Computer Architecture 2023-24 (WBCS010-05)

Lecture 2: Number Systems

Reza Hassanpour
r.zare.hassanpour@rug.nl

Introduction

- Computers work with digital signals, which are discrete and have only two possible states: 0 or 1.
- Binary numbers are the most natural way to represent these two states.
- The binary system simplifies the representation of digital data by aligning with the on/off states of electronic switches.
- Question:
  - How can we represent decimal numbers in binary?

Positional and Non-Positional Number Systems

- Positional number system is the type of number system in which the value of a digit depends upon its position in the number.
  - Ex. Decimal Numbers

- In non-positional number systems, any symbol has a specific value regardless of its position.
  - Ex. Roman Numbers I, II, III, XX, XXV

Decimal Number System

- The decimal number system is a positional number system.
  - Example:

$3 \quad 7 \quad 1 \quad 1$

$10^3 \; 10^2 \; 10^1 \; 10^0$

$3 \text{ X } 10^0 = 1$

$7 \text{ X } 10^1 = 20$

$1 \text{ X } 10^2 = 600$

$1 \text{ X } 10^3 = 5000$

Decimal Number System

- The decimal number system is also known as base 10.
- The values of the positions are calculated by taking 10 to some power.
- It uses 10 digits, the digits 0 through 9.

Binary Number System

- The binary number system is also known as base 2. The values of the positions are calculated by taking 2 to some power.
- Why is the base 2 for binary numbers?
- Because we use 2 digits, the digits 0 and 1.

Binary Number System

- The binary number system is also a positional numbering system.
- Instead of using ten digits, 0 - 9, the binary system uses only two digits, 0 and 1.
- Example of a binary number and the values of the positions:

1	0	0	1	1	0	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0

Conversion From Decimal To Binary

- Use repeated division by radix.
- Example: $56_{(10)}$

	Quotient	Remainder
56÷2=	28	0
28÷2=	14	0
14÷2=	7	0
7÷2=	3	1
3÷2=	1	1
1÷2=	0	1

- Collect remainders in reverse order
- $56_{(10)} = 111000_2$

Conversion From Binary To Decimal

› Multiply each digit by radix to the power of the position of the digit.

› Add up to get the decimal number

Conversion from binary to decimal

$$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5_{10}$$

university of groningen

Signed Integers

- How to represent positive (+) and negative (–)?
- With n bits, we have 2^n encodings
  - Use half of the patterns (the ones starting with zero) to represent positive numbers
  - Use the other half (the ones starting with one) to represent negative numbers
- Three different encoding schemes:
  - *Signed-magnitude*
  - *1's complement*
  - *2's complement*

Sign-Magnitude

- Uses one bit to represent the sign

  0 = positive,   1 = negative
- Remaining bits are used to represent the magnitude
- Range $-(2^{n-1} - 1)$ to $2^{n-1} - 1$

  where n=number of digits
- Example: Let n=4:
- Range is –7 to 7 or

  1111   to 0111

1's complement

- To get a negative number, start with a positive number (with zero as the leftmost bit) and flip all the bits -- from 0 to 1, from 1 to 0

- Examples -- 5-bit 1's complement integers:

00101	(5)	01001	(9)
11010	(−5)	10110	(−9)

Signed-Magnitude and 1's Complement Disadvantages

› In both representations, two different representations of zero

- Signed-magnitude: 00000 = 0 and 10000 = 0
- 1's complement: 00000 = 0 and 11111 = 0

› Operations are not simple

- Think about how to add +2 and −3.
- Actions are different for two positive integers, two negative integers, one positive and one negative

› A simpler scheme: 2's complement

university of groningen

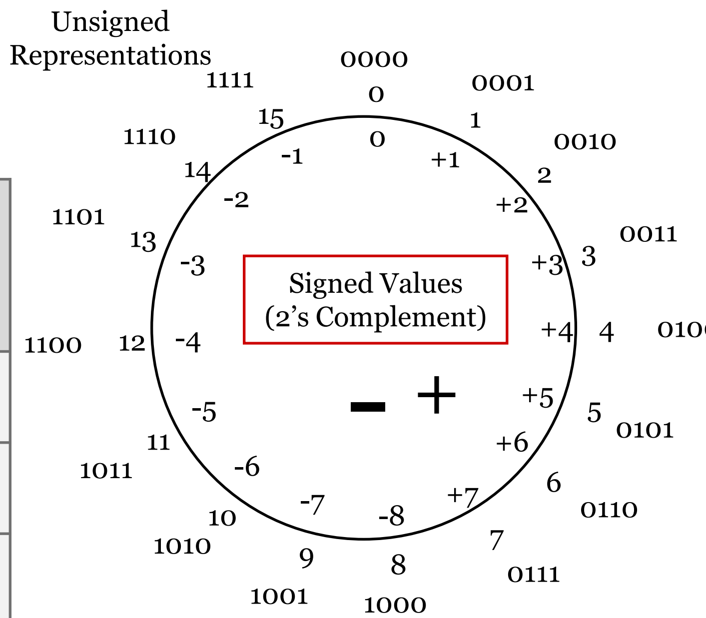
2's Complement

- To simplify circuits, we want all operations on integers to use binary arithmetic, regardless of whether the integers are positive or negative
- When we add +X to –X we want to get zero
- Therefore, we use "normal" unsigned binary to represent +X. And we assign to –X the pattern of bits that will make X + (–X) = 0

$$\begin{array}{rl} 00101 & (5) \\ +\ 11011 & (-5) \\ \hline 00000 & (0) \end{array} \qquad \begin{array}{rl} 01001 & (9) \\ +\ 10111 & (-9) \\ \hline 00000 & (0) \end{array}$$

- **NOTE:** When we do this, we get a carry-out bit of 1, which is ignored
- We only have 5 bits, so the carry-out bit is ignored and we still have 5 bits

2's Complement Integers



4-bit 2's complement	value	4-bit 2's complement	value
0000	0		
0001	1	1111	−1
0010	2	1110	−2
0011	3	1101	−3
0100	4	1100	−4
0101	5	1011	−5
0110	6	1010	−6
0111	7	1001	−7
		1000	−8

- With n bits, represent values from -2^{n-1} to $+2^{n-1}-1$
- NOTE: All positive numbers start with 0, all negative numbers start with 1

Signed Numbers: 4-bit Example

Decimal	2's comp	Sign-Mag
-8	1000	N/A
-7	1001	1111
-6	1010	1110
-5	1011	1101
-4	1100	1100
-3	1101	1011
-2	1110	1010
-1	1111	1001
-0	0000 (= +0)	1000

university of groningen

Converting X to –X

1. Flip all the bits (Same as 1's complement)
2. Add +1 to the result

00101	(5)	10111	(−9)
11010	(1's comp)	01000	(1's comp)
+ 1		+ 1	
11011	(−5)	01001	(9)

› *A shortcut method:*

› Copy bits from right to left up to (and including) the first '1'. Then flip remaining bits to the left

Converting Binary (2's C) to Decimal

1. If leading bit is one, take two's complement to get a positive number
2. Add powers of 2 that have "1" in the corresponding bit positions
3. If original number was negative, add a minus sign

$$X = 01101000_{two}$$

$$= 2^6 + 2^5 + 2^3 = 64 + 32 + 8$$

$$= 104_{ten}$$

› *Examples use 8-bit 2's complement numbers*

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1,024

More Examples

$$X = 00100111_{two}$$
$$= 2^5 + 2^2 + 2^1 + 2^0 = 32 + 4 + 2 + 1$$
$$= 39_{ten}$$

$$X = 11100110_{two}$$
$$-X = 00011010$$
$$= 2^4 + 2^3 + 2^1 = 16 + 8 + 2$$
$$= 26_{ten}$$
$$X = -26_{ten}$$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1,024

› *Examples use 8-bit 2's complement numbers*

Converting Decimal to Binary (2's C) 1

› First Method: *Division*

1. Find magnitude of decimal number (conversion always positive)
2. Divide by two – remainder is least significant bit
3. Keep dividing by two until answer is zero, writing remainders from right to left
4. Append a zero as the MS bit; if original number was negative, flip bits and add +1

$$X = 104_{ten}$$

$$
\begin{array}{rl}
104/2 = 52\ r0 & \textit{bit 0} \\
52/2 = 26\ r0 & \textit{bit 1} \\
26/2 = 13\ r0 & \textit{bit 2} \\
13/2 = 6\ r1 & \textit{bit 3} \\
6/2 = 3\ r0 & \textit{bit 4} \\
3/2 = 1\ r1 & \textit{bit 5} \\
1/2 = 0\ r1 & \textit{bit 6}
\end{array}
$$

$$X = 01101000_{two}$$

university of groningen

Converting Decimal to Binary (2's C) 2

› Second Method: *Subtract Powers of Two*

1. Find magnitude of decimal number
2. Subtract largest power of two less than or equal to number
3. Put a one in the corresponding bit position
4. Keep subtracting until result is zero
5. Append a zero as MS bit; if original was negative, flip bits and add +1

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1,024

$$X = 104_{\text{ten}}$$

$$104 - 64 = 40 \quad \text{bit } 6$$

$$40 - 32 = 8 \quad \text{bit } 5$$

$$8 - 8 = 0 \quad \text{bit } 3$$

$$X = 01101000_{\text{two}}$$

Fractions: Fixed-Point Binary

- We use a "binary point" to separate integer bits from fractional bits, just like we use the "decimal point" for decimal numbers
- Two's complement arithmetic still works the same

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

$$
\begin{array}{rl}
 & 00101000.101 \quad (40.625) \\
+ & 11111110.110 \quad (-1.25) \\
\hline
 & 00100111.011 \quad (39.375)
\end{array}
$$

- **NOTE:** In a computer, the binary point is **implicit** -- it is not explicitly represented. We just interpret the values appropriately

Converting Decimal Fraction to Binary (2's C)

1. Multiply fraction by 2
2. Record one's digit of result, then subtract it from the decimal number
3. Continue until you get 0.0000... or you have used the desired number of bits (in which case you have approximated the desired value)

$$X = 0.421_{ten}$$

$$0.421 \times 2 = 0.842 \quad \text{bit } -1 = 0$$

$$0.842 \times 2 = 1.684 \quad \text{bit } -2 = 1$$

$$0.684 \times 2 = 1.368 \quad \text{bit } -3 = 1$$

$$0.368 \times 2 = 0.736 \quad \text{bit } -4 = 0$$

until 0, or until no more bits...

$X = 0.0110_{two}$ › (if limited to 4 fractional bits)

Operation: Addition

› As discussed, 2's complement addition is just binary addition

- Assume operands have the same number of bits
- Ignore carry-out

$$
\begin{array}{rrl}
 & \texttt{01101000} & (104) \\
+ & \texttt{11110000} & (-\ 16) \\
\hline
 & \texttt{01011000} & (98)
\end{array}
\qquad
\begin{array}{rrl}
 & \texttt{11110110} & (-10) \\
+ & \texttt{11110111} & (-9) \\
\hline
 & \texttt{11101101} & (-19)
\end{array}
$$

Operation: Subtraction

- You can, of course, do subtraction in base-2, but easier to negate the second operand and add
- Again, assume same number of bits, and ignore carry-out

	01101000	(104)		11110110	(−10)
−	00010000	(16)	−	11110111	(−9)
	01101000	(104)		11110110	(−10)
+	11110000	(−16)	+	00001001	(9)
	01011000	(88)		11111111	(−1)

Operation: Sign Extension

- To add/subtract, we need both numbers to have the same number of bits. What if one number is smaller?
- Padding on the left with zero does not work:

4-bit	8-bit
0100 (+4)	00000100 (still +4)
1100 (−4)	00001100 (12, not −4)

- Instead, replicate the most significant bit (the sign bit):

4-bit	8-bit
0100 (+4)	00000100 (still +4)
1100 (−4)	11111100 (still −4)

Overflow

- If the numbers are two big, then we cannot represent the sum using the same number of bits
- For 2's complement, this can only happen if both numbers are positive or both numbers are negative

	01000	(8)		11000	(−8)
+	01001	(9)	+	10111	(−9)
	10001	(−15)		01111	(+15)

- How to test for overflow:

1. Signs of both operands are the same, AND
2. Sign of the sum is different

university of groningen

Hexadecimal Notation: Binary Shorthand

› To avoid writing long (error-prone) binary values, group four bits together to make a base-16 digit. Use A to F to represent values 10 to 15

binary (base 2)	hexadecimal (base 16)	decimal (base 10)	binary (base 2)	hexadecimal (base 16)	decimal (base 10)
0000	**0**	0	1000	**8**	8
0001	**1**	1	1001	**9**	9
0010	**2**	2	1010	**A**	10
0011	**3**	3	1011	**B**	11
0100	**4**	4	1100	**C**	12
0101	**5**	5	1101	**D**	13
0110	**6**	6	1110	**E**	14
0111	**7**	7	1111	**F**	15

› Example: Hex number **3D6E** easier to communicate than binary **0011110101101110**

Converting from binary to hex

› Starting from the right, group every four bits together into a hex digit. Sign-extend as needed.

```
 011 1010 1000 1111 0100 1101 0111
  ↓    ↓    ↓    ↓    ↓    ↓    ↓
  3    A    8    F    4    D    7
```

This is not a new machine representation or data type, just a more convenient way to write the numbers

Very Large or Very Small Numbers: Floating-Point

Large Values: 602300000000000000000000
Small Values: 0.0000000000000000000000000000000006626

- Large values: 6.023×10^{23} -- requires 79 bits
- Small values: 6.626×10^{-34} -- requires > 110 bits
- Range requires lots of bits, but only four decimal digits of precision

- Use a different encoding for the equivalent of "scientific notation": $F \times 2^{E}$

Very Large or Very Small Numbers: Floating-Point

- Use a different encoding for the equivalent of "scientific notation": $F \times 2^E$
- Need to represent F (*fraction*), E (*exponent*), and sign
- IEEE 754 Floating-Point Standard (32-bits):

Floating Point Representation

- The binary number is written in scientific format as $F \times 2^{E}$
- F is normalized to be between 1 and 2

  $$1 \leq F < 2$$

- Because F always has the format 1.xxxx, we may avoid storing 1, and assume its existence.
- 127 is added to E before storing it. Therefore, negative exponents will be between 1 and 126.

Floating Point Representation

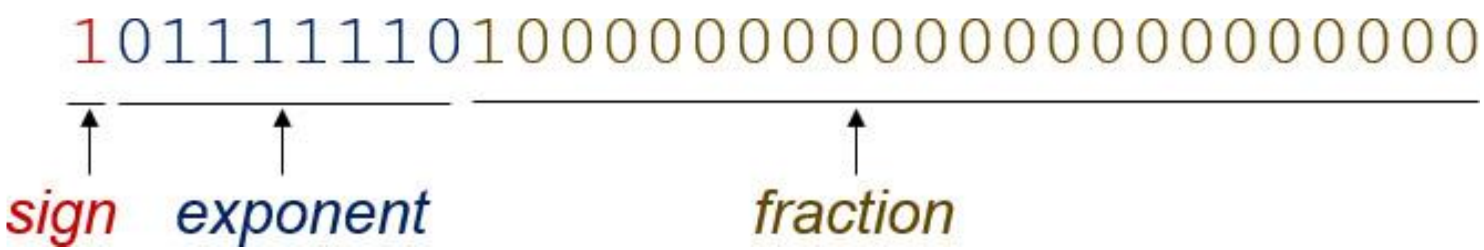
- The sign of the number is given by the left-most bit.
- The smallest valid value of E is 1.
- If E is zero, the existence of 1 before floating point is not assumed

$$N = (-1)^S \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

$$N = (-1)^S \times 0.\text{fraction} \times 2^{-126},\ \text{exponent} = 0$$

Floating-point Example

› Single-precision IEEE floating point number:



- Sign is 1 – number is negative
- Exponent field is 01111110 = 126 (decimal)
- Fraction is 0.100000000000… = 0.5 (decimal)

$$\text{Value} = -1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75.$$

Converting from Decimal to Floating-Point Binary

- **How do we represent $-6\,{}^{5}\!/_{8}$ as a floating-point binary number?**
- Ignoring sign, represent as a binary fraction: 0110.101
- Next, normalize the number, by moving the binary point to the right of the most significant bit:

$$0110.101 = 1.10101 \times 2^{2}$$

- Exponent field = 127 + 2 = 129 = 10000001
- Fraction field = everything to the right of the binary point: 101010000….
- Sign field is 1 (because the number is negative).
- Answer:
- 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Special Values

Infinity

- If exponent field is 11111111, the number represents **infinity**
- Can be positive or negative (per sign bit)

Subnormal

- If exponent field is 00000000, the number is not normalized. This means that the implicit 1 is not present before the binary point. The exponent is −126
- $N = (-1)^S \times 0.fraction \times 2^{-126}$.
- Extends the range into very small numbers
- Example: 00000000000001000000000000000000000

$$0.00001_{two} \times 2^{-126} = 2^{-131}$$

Q: *smallest number?*

Special Values

- The IEEE floating point number representation that strictly enforces the assumption of a leading one bit cannot store the value zero.
- To handle **zero**, the IEEE standard makes an exception:
  - when all bits are zero, the implicit assumption is ignored, and the stored value is taken to be zero.

Example:

0 00000000 00000000000000000000000

Binary Coded Decimal Representation

- Digital computers employ the binary representations for integers and floating point numbers.
- However, not every decimal fraction can be represented using binary floating point numbers.
- Example:
  - Convert 0.1 (10) to binary => 0.000110011001100110011....
  - 0.1 x 2 = 0.2
  - 0.2 x 2 = 0.4
  - 0.4 x 2 = 0.8
  - 0.8 x 2 = 1.6
  - 0.6 x 2 = 1.2
  - 0.2 x 2 =0.4

Binary Coded Decimal Representation

- The use of binary fractions has some unintended consequences, and their use does not suffice for all computations.
- For example, consider a bank account that stores Euros and cents.
- Cents are usually represented as hundredths of Euros.
- 10.83 denotes 10 Euros and 83 cents.
- If binary floating point arithmetic is used for bank accounts, individual cents are rounded, making the totals inaccurate.

BCD Codes

› BCD systems represent digits in binary where four bits are used to represent each digit.

Decimal Symbols	BCD Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Character Sets

- Bits have no intrinsic meaning; the hardware or software must determine what each bit represents.
- More than one interpretation can be used; a group of bits can be created with one interpretation and later used with another.
- Example:
  - character data has both a numeric and symbolic interpretation.
- Each computer system defines a character set to be a set of symbols that the computer and I/O devices agree to use.

Character Sets

- A typical character set contains uppercase and lowercase letters, digits, and punctuation marks.
- In the 1960s, IBM Corporation chose the Extended Binary Coded Decimal Interchange Code (EBCDIC) representation as the character set used on IBM computers.
- The American National Standards Institute (ANSI) defined a character representation known as the American Standard Code for Information Interchange (ASCII).
- The goal was to make peripheral devices compatible.

ASCII Codes

Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Summary

- Digital computers use binary number to represent data.
- Arithmetic operations on binary data can be performed in sign-magnitude, 1’s complement, or 2’s complement forms.
- Floating point representation of binary numbers use sign-exponent-fraction format.
- Floating point representation includes a discrete subset of real numbers.
- Some decimal numbers cannot be represented using a limited number of binary digits.

Questions?