University of Arizona

CSc 340

Foundations of Computer Systems

Christian Collberg

January 16, 2001

Binary Operations

Slide 3–1

Operations on Bit Sequences

- Binary operations are the basis for manipulating numbers represented as bit sequences and for calculating new numbers.
- There are three types of binary operations:
  1. Logical operations.
  2. Shift and rotate operations.
  3. Arithmetic operations.

Slide 3–2

Logical Operations

- Unary operations (this is a *truth table*):

A	not A
0	1
1	0

- Binary operations:

A	B	A and B	A nand B	A or B	A nor B	A xor B	A xnor B
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Slide 3–3

Logical Operations . . .

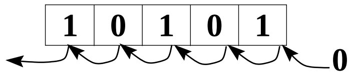
- Logical operations operate on each bit individually.
- Note that xor is "not equal" and xnor is "equal".
- 0111_2 `and` $0010_2 = 0010_2$
- 1010_2 `xor` $0010_2 = 1000_2$

Slide 3–4

Shift/Rotate Operations – Shift Left

- Shift/Rotate Operations operate on a collection of bits.

1. Moves bits to the left (from LSB to MSB)
2. MSB (most-significant bit) is discarded.
3. LSB (least-significant bit) is set to zero.

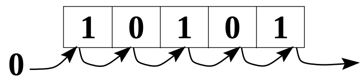


- Example: $00110101_2 \Rightarrow 01101010_2$
- Shifting left by n is the same as multiplying by 2^n.

Slide 3–5

Shift Right

1. Moves bits to the right (from MSB to LSB).
2. LSB is discarded.
3. MSB is set to zero.

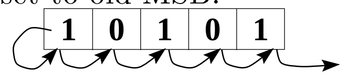


- Example: $00110101_2 \Rightarrow 00011010_2$
- Shifting right by n is the same as dividing by 2^n (for positive numbers).

Slide 3–6

Shift right arithmetic

1. Moves bits to the right (from MSB to LSB).
2. LSB is discarded.
3. New MSB is set to old MSB.

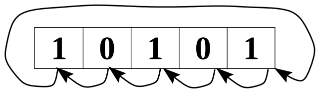


- Example: $00110101_2 \Rightarrow 000110101_2$
- Example: $10110101_2 \Rightarrow 11011010_2$
- Shifting right by n is the same as dividing by 2^n. This is true for both positive and negative 2's complement numbers.

Slide 3–7

Rotate Left

1. Shifts bits to the left (from LSB to MSB).
2. New LSB is set to old MSB.

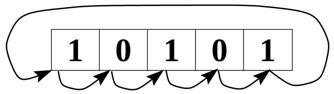


- Example: $10100011_2 \Rightarrow 01000111_2$
- Example: $00100011_2 \Rightarrow 01000110_2$

Slide 3–8

Rotate Right

1. Shifts bits to the right (from MSB to LSB)
2. New MSB is set to old LSB.



- Example: $10100011_2 \Rightarrow 11010001_2$
- Example: $00100011_2 \Rightarrow 10010001_2$

Slide 3–9

Arithmetic Operations – Addition

- A one-bit adder:

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	10

- Note that adding two 1-bit numbers can result in a 2-bit number. If we add 1+1 we have to "carry" a 1 to the addition of the next-most- significant bits.

Slide 3–10

One Bit Addition with Carry

carry in	A	B	carry out	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Slide 3–11

Addition . . .

- Example:

$$\begin{array}{rll} & 0000\ 1110_2 & (14_{10}) \\ + & 0000\ 1010_2 & (10_{10}) \\ \hline = & 0001\ 1000_2 & (24_{10}) \end{array}$$

- Because the result of adding two numbers can have one more bit than the operands, we have to worry about overflow:

Slide 3–12

Addition . . .

		$1100\ 1010_2$	(202_{10})
$+$		$0110\ 0000_2$	(96_{10})
$=$	1	$0010\ 1010_2$	(298_{10})

- The result, 298_{10}, requires 9 bits. If there is only room to store 8 bits we have a problem. Note that the lower-order 8 bits, $0010\ 1010_2$ is not 298_{10}, it's 42_{10}.

Slide 3–13

Negative Numbers – Signed Magnitude

- So far we've only considered positive numbers. Negative numbers are useful too.
- One representation is called signed magnitude. The MSB is called the sign bit – 0 means positive, 1 means negative.
- Example: $7_{10} = 0111_2$
- Example: $-7_{10} = 1111_2$

Slide 3–14

Signed Magnitude . . .

- Signed magnitude has a problem with zero – there are two representations!
  1. $0000_2 = 0_{10}$
  2. $1000_2 = -0_{10} = 0_{10}$
- Having two zeros makes life difficult. Instead of saying

```
if (x == 0)
    y = 5;
```

  we'd have to say

```
if ((x == 0+) || (x == 0-))
    y = 5;
```

Slide 3–15

Negative Numbers – 2's Complement

- An alternative representation that has only one zero is called 2's complement.
- To compute the 2's complement of a number first invert all the bits (compute the "not"), then add 1.

$0000\ 0011_2$	3_{10}
$\Downarrow$	`invert`
$1111\ 1100_2$	
$\Downarrow$	`add one`
$1111\ 1101_2$	-3_{10}

Slide 3–16

2's Complement ...

- This works for converting positive numbers to negative and vice versa.
- Note that the sign-bit is 0 for positive numbers, 1 for negative.
- Why is 2's complement interesting? Try adding $3_{10} \ + \ (-3_{10})$.

$$
\begin{array}{llll}
 & 0000\ 0011_2 & & (3_{10}) \\
+ & 1111\ 1101_2 & & (-3_{10}) \\
\hline
= & 0000\ 0000_2 & & (0_{10})
\end{array}
$$

Slide 3–17

2's Complement ...

- You can add two 2's-complement numbers together and get the correct answer, without worrying about whether they are negative or positive.

$$
\begin{array}{llll}
 & 0000\ 0111_2 & & (7_{10}) \\
+ & 1111\ 1000_2 & & (-8_{10}) \\
\hline
= & 1111\ 1111_2 & & (-1_{10})
\end{array}
$$

Slide 3–18

2's Complement ...

$$
\begin{array}{lllll}
 & & 1110\ 1000_2 & & (-24_{10}) \\
+ & & 0001\ 0000_2 & & (16_{10}) \\
\hline
= & & 1111\ 1000_2 & & (-8_{10})
\end{array}
$$

$$
\begin{array}{lllll}
 & & 1111\ 1111_2 & & (-1_{10}) \\
+ & & 0000\ 1000_2 & & (8_{10}) \\
\hline
= & 1 & 0000\ 0111_2 & & (7_{10})
\end{array}
$$

Slide 3–19

2's Complement – Overflow

- Overflow is a problem. If the sign-bit is the same for both operands, but different for the result, you overflowed. It is impossible to overflow if the sign-bits are different for the operands.

$$
\begin{array}{lllll}
 & 0100\ 1010_2 & & (74_{10}) & \\
+ & 0110\ 0000_2 & & (96_{10}) & \\
\hline
= & 1010\ 1010_2 & & (-86_{10}) & \textit{overflow!}
\end{array}
$$

Slide 3–20

Subtraction

- Change the sign of the number to be subtracted (do the 2's complement conversion above) and add it.

	$0110\ 0000_2$	(96_{10})
$-$	$0100\ 1010_2$	(74_{10})
$=$	???	
	$\Downarrow$	2's complement
	$0110\ 0000_2$	(96_{10})
$+$	$1011\ 0110_2$	(-74_{10})
$=$	$0001\ 0110_2$	(22_{10})

Slide 3–21

Carry and Overflow Revisited

- Recall that an n-bit adder has three inputs (two n-bit values and a single bit for carry in) and two outputs (an n-bit resulting value and a single bit for carry out). An overflow is said to occur when the result exceeds the range of the number system. Overflow is not a problem if the range is infinite (pen and paper), but the registers in computers are of finite length, so this is something we have to be aware of.
- If an unsigned representation is used, the valid range is $0 \ldots 2^{n-1}$. Any value larger than or equal to 2^n would be out of this range. This can be checked by inspecting the value of the carry out bit: overflow has occurred if and only if the carry bit is set.

Slide 3–22

Carry and Overflow Revisited ...

- If 2's-Complement is used instead, the carry bit may or may not indicate an overflow. The valid range is$-(2^{n-1}) \ldots 2^{n-1} - 1$. Adding a positive and a negative number will always result in a value within the valid range; but adding two negative or two positive numbers may result in numbers outside the valid range. To determine if overflow has occurred, inspect the carry into the sign bit position and the carry out of the sign bit position. If they differ, an overflow has occurred.