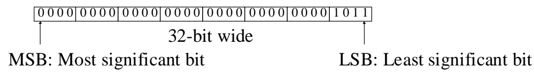
Chapter Three

Arithmetic Operation

- We're going to find out
  - How are negative numbers represented?
  - What is the largest number that can be represented in a computer?
  - What happens if an operation creates a number bigger than can be represented by a computer?
  - What about fraction and real number?

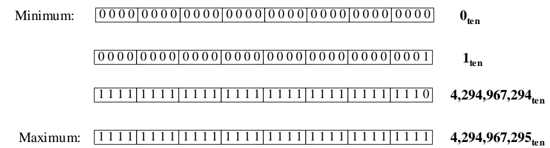
Numbers

- Binary numbers (base 2)
  n-bit 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  Decimal: 0...2^n-1
- In any number base, the value of *i*th digit *d* is d X $Base^i$
- For example: 1011_{two}
- x3 X 2^3 + x2 X 2^2 + x1 X 2^1 + x0 X 2^0
- = 1 X 2^3 + 0 X 2^2 + x1 X 2^1 + 1 X 2^0
- = 11_{ten}
- MIPS word 32-bit long



Possible Representations

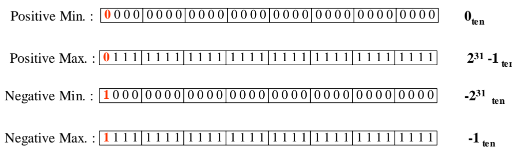
- MIPS word 32-bit long - unsigned numbers
  - 0 ~ 2^{32}-1



- Numbers have infinite number of digits
- What if results of operations cannot be represented by hardware bits?
  - Overflow occurs
  - Handled by OS or program

Possible Representations

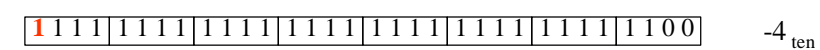
- MIPS word 32-bit long - signed numbers
  - 0 ~ 2^{31}-1 positive number and – 2^{31}~ -1 negative number



- Two's complement
  - Sign and magnitude
  - Hardware only need to check sign bit to see if a number is positive or negative

Possible Representations

- Value of two's complement
  - (x31 X -2^{31}) + (x30 X 2^{30})+ … + (x1 X 2^1) + (x0 X 2^0)



- (x31 X -2^{31}) + (x30 X 2^{30})+ … + (x1 X 2^1) + (x0 X 2^0)
- = -2^{31} + 2^{30} + 2^{29} + . . . + 2^2 + 0 + 0
- = $-2{,}147{,}483{,}648_{ten}$ + $2{,}147{,}483{,}644_{ten}$
- = -4_{ten}

MIPS

- 32 bit signed numbers:

```
0000 0000 0000 0000 0000 0000 0000 0000two = 0ten
0000 0000 0000 0000 0000 0000 0000 0001two = + 1ten
0000 0000 0000 0000 0000 0000 0000 0010two = + 2ten
...
0111 1111 1111 1111 1111 1111 1111 1110two = + 2,147,483,646ten
0111 1111 1111 1111 1111 1111 1111 1111two = + 2,147,483,647ten    maxint
1000 0000 0000 0000 0000 0000 0000 0000two = – 2,147,483,648ten    minint
1000 0000 0000 0000 0000 0000 0000 0001two = – 2,147,483,647ten
1000 0000 0000 0000 0000 0000 0000 0010two = – 2,147,483,646ten
...
1111 1111 1111 1111 1111 1111 1111 1101two = – 3ten
1111 1111 1111 1111 1111 1111 1111 1110two = – 2ten
1111 1111 1111 1111 1111 1111 1111 1111two = – 1ten
```

©2004 Morgan Kaufmann Publishers

Possible Representations

- C language
  - Numbers can be positive or negative: signed number
    - int
  - Numbers can be only positive: unsigned number
    - unsigned int
- How does the hardware know if number is signed or unsigned?

  1000|0000|0000|0000|0000|0000|0000|0000 — Unsigned: 2^{31} ten

  1000|0000|0000|0000|0000|0000|0000|0000 — Signed: -2^{31} ten
  - Comparison instructions must deal with this dichotomy

©2004 Morgan Kaufmann Publishers

Comparison Instructions

- Comparison instructions must deal with signed and unsigned numbers
  - Signed integer: slt and slti
  - Unsigned integer: sltu and sltiu
- Signed vs. unsigned comparison

  \$s0 1111|1111|1111|1111|1111|1111|1111|1111 → -1_{ten} / $4,294,967,295_{ten}$

  \$s1 0000|0000|0000|0000|0000|0000|0000|0001 → 1_{ten}
  - slt \$t0, \$s0, \$s1 # signed comparison $-1_{ten} < 1_{ten}$?
  - sltu \$t1, \$s0, \$s1 # unsigned comparison
    4,294,967,295ten < 1_{ten} ?
  - \$t0=? \$t1=?
    - \$t0=1, \$t1=0,

©2004 Morgan Kaufmann Publishers

Negation Shortcut

- Quick way to negate a two's complement binary number
  - Invert every 0 to 1 and every 1 to 0
  - Add 1 to the result

```
1111|1111|1111|1111|1111|1111|1111|1111     -1 ten
0000|0000|0000|0000|0000|0000|0000|0000
                                      + 1
0000|0000|0000|0000|0000|0000|0000|0001      1 ten
```

©2004 Morgan Kaufmann Publishers

Two's Complement Operations

- Negating a two's complement number: invert all bits and add 1
  - remember: "negate" and "invert" are quite different!
- Converting n bit numbers into numbers with more than n bits:
  - MIPS 16 bit immediate gets converted to 32 bits for arithmetic
  - copy the most significant bit (the sign bit) into the other bits

```
0010  -> 0000 0010
1010  -> 1111 1010
```

©2004 Morgan Kaufmann Publishers

Addition & Subtraction

- Just like in grade school (carry/borrow 1s)

```
  0111       0111       0110
+ 0110     - 0110     - 0101
```

- Two's complement operations easy
  - subtraction using addition of negative numbers

```
  0111
+ 1010
```

- Overflow (result too large for finite computer word):
  - e.g., adding two n-bit numbers does not yield an n-bit number

```
  0111
+ 0001
  1000
```

note that overflow term is somewhat misleading, it does not mean a carry "overflowed"

©2004 Morgan Kaufmann Publishers

Detecting Overflow

- No overflow when adding a positive and a negative number
  - Because sum must be no larger than one of the operands
    - Since operands fit in 32 bits and sum is no larger than an operand, sum must fit in 32 bits
  - 7 + (-6) = 1
- No overflow when signs are the same for subtraction
  - X – Y = X + (-Y)
- Overflow occurs when the value affects the sign:
  - adding two positives yields a negative
    - Sign bit replaced by value
  - adding two negatives gives a positive
    - Sign bit replaced by value
  - subtract a negative from a positive and get a negative
    - Borrow occurs at sign bit
  - subtract a positive from a negative and get a positive
    - Borrow occurs at sign bit

```
  0111
+ 1010
 10001

  1000
+ 1010
 10010
```


Detecting Overflow

Operation	Operand A	Operand B	Result Indicating Overflow
A+B	>=0	>=0	<0
A+B	<0	<0	>=0
A-B	>=0	<0	<0
A-B	<0	>=0	>=0

Detecting Overflow

- Signed number: handle overflow
- Unsigned number: ignore overflow
  - Commonly used for memory addresses
- How does computer recognize or ignore overflow?
  - Instructions cause exception on overflow
    - add and addi
  - Instructions do not cause exception on overflow
    - addu, addiu, subu
- C ignores overflow
  - MIPS compiler will always generate unsigned version of arithmetic instructions
    - addu, addiu, subu

Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Don't always want to detect overflow
  — new MIPS instructions: `addu, addiu, subu`

Floating Point (a brief look)

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576 ´ 10^9
- Scientific notation
  - Have single digit to the left of decimal point
  - e.g. 3.15555 X 10^9
  - Normalized number
    - A number in scientific notation that has no leading 0s
    - e.g. 1.0 X 10^9; 10.0 X 10^9 – not ; 0.1 X 10^9 – not ;
- Binary number in scientific notation
  - Have single digit to the left of binary point
  - e.g. 1.10_{two} X 2^{-1}

Floating Point (a brief look)

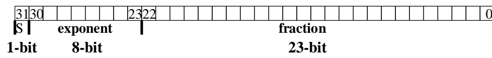
- Floating point
  - Computer arithmetic that represent numbers in which the binary point is not fixed
  - e.g. $1.xxxxxxxxxx_{two}$ X $2^{yyyyyyy}$
- Precision and range
  - For a fixed word size
    - Increase size of fraction → enhance precision
    - Increase size of exponent → increase range
  - MIPS: 32-bit word

31	30 … 23	22 … 0
S	exponent	fraction
1-bit	8-bit	23-bit

  - Floating point number: $(-1)^S$ X F X 2^E

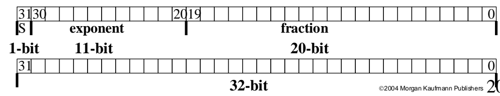
Floating Point (a brief look)

- **Overflow**
  - Exponent is too large to be represented in exponent field
- **Underflow**
  - Non-zero fraction is too small to be represented in fraction field
  - e.g. $1.xxxxxxxxxx_{two} \times 2^{yyyyyyy}$
- **IEEE 754 floating point number**
- **Single precision**
  - Floating point number: $(-1)^S \times F \times 2^E$



Floating Point (a brief look)

- **Reduce overflow and underflow**
- **IEEE 754 floating point number**
  - $(-1)^S \times (1+F) \times 2^E$ (1 is implicit)
- **Single precision**
  - 24 bits for fraction
- **Double precision**
  - 53 bits for fraction



IEEE 754 floating-point standard

- **Leading "1" bit of significand is implicit**
- **Exponent is "biased" to make sorting easier**
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: $(-1)^{sign} \times (1+\text{significand}) \times 2^{\text{exponent} - \text{bias}}$

IEEE 754 Floating Point Number

- **What is IEEE 754 binary representation of -0.75_{ten} in single and double precision?**
- -0.75_{ten} is $-3/4_{ten}$ is $-3/2^2_{ten}$
- Binary fraction $-11_{two}/2^2_{ten}$ or -0.11_{two}
- Normalized scientific notation $-1.1_{two} \times 2^{-1}_{ten}$
- $(-1)^S \times (1+F) \times 2^{(E-127)}$
- $= (-1)^1 \times (1+.1000\ 0000\ 0000\ 0000\ 0000\ 000_{two}) \times 2^{(126-127)}$