Module 2: Data Representation in Computer Systems

BOOK:

COMPUTER ORGANIZATION AND DESIGN, 3ED, DAVID L. PATTERSON AND JOHN L. HANNESSY, MORGAN KAUFMANN PUBLISHERS

CHAPTER 3: ARITHMETIC FOR COMPUTERS

Objectives

- Understand the fundamentals of **numerical data** representation and manipulation in digital computers.
- Master the skill of converting between various **radix systems**.
- Understand how **errors** can occur in computations because of overflow and truncation.
- Understand the fundamental concepts of **floating-point** representation.

Contents

Introduction

- Numbers are represented by binary digits (bits):
  - How are **negative numbers** represented?
  - What is the **largest number** that can be represented in a computer world?
  - What happens if an operation creates a number **bigger than can be represented**?
  - What about **fractions** and **real numbers**?

Number

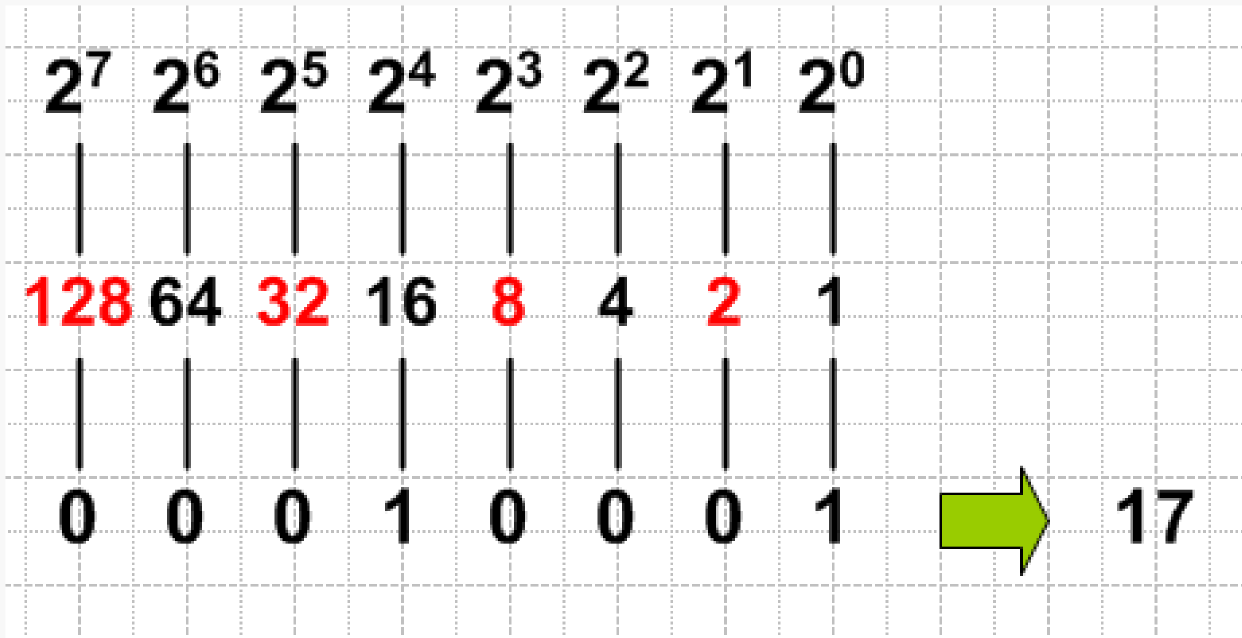
- Numbers are kept in computer hardware as a series of high and low voltage of electronic signals.
- They are considered base 2 numbers (Binary)
- All information is composed of **binary digits** or ***bits***.
- In any number base, the value of *i* th digit of *d* is

$$d \times Base^{i}$$

where *i* start at 0 and increases from right to left

Binary Numbers

- Base 2



d ´ $Base^{i}$

Binary Numbers

1	0	1	1	0	0	1	0
128	64	32	16	8	4	2	1
MSB							LSB

- **Binary Number System**
  - System Digits: 0 and 1
  - Bit (short for *b*inary dig*it*): A single binary digit
  - LSB (least significant bit): The rightmost bit
  - MSB (most significant bit): The leftmost bit
  - Upper Byte (or nybble): The right-hand byte (or nybble) of a pair
  - Lower Byte (or nybble): The left-hand byte (or nybble) of a pair

Binary Numbers

- **Binary Equivalents**
  - 1 Nybble (or nibble) = 4 bits
  - 1 Byte = 2 nybbles = 8 bits
  - 1 Kilobyte (KB) = 1024 bytes
  - 1 Megabyte (MB) = 1024 kilobytes = 1,048,576 bytes
  - 1 Gigabyte (GB) = 1024 megabytes = 1,073,741,824 bytes

Binary Numbers

1	**0**	**1**	**1**	**0**	**0**	**1**	**0**
128	64	32	16	8	4	2	1
MSB							**LSB**

- 8 bits long
- Range of numbers that can be represented;
- = 0 to (2^8 – 1) = 0 to 255

Binary Numbers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

(32 bits wide)

- 32 bits long
- Range of numbers that can be represented =
- = 0 to (2^{32} – 1) = 0 to 4,294,967,295

example

1011_{two}

represents

d ´ $Base^i$

$$
\begin{aligned}
&(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)_{ten} \\
&= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)_{ten} \\
&= 8 + 0 + 2 + 1_{ten} \\
&= 11_{ten}
\end{aligned}
$$

Hence the bits are numbered 0, 1, 2, 3, . . . from *right to left* in a word. The drawing below shows the numbering of bits within a MIPS word and the placement of the number 1011_{two}:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

(32 bits wide)

MIPS Architecture

MIPS (originally an acronym for **Microprocessor without Interlocked Pipeline Stages**) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies (formerly MIPS Computer Systems, Inc.). The early MIPS architectures were 32-bit, with 64-bit versions added later. Multiple revisions of the MIPS instruction set exist, including MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, and MIPS64. The current revisions are MIPS32 (for 32-bit implementations) and MIPS64 (for 64-bit implementations).[1][2] MIPS32 and MIPS64 define a control register set as well as the instruction set.

Several optional extensions are also available, including MIPS-3D which is a simple set of floating-point SIMD instructions dedicated to common 3D tasks,[3] MDMX (MaDMaX) which is a more extensive integer SIMD instruction set using the 64-bit floating-point registers, MIPS16e which adds compression to the instruction stream to make programs take up less room,[4] and MIPS MT, which adds multithreading capability.[5]

Computer architecture courses in universities and technical schools often study the MIPS architecture.[6] The architecture greatly influenced later RISC architectures such as Alpha.

MIPS

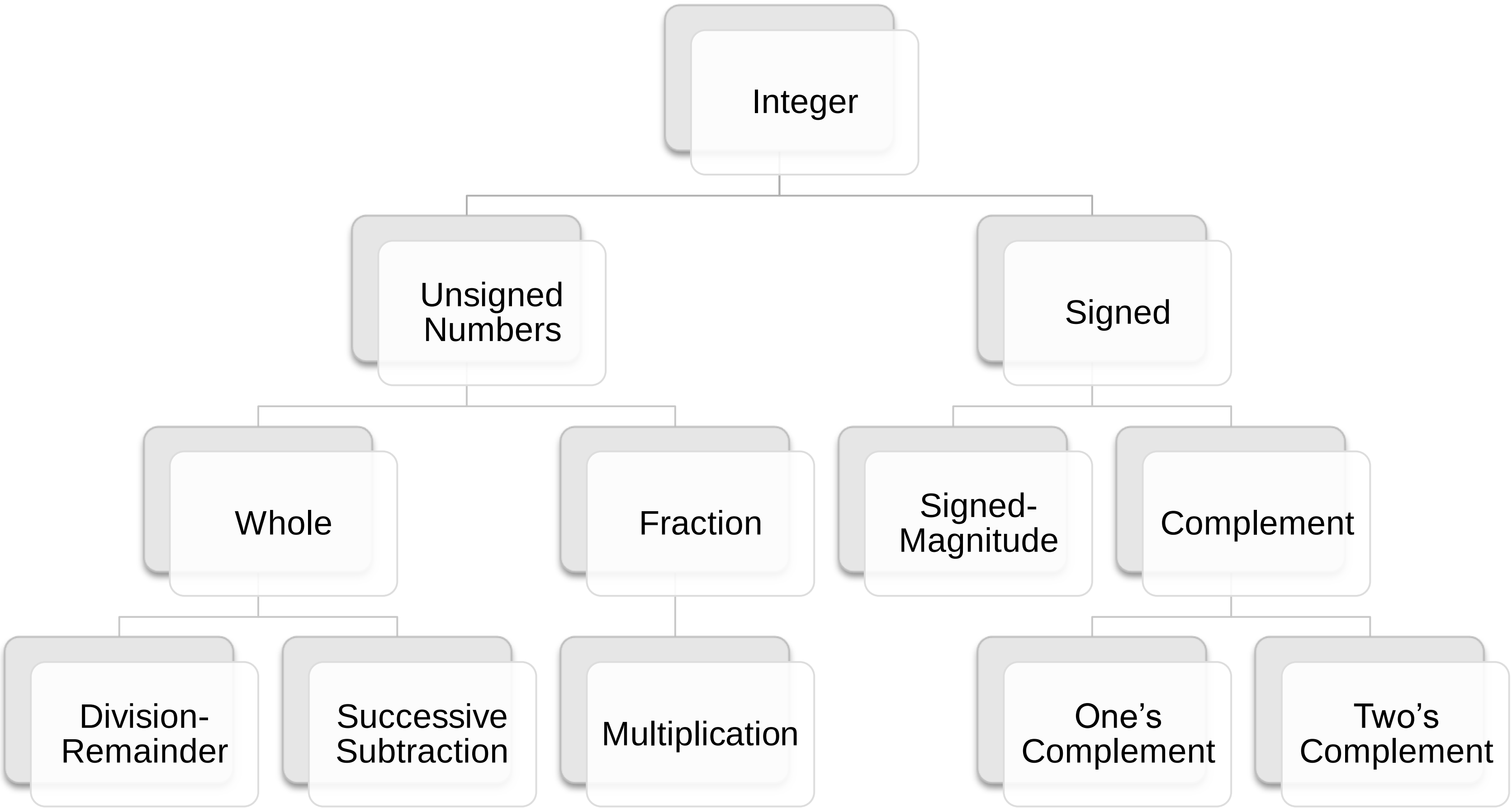
Designer	MIPS Technologies, Inc.
Bits	64-bit (32→64)
Introduced	1981
Design	RISC
Type	Register-Register
Encoding	Fixed
Branching	Condition register
Endianness	Bi
Extensions	MDMX, MIPS-3D
Registers	
General purpose	31 (R0=0)
Floating point	32 (paired DP for 32-bit)

MIPS implementations are primarily used in embedded systems such as Windows CE devices, routers, residential gateways, and video game consoles such as the Sony PlayStation 2 and PlayStation Portable. Until late 2006, they were also used in many of SGI's computer products. MIPS implementations were also used by Digital Equipment Corporation, NEC, Pyramid Technology, Siemens Nixdorf, Tandem Computers and others during the late 1980s and 1990s. In the mid to late 1990s, it was estimated that one in three RISC microprocessors produced was a MIPS implementation.[7]

Fixed-Number Representation

… OR *INTEGER*

Numbers: overview



Unsigned Numbers

- For positive number only.
- Range number = 0 to 2^n-1
- *<u>There is no negative values representation</u>*.

If 32 bits long.
Range of numbers that can be represented
= 0 to (2^{32} – 1) = 0 to 4,294,967,295

Signed Numbers

- Integers as binary values can be positive or negative.
- Problem:
  - *How to represent and encode the actual sign of a number?*
- Solution:
  - Need code to **represent the signs**.
- Three signed representations covered:
  - Signed Magnitude
  - Ones Complement
  - Twos Complement

Signed Numbers: a) Signed Magnitude

- To represent negative values, computer systems allocate the high-order bit to indicate the sign of a value.
  - The high-order bit is the **leftmost** bit in a byte. It is also called the **Most Significant Bit (MSB)**.
  - The remaining bits contain the value of the number.
- If MSB = 0 ➔ number is +ve; If MSB = 1 ➔ number is -ve
- *Note that this leads to having two representations for the number zero*.
- With an 8-bit number representation: Largest number is +127, smallest number is -127

Example:

$+25_{10}$= 0001 1001_2

-25_{10} = 1001 1001_2

In 8 bits numbers:
0000 0000 = +0
0111 1111 = +127
1000 0000 = -0
1111 1111 = -127

Signed Numbers: b) One's Complement

- The **MSB** is the sign bit
- Negative numbers are obtained by flipping (or *complementing*) the bit ➔ **0 to 1**; **1 to 0**
- *Note that this leads to having* ***two representations*** *for the* *number zero*

- With an 8-bit number representation: Largest number is +127, smallest number is -127

Example:

$+25_{10}$= 0001 1001_2

-25_{10} = 1110 0110_2

In 8 bits numbers:
0000 0000 = +0
0111 1111 = +127
1111 1111 = -0
1000 0000 = -127

Signed Numbers: c) Two's Complement

- The **MSB** is the sign bit
- Negative numbers are obtained by *adding 1* to *One's complement negative*.
- *Note that this has only* ***one representation*** *for the number zero*
- With an 8-bit number representation: Largest number is +127, smallest number is -128.

Example:
$+25_{10}$= 0001 1001_2
- 25_{10} = 1110 0111_2

In 8 bits numbers:
0000 0000 = +0
0111 1111 = +127
1111 1111 = -128
1000 0000 = -127

Calculation of 2's Complement

- **Step 1:**
  - invert the binary equivalent of the number by changing all of the ones to zeroes and all of the zeroes to ones (also called 1's complement)
- **Step 2:**
  - Then add one.
- Example: -17

17 = 00010001

Step1 : 11101110

Step2 : 11101110

+ 1

11101111

-17 = 11101111

Sem11415_hkm

Fixed-Number Arithmetic

Arithmetic Operations

- Complement
- Addition
- Subtraction
- Multiplication
- Division

Example: Signed Arithmetic Operation

The operation:
(-19) + 13 = -6

Signed Magnitude:

```
  10010011
+ 00001101
---------------------
  10100000 (-32)
```

Ones Complement:

```
  11101100
+ 00001101
---------------------
  11111001 (-6)
```

Twos Complement:

```
  11101101
+ 00001101
---------------------
  11111010 (-6)
```

Binary Addition

- **Rules of Binary Addition**

  0 + 0 = 0

  0 + 1 = 1

  1 + 0 = 1

  1 + 1 = 0, and carry 1 to the next more significant bit

- Example: 26 + 12

Carry

$$
\begin{array}{cccccccccc}
 & & & & 1 & 1 & & & \\
 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
+ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\hline
 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Binary Addition: Example

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)

$(-8 \le x \le +7)$

Binary Subtraction (a)

- **Rules of Binary Subtraction**

  0 - 0 = 0

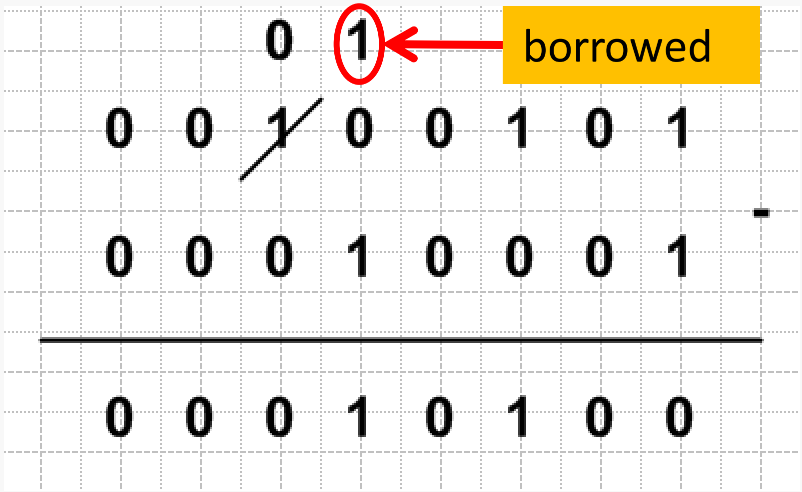
  0 - 1 = 1, and borrow 1 from the next more significant bit

  1 - 0 = 1

  1 - 1 = 0

- Example: 37 -17

- *direct subtraction



Binary Subtraction (b)

- Another method : Using addition with 2's complement
- Example: 37 -17 → 37 + (-17)

```
    0 0 1 0 0 1 0 1
+   1 1 1 0 1 1 1 1
  -----------------
  1 0 0 0 1 0 1 0 0
```

ignore

Binary Subtraction: Example

M ???
S ???

(−8 ≤ x ≤ +7)

<pre> 0010 = 2
 +1001 = −7
  1011 = −5
 	
(a) M = 2 = 0010
    S = 7 = 0111
   −S =   1001</pre>
	<pre> 0101 = 5
	 +1110 = −2
	 10011 = 3
	
	(b) M = 5 = 0101
	    S = 2 = 0010
	   −S = 1110</pre>

<pre> 1011 = −5
 +1110 = −2
 11001 = −7
 	
(c) M = −5 = 1011
    S = 2 = 0010
   −S =   1110</pre>
	<pre> 0101 = 5
	 +0010 = 2
	  0111 = 7
	
	(d) M = 5 = 0101
	    S = −2 = 1110
	   −S = 0010</pre>

<pre> 0111 = 7
 +0111 = 7
  1110 = Overflow
 	
(e) M = 7 = 0111
    S = −7 = 1001
   −S =   0111</pre>
	<pre> 1010 = −6
	 +1100 = −4
	 10110 = Overflow
	
	(f) M = −6 = 1010
	    S = 4 = 0100
	   −S = 1100</pre>

Sign Extension

- Extending a number representation to a larger number of bits.
- Example: **2** in 8 bit binary to 16 bit binary.

00000010 **00000000 00000010**

- In signed numbers, it is important to extend the sign bit to **preserve** the number (+ve or –ve)
- Example: **-2** in 8 bit binary to 16 bit binary.

11111110 **11111111 11111110**

Sign bit

Sign bit extended

Sign bit

Detecting Overflow in Two's Complement Numbers

- Overflow occurs when adding two positive numbers and the sum is negative, or vice versa
  - **Why ??? ➔**

    A carry out occurred into the sign bit

```
 1010 = -6
+1100 = -4
10110 = Overflow
```

Overflow in Humankind History

The $7 billion Ariane 5 rocket, launched on June 4, 1996, veered off course 40 seconds after launch, broke up, and exploded. The failure was caused when the computer controlling the rocket overflowed its 16-bit range and crashed.

The code had been extensively tested on the Ariane 4 rocket. However, the Ariane 5 had a faster engine that produced larger values for the control computer, leading to the overflow.

Overflow Rule for addition

- If 2 Two's Complement numbers are added, and they both have the same sign (both positive or both negative), then overflow occurs if and only if the result has the opposite sign.
  - Adding two positive numbers must give a positive result
  - Adding two negative numbers must give a negative result
- **Overflow** occurs if

$$(+A) + (+B) = -C$$
$$(-A) + (-B) = +C$$

- Overflow never occurs when adding operands with different signs.

Example: Overflow Rule for addition

- Example: Using 4-bit Two's Complement numbers (−8 ≤ x ≤ +7)
- (-7) + (-6) = (**-13**) **but ➔ Overflow** (largest −ve number is −8)

-7 = 1001 **-6 = 1010**

```
    1001
+   1010
-----------
 1  0011 (3)
```

Overflow occurs

The sign bit has changed to +ve

Example: Overflow Rule for addition

- Example: Using 8-bit Two's Complement numbers
- 80 + 80 = 160
- **BUT … result of -96 ➔ Overflow**

```
    01010000
+   01010000
------------------
    10100000 (-96)
```

Not **160** because the sign bit is 1.
(*largest +ve number in 8 bits is 127*)

Overflow Rule for Subtraction

- If 2 Two's Complement numbers are subtracted, and their signs are different, then *overflow occurs* if and only if the result has the same sign as the subtrahend.
- *Overflow occurs* if

  (+A) − **(−B)** = **−**C

  (−A) − **(+B)** = **+**C

result has the same sign as the subtrahend ➔ overflow happens

- Example: Using 4-bit Two's Complement numbers (−8 ≤ x ≤ +7)
- Subtract −6 from +7 (**i.e. 7 – (-6)**) ….

Example: Overflow Rule for Subtraction

- Example: Using 4-bit Two's Complement numbers ($-8 \le x \le +7$)
- Subtract −6 from +7 (i.e. 7 – $\boxed{(-6)}$)

subtrahend

result

$(+A) - (-B) = -C$

7 = 0111

6 = 0110

```
   0111
+  0110
-----------
   1101 (-3)
```

Overflow occurs

Result has same sign as subtrahend (-6)

A little summary

- **Addition**
  - Add the values, discarding any carry-out bit
- **Subtraction**
  - Negate the subtrahend and add, discarding any carry-out bit
- **Overflow**
  - Occurs when adding two positive numbers produces a negative result, or when adding two negative numbers produces a positive result.
  - Adding operands of different signs never produces an overflow

A little summary

- **Overflow**
  - Overflow conditions for addition and subtraction

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

A little summary

- **Overflow**
  - Notice that discarding the carry out of the most significant bit during Two's Complement addition is a normal occurrence, and does not by itself indicate overflow

Binary Addition: Example

$\begin{array}{rcr} 1001 & = & -7 \\ +\underline{0101} & = & 5 \\ 1110 & = & -2 \end{array}$	$\begin{array}{rcr} 1100 & = & -4 \\ +\underline{0100} & = & 4 \\ 10000 & = & 0 \end{array}$
(a) $(-7) + (+5)$	(b) $(-4) + (+4)$

... multiplication

- ...
  - ~~Complement~~
  - ~~Addition~~
  - ~~Subtraction~~
  - Multiplication
  - Division