Computer Architecture. Week 3

Muhammad Fahim, Alexander Tormasov

Innopolis University

m.fahim@innopolis.ru
a.tormasov@innopolis.ru

September 17, 2020

Topic of the lecture

- Number Systems

Topic of the tutorial

- FPGA Tutorial (Part II)

Topic of the lab

- Hands on FPGA board
- Simple example of operations on numbers

Content of the class

- Number System
- The MIPS Number System
- Numbers and Numerals
- Numeric Systems
- Base Conversion
- Finite Precision Numbers
- Positive and Negative Numbers
- Binary and Floating Point Arithmetic Operations
- Standard IEEE 754
- Fractional Representation

Number System

- A set of values used to represent different quantities is known as Number System

Number System for Human

- As humans, we generally count and perform arithmetic using decimal having 10 digits from 0 to 9.
- Historically, it seems that the main reason we usedecimal (i.e., base 10) is that humans have ten fingers
- Numbers may be represented in any base.
  - For example, 123 base 10 = 1111011 base 2.

Number System for Computers

- Numbers are kept in computer hardware as a series of high and low electronic signals
- Computers perform all of their operations using the binary (base 2) number system.
- All program code and data are stored and manipulated in binary form.
- Calculations are performed using binary arithmetic.
- Each digit in a binary number is known as a bit (for binary digit) and can have only one of two values, 0 or 1.

The MIPS Number System

- Microprocessor without Interlocked Pipeline Stages
- MIPS developed at Stanford by Hennessey and the team.
- MIPS Computer Systems founded 1984.
- MIPS in 1998; spun it out as MIPS Technologies

CISC vs. RISC Architectures

- A Complex Instruction Set Computer (CISC) is an alternative. Intel's x86 is the most prominent example; also Motorola 68000 and DEC VAX.
- MIPS is a Reduced Instruction Set Computer (RISC). Others include ARM, PowerPC, SPARC, HP-PA, and Alpha.
- RISC's underlying principles are:
  - Simplicity favors regularity
  - Make the common case fast
  - Smaller is faster
  - Good design demands good compromises

The MIPS Number System

- The drawing below shows the numbering of bits within a MIPS word and the placement of the number

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)

Example:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$$
$$\ldots\ldots$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = 4\ 294\ 967\ 293_{10}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 4\ 294\ 967\ 294_{10}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 4\ 294\ 967\ 295_{10}$$

Numbers and Numerals

- **Number**: abstract entity
- **Numeral**: string of symbols that represent a number in a given system
- The same number can be represented by different numerals in different numeric systems
- Example I
  - 234 in decimal system
  - CCXXXIV using roman system
- Example II
  $(10)_{10} = 10$
  $(10)_2 = (2)_{10} = 2$

Numeric Systems – 1/5

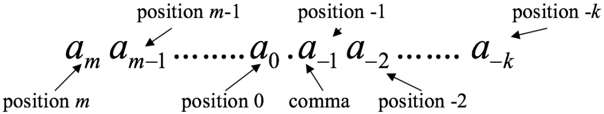
- To define a Numeric System we need:
  - A set of symbols that we will call digits (E.g. 1,2,3, A,C, ...)
  - Some rules to build up numbers
- We can define Positional Numeric System or Non Positional Numeric Systems
- Non Positional Numeric System: the value of digits in the number is position independent
- Example: (Roman Numeric System)
  the symbol V means 5 always, but IV $\neq$ VI . . . .

Numeric Systems – 2/5

- Positional Numeric System: digit value depends on its position within the number (weight)
  - Each digit represents the coefficient of a power of the base
  - Exponent is given by the position of the digit within the number

$$\text{base} = b$$

$$\text{used symbols} = 0 \leq a_i \leq b - 1$$

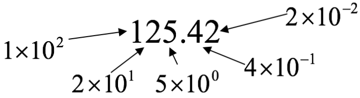


$$\mathrm{N} = \sum_{i=-k}^{m} a_i \, b^i$$

- Example (The Decimal System)

$$\text{base} = 10$$

$$\text{used symbols} = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

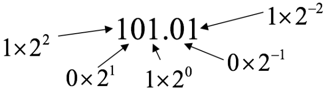


$$125.42 = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0 + 4 \times 10^{-1} + 2 \times 10^{-2}$$

- Example (The Binary System)

$$\text{base} = 2$$

$$\text{used symbols} = 0, 1$$



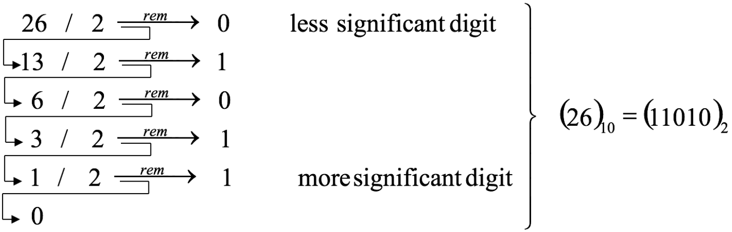
$$101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

Numeric Systems – 5/5

- Other Systems Used
- Octal System
  Base = 8
  Symbols used = 0,1,2,3,4,5,6,7
- Hexadecimal System:
  Base = 16
  Symbols used = 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Base Conversion (decimal to binary)

- It is obtained by repeatedly dividing the number by 2, the remainder of division at each step is the digit of the binary number.



Base Conversion (binary to decimal)

- Generalizing the point, in any number base, the value of ith digit d is d * Base^i
- For example, 1101_2 represents:

$$
\begin{aligned}
&((1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10} \\
&= ((1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1))_{10} \\
&= (8 + 4 + 0 + 1)_{10} \\
&= 13_{10}
\end{aligned}
$$

Finite Precision Numbers

- Definition: numbers with a finite number of digits.
- Some properties are lost:
  - Operators closure
  - Distributive and associative properties
  - Holes in the representation of real numbers

Finite Precision Numbers – Example 1

- Let's use integer numbers with 2 decimal digits and sign
  - Interval represented: [-99, +99]
  - Lost closure with respect to operator +
    76+30 = ???
    (106 is out of the interval)
  - Lost associativity: 25+(90-30) $\neq$ (25+90)-30

Finite Precision Numbers – Example 2

- Let's use consider rational numbers with two decimal digit after the point
  - Interval represented: [-0.99, +0.99]
  - We cannot represent any additional number between 0.05 and 0.06
  - Again lost closure with respect to operator +: 0.90+0.30 = ??? (1.20 is out of the interval)
  - Again lost associativity: 0.25+ (0.90-0.30) $\neq$ (0.25+ 0.90)-0.30

- By representing positive integers in binary notation, n digits (bits), cover the interval $[0, 2^n\text{-}1]$
- It is easy to note that if the maximum number re-presentable using n bit is

$$X = 2^n - 1$$

then to represent number X, the necessary number of bits is

$$n = \text{Int}(\log_2 (X)+1)$$

- Example: using n = 3 interval [0, 7] is completely represented

0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Signed Number Representation

- In mathematics, positive numbers (including zero) are represented as unsigned numbers. That is we do not put the +ve sign in front of them to show that they are positive numbers.
- However, when dealing with negative numbers we do use a -ve sign in front of the number to show that the number is negative.
- However, in digital circuits there is no provision made to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of 0's and 1's.

Sign and Magnitude Representation

- Sign and Magnitude is one of the method used to represent signed numbers in binary format
  - The first bit is used for the sign 0 mean + ; 1 mean –
  - n-1 bits are used for the magnitude
  - Represented interval: [$-2^{n-1}+1$, $2^{n-1}-1$]
- Example: Using n=4 interval [-7, 7] is completely represented

  5 0101

  -5 1101

Issues with Sign and Magnitude

Example: Using n=4 interval [-7, 7] is completely represented

Pattern	Value Represented Sign Magnitude
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

- Having two patterns to represent 0 is wasteful.
- The signed magnitude representation has the advantage that it is easy to read the value from the pattern.
- But does it have the binary arithmetic property?
- For instance, what is the result of pattern(-1) + pattern(1)?

One's Complement Representation

- One's complement number representation is used for signed numbers in binary format
  - The leftmost bit defines the sign of the number (0 for +; 1 for -)
  - The integer is changed to binary (the sign is ignored).
  - 0s are added to the left of the number to make a total of N bits
  - If the sign is positive, no more action is needed. If the sign is negative, every bit is inverted
  - Represented interval: [$-2^{n-1}+1$, $2^{n-1}-1$]

Example: using n=4 interval [-7, 7] is completely represented

5 0101

-5 1010

Issues with One's Complement

Pattern	Value Represented	
	Sign Magnitude	1's complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-0	-7
1001	-1	-6
1010	-2	-5
1011	-3	-4
1100	-4	-3
1101	-5	-2
1110	-6	-1
1111	-7	-0

- A negative number is represented by flipping all the bits of a positive number.
- We still have 2 patterns for 0.
- It is still easy to read a value from a given pattern.
- How about the arithmetic property?
- Suggestion: try the following
  -1 + 1 = ??
  -0 + 1 = ??
  0 + 1 = ??

Two's Complement Representation (1/2)

- It is most common and widely used representation today.
  - The leftmost bit defines the sign of the number (0 for +; 1 for -)
  - The integer is changed to binary, (the sign is ignored).
  - 0s are added to the left of the number to make a total of N bits
  - If the sign is positive, no more action is needed. If the sign is negative, every bit is complemented and 1 is added.
  - Represented interval: [-2^{n-1} , 2^{n-1} -1]

Example

using n=4 interval [-8, 7] is completely represented

5 0101

-5 1011

Two's Complement Representation (2/2)

- **Example:** We want to find how -28 would be expressed in two's complement notation (8-bit representation)
- **Solution**
  - The integer is changed to binary 00011100
  - Invert the bits 11100011
  - Add 1
  - Number representation is 11100100

Signed Number Representation (Summary)

Pattern	Value Represented		
	Sign Magnitude	1's complement	2's complement
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Note: we do not "lose" any more a number: 0 has exactly one representation.
It holds the arithmetic property, but the reading of a negative pattern is not trivial.

Excess Notation

- Excess 8 notation indicates that the value for zero is the bit pattern for 8, that is 1000
- The bit patterns are 4 bits long
- Positive numbers are above it in order and negative numbers are below it in order.

An Excess Eight Conversion Table

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

Excess Notation (Continue)

- That is the zero point for Excess 128 notation is 128; the zero point for excess 64 notation is 64; and so forth.
- For example, let's say we want to determine the pattern for 15 in Excess 128 notation.
- The decimal number would be 128 + 15, or 143. Therefore, the bit pattern would be 10001111

Excess Notation (Continue)

- **Example:** Represent –25 in Excess 127 using an 8-bit allocation.
- **Solution:**
  - 127 + (-25) = 102
  - 102 in binary 1100110
  - Add 0's to the left to make it 8 bit 01100110

Note: again we do not "lose" any more a number: 0 has exactly one representation

$$\boxed{\text{stored number} = \text{actual number} + \text{excess notation}}$$

$$\boxed{\text{actual number} = \text{stored number} - \text{excess notation}}$$

Binary Arithmetic: Sum

- Sum in binary notation is performed bit by bit carrying the rest to next digit

$$
\begin{aligned}
&0+0=0 \\
&0+1=1 \\
&1+0=1 \\
&1+1=0 \qquad \text{carry } 1
\end{aligned}
$$

- Example

$$
\begin{array}{r}
3+ \\
2 \\
\hline
5
\end{array}
\qquad
\begin{array}{l}
0011+ \\
0010 \\
\hline
0101
\end{array}
$$

- In 2's C sum and subtraction are managed in the same way, just throw away carry.

$$\begin{array}{r} 65+ \\ \text{-}2 \\ \hline 63 \end{array} \qquad (1)\ \begin{array}{r} 0100\ 0001+ \\ 1111\ 1110 \\ \hline 0011\ 1111 \end{array}$$

- If two terms have different sign the result is always correct.
- If two terms have the same sign but the result has a different one... we have an ERROR
- The sum 100 + 100 using 2 bit representation

$$\begin{array}{r} 0110\ 0100+ \\ 0110\ 0100 \\ \hline 1100\ 1000 \end{array} \qquad \text{Overflow error!}$$

What will be the output of the following program?

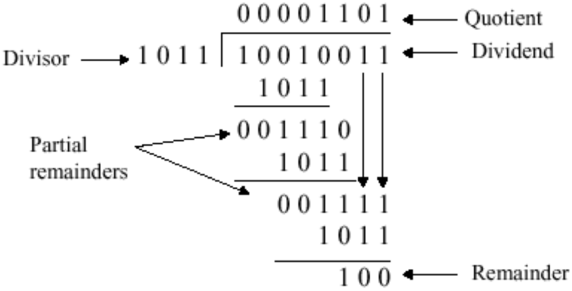
```
#include <stdio.h>
#include <limits.h>

int main()
{
    int count;
    count =INT_MAX;      //2147483647
    count = count +1;
    printf("%d", count);
}
```

Output: -2147483648

Binary Arithmetic: Multiplication

$$
\begin{array}{rcccccccclll}
 & & & & & 1 & 1 & 0 & 1 & & (13)_{10} & \text{Multiplicand M} \\
 & & & & \times & 1 & 0 & 1 & 1 & & (11)_{10} & \text{Multiplier Q} \\
\cline{5-9}
 & & & & & 1 & 1 & 0 & 1 & \left.\rule{0pt}{0pt}\right] & & \\
 & & & & 1 & 1 & 0 & 1 & & & & \text{Partial products} \\
 & & & 0 & 0 & 0 & 0 & & & & & \\
 & & 1 & 1 & 0 & 1 & & & & & & \\
\cline{2-9}
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & & (143)_{10} & \text{Product P}
\end{array}
$$

Binary Arithmetic: Division



- Representation of floating point number is not unique.

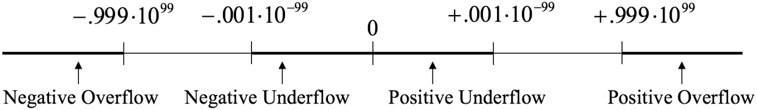
- For example: The number 55.66 can be represented as
  5.566×10^1
  0.5566×10^2
  0.05566×10^3 and so on..

- It is important to note that floating-point numbers suffer from loss of precision when represented with a fixed number of bits (e.g., 32-bit or 64-bit).
- Reason: This is because there are infinite number of real numbers (even within a small range of says 0.0 to 0.1).

Floating-Point Number Representation

- On the other hand, a $n-bit$ binary pattern can represent a finite 2^n distinct numbers.
- Hence, not all the real numbers can be represented. The nearest approximation will be used instead, resulted in loss of accuracy.
- Modern computers adopt IEEE 754 standard for representing floating-point numbers.
- IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.
- There are two representation schemes:
  - 32-bit single-precision
  - 64-bit double-precision.

- Example: Using Base 10
- Using numerals with 5 digits of the kind $\pm$ .XXX $\pm$EE
- Mantissa = $\pm$ .XXX three signed digits $0.001 \leq m < 1$
- Exponent = $\pm$EE two signed digits $-99 \leq e \leq 99$
- Represented numbers are:



Represented interval is
$-.999 \cdot 10^{99} \leq N \leq -.001 \cdot 10^{-99}; \quad .001 \cdot 10^{-99} \leq N \leq .999 \cdot 10^{99}$

Standard IEEE 754 (1985)

- Standard definition (i.e., architecture independent)
- Single precision (uses 32 bits to represent sign, exponent and mantissa

+/- (1 bit)	Exponent (8 bits)	Mantissa (23 bits)

- Double precision (uses 64 bits)

+/- (1 bit)	Exponent (11 bits)	Mantissa (52 bits)

- Some configurations of the exponent are reserved (i.e. not standardized)

Unum (Number Format)

- The unum (universal number) format is a format similar to floating point, proposed by John Gustafson.
- It is an alternative to the now ubiquitous IEEE 754 format.
- Unum implementations have been explored in Julia.
  - Julia is a high-level, high-performance dynamic programming language for numerical computing.

Ternary Number System

- The ternary numeral system also called base-3.
- Analogous to a bit, a ternary digit is a trit (**tr**inary dig**it**).
- One trit is equivalent to $log_2 3$ (about 1.58496) bits of information.
- The ternary numeral system has three possible values:
  - Standard (unbalanced) system: Which uses the values 0, 1 and 2
  - Balanced system: Which uses the values -1, 0, 1

Ternary Number System

- The first modern electronic ternary computer **Setun** was built in 1958 in the Soviet Union at the Moscow State University by Nikolay Brusentsov.

- Why not famous as binary system?
  - It is much harder to build components that use more than two states.
  - If you use more than two states you need to be compatible to binary, because the rest of the world uses it.

Saturation Arithmetic – 1/2

- It is a version of arithmetic in which all operations such as addition and multiplication are limited to a fixed range between a minimum and maximum value.
- If the result of an operation is greater than the maximum, it is clamped to the maximum.
- If it is below the minimum, it is clamped to the minimum
  - For example, if the valid range of values is from -100 to 100

$$
\begin{aligned}
60 + 30 &= 90 \\
60 + 43 &= 100 \\
(60 + 43) - (75 + 75) &= 0 \\
99 \times 99 &= 100 \\
30 \times (5 - 1) &= 100
\end{aligned}
$$

Saturation Arithmetic – 2/2

- Saturation arithmetic enables efficient algorithms for many problems, particularly in digital signal processing.
- For example:
  - Adjusting the volume level of a sound signal can result in overflow, and saturation causes significantly less distortion to the sound than wrap-around.

Summary

- The MIPS Number System
- Numeric Systems
- Base Conversion
- Finite Precision Numbers
- Positive and Negative Numbers
- Binary and Floating Point Arithmetic Operations
- Standard IEEE 754 (Floating point)
- Ternary Number System
- Unum (Universal Number)
- Saturation Arithmetic

Acknowledgements

- This lecture was created and maintained by Muhammad Fahim, Giancarlo Succi and Alexander Tormasov