = 16$

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMax</td>
<td>65535</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>Tmax</td>
<td>32767</td>
<td>7F FF</td>
<td>01111111 11111111</td>
</tr>
<tr>
<td>Tmin</td>
<td>-32768</td>
<td>80 00</td>
<td>10000000 00000000</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00 00</td>
<td>00000000 00000000</td>
</tr>
</tbody>
</table>
### Values for Different Word Sizes

<table>
<thead>
<tr>
<th></th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>UMax</td>
<td>255</td>
</tr>
<tr>
<td>Tmax</td>
<td>127</td>
</tr>
<tr>
<td>Tmin</td>
<td>-128</td>
</tr>
</tbody>
</table>

- **Observations**
  
  \[
  \begin{align*}
  |TMin| &= TMax + 1 \\
  \text{Asymmetric range} \\
  UMax &= 2 \times TMax + 1
  \end{align*}
  
- **C Programming**
  
  - `#include <limits.h>`
  - declares constants, e.g.,
    - `ULONG_MAX`
    - `LONG_MAX`
    - `LONG_MIN`
  - values platform-specific
Quiz 1

• What is $-\text{TMin}$ (assuming two’s complement signed integers)?
  a) $\text{TMin}$
  b) $\text{TMax}$
  c) 0
  d) 1
4-Bit Computer Arithmetic
Signed vs. Unsigned in C

• Constants
  – by default are considered to be signed integers
  – unsigned if have “U” as suffix
    0U, 4294967259U

• Casting
  – explicit casting between signed & unsigned
    ```c
    int tx, ty;
    unsigned ux, uy; // “unsigned” means “unsigned int”
    tx = (int) ux;
    uy = (unsigned int) ty;
    ```

  – implicit casting also occurs via assignments and procedure calls
    ```c
    tx = ux;
    uy = ty;
    ```
Casting Surprises

- Expression evaluation
  - if there is a mix of unsigned and signed in single expression, *signed values implicitly cast to unsigned*
  - including comparison operations `<`, `>`, `==`, `<=`, `>=`
  - examples for $W = 32$: $T_{MIN} = -2,147,483,648$, $T_{MAX} = 2,147,483,647$

<table>
<thead>
<tr>
<th>Constant₁</th>
<th>Constant₂</th>
<th>Relation</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0U</td>
<td>==</td>
<td>unsigned</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td><code>&lt;</code></td>
<td>signed</td>
</tr>
<tr>
<td>-1</td>
<td>0U</td>
<td><code>&gt;</code></td>
<td>signed</td>
</tr>
<tr>
<td>2147483647</td>
<td>-2147483647-1</td>
<td><code>&gt;</code></td>
<td>signed</td>
</tr>
<tr>
<td>2147483647U</td>
<td>-2147483647-1</td>
<td><code>&lt;</code></td>
<td>unsigned</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td><code>&gt;</code></td>
<td>signed</td>
</tr>
<tr>
<td>(unsigned)-1</td>
<td>-2</td>
<td><code>&gt;</code></td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>2147483648U</td>
<td><code>&lt;</code></td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>(int)2147483648U</td>
<td><code>&gt;</code></td>
<td>signed</td>
</tr>
</tbody>
</table>
Quiz 2

What is the value of

\[(\text{long}) \text{ULONG\_MAX} \ - \ (\text{unsigned long})-1\]

???

a) -1
b) 0
c) 1
d) ULONG\_MAX
Sign Extension

• Task:
  – given $w$-bit signed integer $x$
  – convert it to $w+k$-bit integer with same value

• Rule:
  – make $k$ copies of sign bit:
    – $X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$
Sign Extension Example

### Example

```c
short int x = 15213;
int    ix = (int) x;
short int y = -15213;
int    iy = (int) y;
```

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>15213</td>
<td>3B 6D</td>
<td>00111011 01101101</td>
</tr>
<tr>
<td>ix</td>
<td>15213</td>
<td>00 00 3B 6D</td>
<td>00000000 00000000 00111011 01101101</td>
</tr>
<tr>
<td>y</td>
<td>-15213</td>
<td>C4 93</td>
<td>11000100 10010011</td>
</tr>
<tr>
<td>iy</td>
<td>-15213</td>
<td>FF FF C4 93</td>
<td>11111111 11111111 11000100 10010011</td>
</tr>
</tbody>
</table>

- Converting from smaller to larger integer data type
  - C automatically performs sign extension
Does it Work?

\[
\begin{align*}
\text{val}_w &= -2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i \\
\text{val}_{w+1} &= -2^w + 2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i \\
&= -2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i \\
\text{val}_{w+2} &= -2^{w+1} + 2^w + 2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i \\
&= -2^w + 2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i \\
&= -2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i
\end{align*}
\]
Unsigned Multiplication

- Standard multiplication function
  - ignores high order \( w \) bits
- Implements modular arithmetic

\[
\text{UMult}_w(u, v) = u \cdot v \mod 2^w
\]
Signed Multiplication

Operands: $w$ bits

True Product: $2w$ bits

Discard $w$ bits: $w$ bits

- **Standard multiplication function**
  - ignores high order $w$ bits
  - some of which are different from those of unsigned multiplication
  - lower bits are the same
Power-of-2 Multiply with Shift

- **Operation**

  - \( u \ll k \) gives \( u \times 2^k \)
  
  - both signed and unsigned

  \[
  \text{operands: } w \text{ bits} \\
  \text{true product: } w+k \text{ bits} \\
  \text{discard } k \text{ bits: } w \text{ bits}
  \]

  \[
  \begin{array}{c}
  u \\
  \times 2^k \\
  u \times 2^k
  \end{array}
  \]

  \[
  \begin{array}{c}
  k \\
  \end{array}
  \]

- **Examples**

  \[
  u \ll 3 == u \times 8 \\
  u \ll 5 - u \ll 3 == u \times 24
  \]

- most machines shift and add faster than multiply

  » compiler generates this code automatically
Unsigned Power-of-2 Divide with Shift

- Quotient of unsigned by power of 2
  - \( u >> k \) gives \( \lfloor \frac{u}{2^k} \rfloor \)
  - uses logical shift

<table>
<thead>
<tr>
<th>Division</th>
<th>Computed</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>15213</td>
<td>15213</td>
<td>\text{3B 6D}</td>
</tr>
<tr>
<td>( x &gt;&gt; 1)</td>
<td>7606.5</td>
<td>7606</td>
<td>\text{1D B6}</td>
</tr>
<tr>
<td>( x &gt;&gt; 4)</td>
<td>950.8125</td>
<td>950</td>
<td>\text{03 B6}</td>
</tr>
<tr>
<td>( x &gt;&gt; 8)</td>
<td>59.4257813</td>
<td>59</td>
<td>\text{00 3B}</td>
</tr>
</tbody>
</table>
Signed Power-of-2 Divide with Shift

- Quotient of signed by power of 2
  - $x \gg k$ gives $\lfloor \frac{x}{2^k} \rfloor$
  - uses arithmetic shift
  - rounds wrong direction when $x < 0$

<table>
<thead>
<tr>
<th>Division</th>
<th>Computed</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>-15213</td>
<td>C4 93</td>
<td>11000100 10010011</td>
</tr>
<tr>
<td>$y \gg 1$</td>
<td>-7606.5</td>
<td>E2 49</td>
<td>11100010 01001001</td>
</tr>
<tr>
<td>$y \gg 4$</td>
<td>-950.8125</td>
<td>FC 49</td>
<td>11111100 01001001</td>
</tr>
<tr>
<td>$y \gg 8$</td>
<td>-59.4257813</td>
<td>FF C4</td>
<td>11111111 11000100</td>
</tr>
</tbody>
</table>
Correct Power-of-2 Divide

- Quotient of negative number by power of 2
  - want \( \left\lfloor \frac{x}{2^k} \right\rfloor \) (round toward 0)
  - compute as \( \left\lfloor \frac{x+2^k-1}{2^k} \right\rfloor \)
    - in C: \((x + (1<<k) - 1) >> k\)
    - biases dividend toward 0

Case 1: no rounding

<table>
<thead>
<tr>
<th>Dividend:</th>
<th>( u )</th>
<th>+2(^k)-1</th>
<th>( u ) / 2(^k)</th>
<th>( u / 2^k )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1\ldots0</td>
<td>0\ldots001</td>
<td>0\ldots01</td>
<td>1\ldots00</td>
</tr>
<tr>
<td>Divisor:</td>
<td>( u ) / 2(^k)</td>
<td>0\ldots010\ldots00</td>
<td>( u / 2^k )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1\ldots111\ldots1</td>
<td>1\ldots11</td>
<td>1\ldots11</td>
<td></td>
</tr>
</tbody>
</table>

**Biasing has no effect**
Correct Power-of-2 Divide (Cont.)

Case 2: rounding

dividend:
\[
\begin{array}{c}
  x \\
  +2^k - 1
\end{array}
\]

divisor:
\[
\begin{array}{c}
  \frac{x}{2^k} \\
  \lfloor \frac{x}{2^k} \rfloor
\end{array}
\]

Biasing adds 1 to final result
Why Should I Use Unsigned?

- *Don’t* use just because number nonnegative
  - easy to make mistakes
    ```c
    unsigned i;
    for (i = cnt-2; i >= 0; i--)
      a[i] += a[i+1];
    ```
  - can be very subtle
    ```c
    #define DELTA sizeof(int)
    int i;
    for (i = CNT; i-DELTA >= 0; i-= DELTA)
      ...
    ```
- *Do* use when performing modular arithmetic
  - multiprecision arithmetic
- *Do* use when using bits to represent sets
  - logical right shift, no sign extension
Byte-Oriented Memory Organization

• Programs refer to data by address
  – conceptually, envision it as a very large array of bytes
    » in reality, it’s not, but can think of it that way
  – an address is like an index into that array
    » and, a pointer variable stores an address

• Note: system provides private address spaces to each “process”
  – think of a process as a program being executed
  – so, a program can clobber its own data, but not that of others
Machine Words

- Any given computer has a “word size”
  - nominal size of integer-valued data
    » and of addresses
  - until recently, most machines used 32 bits (4 bytes) as word size
    » limits addresses to 4GB ($2^{32}$ bytes)
    » becomes too small for memory-intensive applications
      - leading to emergence of computers with 64-bit word size
  - machines still support multiple data formats
    » fractions or multiples of word size
    » always integral number of bytes
Word-Oriented Memory Organization

- Addresses specify byte locations
  - address of first byte in word
  - addresses of successive words differ by 4 (32-bit) or 8 (64-bit)

<table>
<thead>
<tr>
<th>32-bit Words</th>
<th>64-bit Words</th>
<th>Bytes</th>
<th>Addr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr = 0000</td>
<td>Addr = 0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>Addr = 0004</td>
<td></td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>Addr = 0008</td>
<td></td>
<td>0002</td>
<td></td>
</tr>
<tr>
<td>Addr = 0012</td>
<td></td>
<td>0003</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Addr = 0000</td>
<td>0004</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0005</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0006</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0007</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0008</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0009</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0011</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0012</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0013</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0014</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0015</td>
<td></td>
</tr>
</tbody>
</table>
Byte Ordering

- Four-byte integer
  - 0x76543210
- Stored at location 0x100
  - which byte is at 0x100?
  - which byte is at 0x103?

<table>
<thead>
<tr>
<th></th>
<th>0x100</th>
<th>0x101</th>
<th>0x102</th>
<th>0x103</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0x100</td>
<td>32</td>
<td>0x101</td>
<td>0x102</td>
</tr>
<tr>
<td>32</td>
<td>0x101</td>
<td>54</td>
<td>0x102</td>
<td>0x103</td>
</tr>
<tr>
<td>54</td>
<td>0x102</td>
<td>76</td>
<td>0x103</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>0x103</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Little-endian

Big-endian
Byte Ordering (2)

Big Endian

Little Endian
Quiz 3

```c
int main() {
    long x = 1;
    proc((int *)&x);
    return 0;
}

void proc(int *arg) {
    printf("%d\n", *arg);
}
```

What value is printed on a big-endian 64-bit computer?

a) 0  
b) 1  
c) $2^{32}$  
d) $2^{32}-1$