# 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.


© 1997 LS All Rights Reserved

# Representation

Terminology --

*binary*: numbering system using 2 as the base value.

*hexadecimal*: numbering system using 16 as base.

*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 0001 | 1 |
| 0000 0000 0000 0010 | 2 |
| . . . | |
| 1111 1111 1111 1101 | 65533 |
| 1111 1111 1111 1110 | 65534 |
| 1111 1111 1111 1111 | 65535 |

| 2s Complement Binary | Decimal |
|---|---|
| 0000 0000 0000 0000 | 0 |
| 0000 0000 0000 0001 | 1 |
| 0000 0000 0000 0010 | 2 |
| . . . | |
| 0111 1111 1111 1111 | 32767 |
| 1000 0000 0000 0000 | -32768 |
| 1000 0000 0000 0001 | -32767 |
| 1000 0000 0000 0010 | -32766 |
| . . . | |
| 1111 1111 1111 1101 | -3 |
| 1111 1111 1111 1110 | -2 |
| 1111 1111 1111 1111 | -1 |

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


© 1997 LS All Rights Reserved

# Converting 2s complement

A positive number is its unsigned equivalent, provided the msb is 0.
Finding -x given x: complement all bits and add 1.
In n bit 2s complement arithmetic: $x + (-x) = 2^n$.

| | | |
|---|---|---|
| | $0000\ 0000\ 0000\ 1010_2$ | $10_{10}$ |
| Compl.: | $1111\ 1111\ 1111\ 0101$ | |
| Add 1: | $+\quad\quad\quad\quad\quad\quad\quad\quad 1$ | |
| | $1111\ 1111\ 1111\ 0110_2$ | $-10_{10}$ |
| Compl.: | $0000\ 0000\ 0000\ 1001$ | |
| Add 1: | $+\quad\quad\quad\quad\quad\quad\quad\quad 1$ | |
| | $0000\ 0000\ 0000\ 1010_2$ | $10_{10}$ |


© 1997 LS All Rights Reserved

# What's Your Sign?

The standard `add`, `sub` and `addi` instructions use signed 2s complement values.

The immediate field is *sign-extended* to make a 32 bit value.

A 2s complement value represented with *x* bits can be converted to the same value in a *y* bit representation *y* > *x*, by replicating the sign bit.

A second set of operations perform unsigned arithmetic:

```
addu $8, $9, $10 # add unsigned

subu $8, $9, $10 # subtract unsigned

addiu  $8, $9,10 # add unsigned immediate
```

- 32-bit value created with leading 0s

$+10 = 0...00\ 1010_2$
$-10 = 1...11\ 0110_2$


© 1997 LS All Rights Reserved

# Representation and Comparison

In a 5-bit field
$1\ 0110_2$
is -10 in 2s C
but 22 as an
unsigned value

MIPS has two kinds of numbers: signed and unsigned.

Signed numbers use 2s complement representation.

Unsigned numbers use standard (positive)
binary representation filling the entire field or word.

Interpreting numbers as 2s complement or unsigned numbers yields different results during comparisons:

10110 < 00110 is true since -10 < 6 as 2s complement

00110 < 10110 is true since 6 < 22 as unsigned numbers

MIPS has additional comparison operators:

```
sltu $8, $9, $10 # set less than unsigned

sltiu $8, $9, 10 # set less than immed. unsigned
```



© 1997 LS All Rights Reserved

# Facts of Finite Representation

When combining two numbers produces a result larger than can be represented in the available space, an overflow occurs.
Overflow is not always bad, but it must be reportable.
Overflow is impossible when . . .

```
  0111
+ 0101
  1100
```

adding numbers with opposite signs

. . . because the result must be "in between"

subtracting numbers with like signs

. . . because "add" rule applies after "negating B"

Conditions causing overflow:

| Operation | Op A | Op B | Overflow |
|---|---|---|---|
| A+B | ≥0 | ≥0 | <0 |
| A+B | <0 | <0 | ≥0 |
| A-B | ≥0 | <0 | <0 |
| A-B | <0 | ≥0 | ≥0 |



© 1997 LS All Rights Reserved

# Logical Operations

MIPS provides operations for bitwise ANDing and ORing

```
and $8, $9, $10   # logical and
or  $8, $9, $10   # logical or
```

Negation?

```
  00000
- 01101
  10011
- 00001
  10010
```

The immediate verions of logical operations are "unsigned"

```
andi   $8, $9, 10    # logical and immediate
ori    $8, $9, 10    # logical or immediate
```

| And | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| Or | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| Negation | |
|---|---|
| 0 | 1 |
| 1 | 0 |



© 1997 LS All Rights Reserved

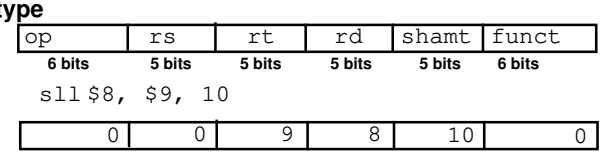
# Shifting Registers

There are two logical shift operations that translate bits left or right and fill vacated positions with 0s

```
sll $8, $9, 10    # shift left logical
srl $8, $9, 10    # shift right logical
```

Shifts are R-type instructions, with the "rs" field ignored. The shift amount is represented in the shamt field.

**R-type**




```
sll $8, $9, 10
```

| 0 | 0 | 9 | 8 | 10 | 0 |
|---|---|---|---|---|---|

Multiplication and division by powers of 2 can be performed for unsigned numbers using shifts left and right, respectively.

Reg 8: 00...0 0011 1000 `sll $8,$8,2` yields Reg 8: 00...0 1110 0000
Reg 8: 00...0 0011 1000 `srl $8,$8,2` yields Reg 8: 00...0 0000 1110



© 1997 LS All Rights Reserved