Signed Integer Representations

Signed Integer Representations

- Signed magnitude representation: setting a bit (often the most significant bit) to
  - 0 for positive numbers
  - 1 for negative numbers
- One's complement:
  - flipping the bits of +*n* to get –*n*.
- Two's complement:
  - adding one to the one's complement.

Examples

Eight-bit ones' complement

Binary value	Ones' complement interpretation	Unsigned interpretation
00000000	+0	0
00000001	1	1
⋮	⋮	⋮
01111101	125	125
01111110	126	126
01111111	127	127
10000000	−127	128
10000001	−126	129
10000010	−125	130
⋮	⋮	⋮
11111101	−2	253
11111110	−1	254
11111111	−0	255

Eight-bit two's complement

Binary value	Two's complement interpretation	Unsigned interpretation
00000000	0	0
00000001	1	1
⋮	⋮	⋮
01111110	126	126
01111111	127	127
10000000	−128	128
10000001	−127	129
10000010	−126	130
⋮	⋮	⋮
11111110	−2	254
11111111	−1	255

3

Why Two's Complement?

```
      binary    decimal
    11111110     −1
+   00000010     +2
──────────────   ──
 1 00000000      0    ← Not the correct answer
          1      +1   ← Add carry
──────────────   ──
    00000001      1   ← Correct answer
```

4

Bitwise Operators

Bitwise Operators (2.9)

- Work on individual bits

  `&` `|` `^` `~`

- Example:

```
a = 1; /* 0000 0001 */
b = 2; /* 0000 0010 */
c = a & b;  /* c = 0000 0000 = 0 */
d = a && b; /* d = 1 */
```

- `&` is a bitwise operator.
- `&&` is a relational operator.

Example

```
main()
{
  int a = 5;    /* 0000 ... 0000 0101 */
  int b = 9;    /* 0000 ... 0000 1001 */

  printf( "%d\n", a & b );  /* 0000 ... 0000 0001 */
  printf( "%d\n", a | b );  /* 0000 ... 0000 1101 */
  printf( "%d\n", a ^ b );  /* 0000 ... 0000 1100 */

  int c = 0XF;
  printf( "%x %x\n", c, ~c );   /* ~c = FFFF FFF0 */
}
```

7

Application: Bit Masking

```
int n;
n = n & 0177;
```

sets to zero all but the low-order 7 bits of **n**.

```
int x;
x = x & ~077;
```

sets the last six bits of **x** to zero.

```
#define SET_ON
  0XFFFF
int x;
x = x | SET_ON;
```

sets to one in **x** the bits that are set to one in **SET_ON**.

8

Bit Shifting

- x<<y means shift x to the left y times.
  - equivalent to multiplication by 2^y
- x>>y means shift x to the right y bits.
  - equivalent to division by 2^y
- Left shifting 3 many times:

0	3
1	6
2	12
3	24
4	48
5	...

9

Application: Bit Counting

```
/* bitcount: count 1 bits in x */
int bitcount(unsigned x) {
  int b;
  for ( b = 0; x != 0; x >>= 1 )
     if ( x & 01 )
          b++;
  return b;
}
```

10

Right Shifting – Unsigned Numbers

If unsigned numbers, then logical shift (filled with 0).

```
unsigned int i = 714;
```

357 178 89 44 22 11 5 2 1 0

11

Right Shifting – Signed Numbers

- If signed numbers, it could be
  - logical shift (filled with 0), or
  - arithmetic shift (keep the sign of signed numbers)
- Undefined in C (implementation dependent)

```
i = -714
```

-357 -178 -89 . . . -3 -2 -1 -1 -1 -1

(assume two's compliment and arithmetic shift)

12