Number Representation

Tom Kelliher, CS 220

Oct. 10, 2005

1 Administrivia

Announcements

Assignment

Study for exam.

From Last Time

Instruction formats, program build process.

Outline

1. Introduction.
2. Sign-magnitude, one's complement, and two's complement representations.
3. Negation and sign extension.

Coming Up

First exam.

2 Number Representation

1. Weighted, positional number systems:

   (a) Digit weights.

   (b) Digit positions within a number.

2. Conventionally, a base x system uses x numerals (symbols).

   Consider decimal. This really isn't the case. Why? How can we make it so?

3. Consider a four bit binary number: $b_3b_2b_1b_0$. What does b_i contribute to the value of the number?

4. Hexadecimal: For the sake of brevity, binary numbers are often written in hexadecimal form, because of a direct conversion between the two:

Binary	**Hexadecimal**
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

   Examples: convert 110101 to hex and AE to binary.

5. MIPS registers are 32-bits. How do you store 4 in a register?

   What is the range of *unsigned* values representable in the MIPS? In general, with i-bits, how many unsigned values can we represent?

6. What do we do about numbers that are too big? Fractions? The reals?

7. What about negative values? How do we represent them?

   How do we solve this in the decimal system?

   The *sign bit*. Location within a word.

2.1 Sign-Magnitude Representation

Msb is sign, remaining bits are magnitude.

1. Problems with sign-magnitude representation:

   (a) Multiple zero representations.

   (b) Consider the algorithm for `a + b`:

```
if a and b are of the same sign
{
   sign of sum = sign of a;
   magnitude of sum = magnitude of a + magnitude of b;
}
else if magnitude of a > magnitude of b
{
   sign of sum = sign of a;
   magnitude of sum = magnitude of a - magnitude of b;
}
else
{
   sign of sum = sign of b;
   magnitude of sum = magnitude of b - magnitude of a;
}
```

   Symmetric range.

2.2 One's Complement Representation

Msb is sign bit.

1. Positive values have same representation as for unsigned case.
2. To compute the inverse, complement all the bits.
3. Two 0s.
4. Symmetric range.
5. Addition may cause wraparound carries.

2.3 Two's Complement Representation

Again, msb is sign bit.

1. Positive values have same representation as for unsigned case.
2. To compute the inverse, complement all the bits and add 1.
3. One 0.
4. Asymmetric range: one more negative value.

3 2's Complement: Negation, Sign Extension

1. Negation:
   (a) Two's complement is called by that name because

$$2^n - x = -x$$

   Obviously,

$$(2^n - 1) - x = \overline{x}$$

So:

$$\begin{aligned} -x &= 2^n - x \\ &= (2^n - 1) - x + 1 \\ &= \overline{x} + 1 \end{aligned}$$

Hence our algorithm for negating a two's complement number.

2. Sign Extension:

   (a) How do you place a 16-bit *signed* immediate into a register?

   (b) How do you load a *signed* byte into register?

   (c) The sign extension algorithm.

   (d) Load byte *unsigned*.

   (e) How can this simple procedure preserve the value of a negative number? Proof?

3. Signed & unsigned comparison. Or, sometimes 1111 is less than 0000 and sometimes it's not.