Arithmetic

Computers do not store numbers or letters, per se. They only store bit sequences. The bit sequences can be interpreted as representing integers or floating point numbers. Arithmetic is accomplished by the direct hardware implementation of arithmetic algorithms.

Converting Decimal to b-Bit Binary

- Let \( d \) be decimal number, less than \( 2^b - 1 \)

\[ b = 0 \text{ quit} \]

\[ b := b - 1 \]

\[ d > 2^b \]

\[ d = d - 2^b \]

Output 0 in position \( b \)

Output 1 in position \( b \)

Converting 2s Complement

A positive number is its unsigned equivalent, provided the msb is 0.

Finding \(-x\) given \(x\): Complement all bits & add 1

In \( n \)-bit 2s complement, \( x + -x = 2^n \)

Consequences of Signed Fields

A field of \( b \) bits can represent \( 2^b \) configurations.

- If the field is used for unsigned numbers ...
  
  - Range: 0 to \( 2^b - 1 \)
  
- If the field is used for signed numbers ...
  
  - Range: \(-2^{b-1} \) to \( 2^{b-1} - 1 \)

- If the field is used so that some bits are always the same, then do not represent them

Range of byte addresses for instructions:

0 to \( 2^{b-2} \) since least significant bits are 00

Representation

- Terminology
  
  - least significant bit (lsb): least magnitude bit.
  
  - most significant bit (msb): greatest magnitude bit.
  
  - unsigned integers: \( k \) bit sequence representing 0 to \( 2^k - 1 \)
  
  - two's complement: number representation for signed integers.

Unsigned Binary          Decimal

0000 0000 0000 0000 0 0000 0000 0000 0000 0

0000 0000 0000 0001 1 0000 0000 0000 0001 1

0000 0000 0000 0010 2 0000 0000 0000 0010 2

. . . . . .

1111 1111 1111 1101  65533 0111 1111 1111 1111 82767

1111 1111 1111 1110  65534 1000 0000 0000 0000 52428

1111 1111 1111 1111  65535 1000 0000 0000 0001 65535

2s Complement is "unbalanced" since it has 1 more negative number than positive numbers. Why?

Unsigned gets a larger range at the expense of no negative representation.

Subtraction

- The well-known rule that subtraction is equivalent to addition of a negated operand is important in computing

\[ a - b = a + (-b) \]

- In the arithmetic-logic unit of a computer, there is no subtraction circuitry per se, just negation

```
010 . . . 0     a
110 . . . 1     E
+ 000 . . . k
```

Consequences of Signed Fields

A field of \( b \) bits can represent \( 2^b \) configurations.

- If the field is used for unsigned numbers ...
  
  - Range: 0 to \( 2^b - 1 \)

- If the field is used for signed numbers ...
  
  - Range: \(-2^{b-1} \) to \( 2^{b-1} - 1 \)

- If the field is used so that some bits are always the same, then do not represent them

Range of byte addresses for instructions:

0 to \( 2^{b-2} \) since least significant bits are 00

```
000 . . . 0
```

\[ b \]
Representations

A bit sequence is neither signed nor unsigned, integer or floating point, character or pixel... it's just a bit sequence -- the key is how the bits are interpreted.

- The interpretation nearly always matters, especially in comparisons:
  - \(10110 < 00110\) is true since \(-10 < 6\) as 2s complement
  - \(10110 > 00110\) is true since \(22 > 6\) as unsigned

- MIPS has additional comparison operators:
  - \(slt\) $8$, $9$, $10$ #set less than unsigned
  - \(slti\) $8$, $9$, $10$ #set less than immed. unsigned

Why are there no unsigned variants for \(beq\), \(bne\), \(bgtz\), \(blez\), \(sh\), \(sb\), etc.?

Facts of Finite Representation

When combining two numbers produces a result larger than can be represented in the available space, an overflow occurs.

\[
\begin{array}{c}
\text{Overflow occurs when} \\
010 \ldots 0 + 010 \ldots 0 = 100 \ldots 0
\end{array}
\]

Overflow is not always bad, but it must be reportable.

Overflow is impossible when...

- adding numbers with opposite signs because the result is numerically between the operands
- subtracting numbers with like signs, since the "addition" rule applies once B is negated

Conditions Causing Overflow

- For add operations (add and subtract), the overflow can be detected by the sign of the operands and the result.

\[
\begin{array}{c|c}
\text{Overflow} & \text{Op A} & \text{Op B} \\
\hline
A+B & \geq 0 & \geq 0 & <0 \\
A+B & <0 & \geq 0 & \geq 0 \\
A-B & \geq 0 & <0 & <0 \\
A-B & <0 & \geq 0 & \geq 0 \\
\end{array}
\]

Notice that the subtraction rule is simply the Addition rule applied when subtraction is negation of the second operand followed by addition.