CIS 351 - Computer Organization & Assembly Language

Nathan Bowman

Images taken from Harris & Harris book

Useful Operations in Binary and Hex

There are a few handy things you will want to know about binary and hex numbers

- shifting bits in binary is multiplication/division
- binary->hex and hex->binary conversions are simple

Bit shifts

Consider 450 in base 10

If we multiply by 10, it becomes 4500

We have simply shifted the digits left and put a 0 in the "hole" left behind

Bit shifts

What if we muliply by 10 again?

Just shifts again: 450 * 10 * 10 = 45000

So: 450 * 10^2 = 45000

Bit shifts

Any time we multiply by a *power of ten*, we shift left by that *power*

Note that this applies to 10^1, 10^2, 10^3, ...

It does *not* apply to 10 * 1, 10 * 2, 10 * 3, ...

Be careful not to get those confused

Bit shifts

Similary, 5600/10 = 560

Dividing by 10 is simply shifting the digits to the right

Once again, we can shift more than once if we work with *powers of 10*

Bit shifts

5600/10/10 = 5600/10^2 = 56

This applies to 10^2, *not* 10 * 2

Bit shifts

The same idea holds in any base: multiplication or division by a power of the base itself is simply a shift left or right

We tend to use this fact a lot in binary

Bit shifts

Quick, what is 1100_2 / 2?

Bit shifts

1100_2 / 2 is a single right shift, so 110_2

You can double-check the math:

- 1100_2 = 12
- 110_2 = 6

Bit shifts

What about 1100_2 / 4?

Bit shifts

Two shifts: 1100_2 / 4 = 11_2

Should should double-check the math again for practice

Bit shifts

What about 1100_2 * 2?

Bit shifts

1100_2 * 2 = 11000_2

Decimal: 12 * 2 = 24

Bit shifts

What about 1100_2 * 6?

Bit shifts

Trick question! (ish)

6 = 2 * 3, but it is not a power of 2, so the shift trick won't work

We aren't going to worry about binary multiplication by arbitrary numbers at the moment

Hex conversions

We use binary numbers in computers because they are a natural way to store information -- switches are either on or off

Hexadecimal numbers are also used often in the field, but not because they are intrinsically useful

Hexadecimal numbers are useful mainly as a more terse way of writing binary numbers

Hex conversions

Note that $16 = 2^4$

This means that a single hex digit stores exactly as much information as 4 bits

Converting a single hex digit to binary (or vice versa) is simply a matter of practicing small conversions that we know how to do

For example, 0xB = 11 (decimal)

$11 = 1011_2$

Hex conversions

We could do this same process to convert between any two bases:

- convert from first base to base ten
- convert from base ten to second base

What makes binary->hex and hex->binary conversions so easy is that we can do the conversion 4 bits at a time, so we never need to deal with larger numbers

Hex conversions

For example, to convert 0x49 to binary:

- convert the 4 to a 4-bit binary number: 0100
- convert the 9 to a 4-bit binary number: 1001

The solution is 01001001_2 (you can drop the leading 0 if you like)

Hex conversions

The same idea works converting the other way

To convert 11001010_2 to hex:

- convert 1100 to hex: C
- convert 1010 to hex: A

Solution is 0xCA

Hex conversions

If the binary number does not break evenly into chunks of four bits, add 0s to the *left* of the number before converting

To convert 11010_2 to hex:

- rewrite as 00011010_2
- convert 0001 to hex: 1
- convert 1010 to hex: A

Solution is 0x1A

Hex conversions

We often work in terms of **bytes**, which are groups of 8 bits

Rather than writing out 8 bits, it is often more convenient to write a pair of hex digits to represent the same thing

00111000_2 is one byte

0x38 represents the same byte