



## Hexadecimal representation

- Binary hard to read for humans
  - Especially 16 bits, 32 bits, 64 bits
  - 1010101001010101
- Base 16 representation or hex representation
- Symbols = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
- Value =  $\sum i \times 16^i$
- First 10 (0 through 9) symbols are same as decimal
- A=10,B=11,C=12, D=13, E=14, F=15

5

## Decimal to binary

```
main(){
int N,i=0; int Maxbits=32; char bitarray[Maxbits]
while (N > 0){
/* MSB is bit 0, LSB is bit 31 */
bitarray[Maxbits-1-i]=N%2;i++;
N=N/2;
}
```

- Example: Convert 10 in decimal to binary
  - Above code works for any base
- For hex, replace N%2 by N%16 and replace reminders > 9 by A, B, C, D, E and F
- Example: Convert 240 to HEX

6

## Binary to decimal

- Convert 110110 to decimal
- Value =  $i \times 2^i$
- Value =  $2^5 2^4 2^2 2^1$ 
  - = 32+ 16 + 4 + 2
  - = 54
  - 
  -

7

## Converting Hex to binary

- Each digit in Hex can be represented by 4 bit binary (base 16) or nibble
- Convert 2A8C to binary
- 0010 1010 0100 1100

8

## Convert Binary to hex

- Group binary bits into groups of four
- Replace each nibble by a hex digit
- Example 1011011110011100
- 1011011110011100
- B79C or 0x B79C
- In C, numeric constants starting with 0x are interpreted as being in hexadecimal

9

## examples

- 0x8F7A93 to binary
- 1011011110011100 to hex
- 0xC4E5 to decimal

10

## Decimal and binary fractions

- In decimal, digits to the right of radix point have value  $1/10^i$  for each digit in the  $i^{\text{th}}$  place
- 0.25 is  $2/10 + 5/100$
- Similarly, in binary, digits to the right of radix point have value  $1/2^i$  for each  $i^{\text{th}}$  place
  - Fractions same except the base is different
- 8.625 is 1000.101

11

## Decimal to binary example

- 0.625 to binary
- ANS: 0.101
- $0.625 * 2 = 1.25$
- output 1
- $0.25 * 2 = 0.5$
- output 0
- $0.5 * 2 = 1$
- output 1
- Exit

```
Algorithm
/* precondition: 0 < number < 1)
number=decimalfraction
while (number >0)
{ number=number*2 /*shift
left */
if (number >=1) {Output
result=1; number=number-1}
else Output result=0
}
```

12

## Decimal to binary fractions

Decimal	Binary
0.5	0.1
0.25	0.01
0.625	0.101
0.75	0.11

13

## Data sizes

- All Information is represented in binary form but require different sizes
- Characters in 1 byte, integers 2 to 4 bytes, real numbers 4 to 8 bytes

C declaration	32-bit machine	64-bit machine
char	1	1
Short int	2	2
int	4	4
pointer	4	8
float	4	4
double	8	8

14

## Big endian vs little endian

- See a binary representation of a number in memory (multiple bytes)
- How to determine value
- Most Significant byte first ... big endian
- Least significant byte first ... little endian
- Depends on the type of machine
- Why we need to know:
  - If we are to interpret bytes and determine value
- One computer (big endian) sending data to another computer (small endian)
  - Need to convert into standard form before transmitting

15

## Representing integers

- Signed integers
  - Unsigned or natural numbers
  - Directly represent as binary
  - What about negative numbers?
- Signed Magnitude
- MSB is sign bit
 

S	Magnitude
---	-----------

  - 4 is 0100 -4 1100
  - 3 is 0011 -3 1011
  - 2 is 0010 -2 1010
  - 1 is 0001 -1 1001
  - 0 is 0000 ..what is 1000?
- Problems: two zeros +ve 0 and -ve 0
- normal bit-wise addition does not work... -1 + 4

16

## Representing Integers

- Divide the binary space into two halves
- One with leading 0s are +ve
- One with leading 1s are -ve
- This is called two's complement representation
- Two bit binary numbers
- 00 01 10 11
- 0 1 -2 -1
- Three bit

000	001	010	011	100	101	110	111
0	1	2	3	-4	-3	-2	-1

17

## 2s complement advantages

- Adding two integers works as normal arithmetic
- Normal arithmetic works for 2's complement (ignore carry)
- Only 1 zero
- $X + (-X) = 0$
- Used in almost all computers

18

## Negate a 2s complement

- invert the bits and add 1
- $X = 001 \bar{X} 110 \rightarrow 111$
- All 1s is -1, so add 1 to make it zero
- So, to get  $-X$ , invert all bits and add 1

19


## Finding 2s complement

- Take the binary representation and invert all bits
  - Step 1: 0 to 1 and 1 to 0
  - Step 2: Then add 1
- Example: 00101 (5)
- (-5) is .....
- Find 2s complement of 9

20

## Finding 2s complement

- Copy all bits from right to left until first 1
- Then flip all bits

• 0100 1  
  
 • 1011 1

- Example find 2s complement of 0100, 0110

• 0100      0110  
 .....  
 .....  
 .....  
 .....

21

## Decimal value of 2s complement

- Most significant bit has -ve value
- $X_{n-1}, X_{n-2}, \dots, X_1, X_0$
- Value of MSB is  $-2^{(n-1)}$
- Rest is same as +ve binary number
- 110011 ... what is the decimal value?

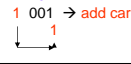
22

## Decimal value of 2's complement

- If leading bit or MSB is 1, take 2's complement to get +ve number
- Determine decimal value of number
- And add -ve sign to it
- If leading bit or MSB is 0, compute as normal
- Example  $1100110_{TWO}$

23

## 1s complement

- Alternative representation
- -ve numbers represented by the complement
- Max numbers that can be represented is halved
- When adding +ve and -ve numbers carry need to be added to the result
- $-3 + (-2)$
- 100
- 101
- 1 001 → add carry
- 
- 010 → (-)5

000	001	010	011	100	101	110	111
0	1	2	3	-3	-2	-1	-0

24

## ASCII

- Characters are also stored as bits
- 1 byte is 8 bits but MSB is used for error detection
- ASCII represents typical keys on a keyboard
- 7 bits is 128 different possibilities
- 7 bit ASCII included printable and non-printable characters
- Non-printable for controlling printer such as LF or linefeed

25

## ASCII table

	0	1	2	3	4	5	6	7
0	NUL	DL	space	0	@	P	-	p
1	SOH	DC1	SON	!	1	A	Q	a
2	STX	DC2	*	?	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	^	7	G	W	g	w
8	BS	CAN	{	8	H	X	h	x
9	HT	EM	}	9	I	Y	i	y
A	LF	SUB	*	.	J	Z	j	z
B	VT	ESC	+	:	K	[	k	{
C	FF	FS	.	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

26

## unicode

- Extended to 16 bit characters
- Represent other characters (Japanese)
- Java supports this format
- In Unicode, the first 128 characters are ASCII

27

## Floating point

- With integers range is limited
- short int 2 bytes  $2^{16}$
- unsigned short int
  - Unsigned: Range is natural number 0 to 65536
  - Signed: 2s complement is -32768 to 32767
- for int (4 bytes) -2147483648 to 2147483647
- Can also be represented as magnitude and exponent
- $2.147483647 \times 10^9$

28



## Exponent bias

- The binary exponent is represented by adding a bias of 127
- Saves an extra bit for sign bit
- Note: it is not 2s complement
- $2^3$  is  $2^{130}$  which is  $2^{10000010}$
- $2^0$  is  $2^{127}$  which is  $2^{01111111}$
- $2^{-3}$  is  $2^{124}$  which is  $2^{01111100}$

33

## Normalization

- The exponent value is adjusted so that the mantissa has a 1 before the radix point
- 0.25 is  $0.01 \times 2^0$  adjusted to  $1 \times 2^{-2}$
- 4.625 is  $100.101 \times 2^0$  adjusted to  $1.00101 \times 2^2$
- 64 is  $1000000 \times 2^0$  adjusted  $1 \times 2^6$
- In this way, the mantissa always has 1 digit with value 1
- This can be omitted to save a bit!!!

34

## Decimal to floating point

- 3.625
- In binary
- $101.101 \rightarrow 1.01101 \times 2^2$
- Exponent field has value 2
  - add 127 to get 129
- Exponent is 10000001
- Mantissa is 01101
- Sign bit is 0
- **0** 011010000000000000000000 **10000001**

35

## One more example

- Convert 12.375 to floating point representation
- Binary is 1100.011

36

## Floating point to decimal

1	1000010	010010000000000000000000
---	---------	--------------------------

What is the decimal value of the above?  
Exponent is (1000010) - 127  
Mantissa is 1.01001  
Sign bit is -1

s	f	e
---	---	---

$$(1 - 2s) * (1 + f) * 2^{e - bias}$$

37

## Special cases

- 0 00000000 000000000000000000000000
- All zeros 0
- 1 00000000 000000000000000000000000
- All zeros with sign bit is -0
- 0 11111111 000000000000000000000000
- Infinity
- 1 11111111 000000000000000000000000
- -Infinity

38

## Extended precision

- 80 bits used to represent a real number
- 1 sign bit, 15 bit exponent, 64 bit mantissa
- 20 decimal digits of accuracy
- $10^{-4932}$  to  $10^{4932}$
- Not supported in C

39