Base Representation and memory

Base Reprsentation

- Binary ---- Base 2
- Characters: 1, 0

- Place Values 2^4 2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} 2^{-3}
- 16 8 4 2 1 . 0.5 0.25 0.125

Conversion of Binary to base 10

<u>Use place value notation</u>

e.g. convert 101010_2 to base 10

- Match the place value with the corresponding character

1	0	1	0	1	0
32	16	8	4	2	1

Conversion of Binary to base 10

- Multiply the place value with the its corresponding binary character

1	0	1	0	1	0
*	*	*	*	*	*
32	16	8	4	2	1
32	0	8	0	2	0

- Sum the result

32 + 0 + 8 + 0 + 2 + 0 = 42

Conversion of base 10 to base 2

- Take the base 10 number divide by the base and collect the remainder, repeat the process but taking the resultant term of the division each time.
- Terminate when zero is reached
- The binary result is the sequence of the remainders generated, which are read from the bottom to the top.

Conversion of base 10 to base 2

- e.g. Convert 456_{10} to base 2

$456 \div 2$	$\longrightarrow$	228	r 0
$228 \div 2$	$\longrightarrow$	114	r 0
$114 \div 2$	$\longrightarrow$	57	r 0
$57 \div 2$	$\longrightarrow$	28	r 1
$28 \div 2$	$\longrightarrow$	14	r 0
$14 \div 2$	$\longrightarrow$	7	r 0
$7 \div 2$	$\longrightarrow$	3	r 1
$3 \div 2$	$\longrightarrow$	1	r 1
$1 \div 2$	$\longrightarrow$	0	r 1

Read binary number from the bottom to the top

111001000_2

Converting decimal fractions to base 2

- When converting decimal fractions one takes the base 10 fraction and then multiply it by the base (2)
- One then splits the result into the fraction part and the whole number part
- whole number part is then is stored and the process is repeated with the resultant fraction part.
- The process is terminated when the fraction part of the product is 0
- One then reads the sequence of whole number parts generated from the top to the bottom

e.g. convert 0.5625_{10} to base 2

Fraction	Multiply by base	Product	Fraction part of product	Whole number part of product
0.5625	X 2	1.125	0.125	1
0.125	X 2	0.25	0.25	0
0.25	X 2	0.5	0.5	0
0.5	X 2	1.0	0	1

Binary no.
read from
top to bottom

Note that not all decimal fractions can be converted into binary e.g.

1/5 = (0.2).

Hexadecimal ---- base16

- Hexadecimal Characters—including their decimal na binary equivalent values

Dec	Binary	Hex	Dec	Binary	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Hexadecimal ---- base16 Place values

16^4	16^3	16^2	16^1	16^0	16^{-1}	16^{-2}	16^{-3}
65536	4096	256	16	1	1/16	1/256	1/4096

To convert Base 16 to base 10

- Use place values, (same method as that of binary)
- e.g. convert 1AB8 into decimal
- 1) Multiply each character with the corresponding base value

1	**A**	**B**	**8**
16^3	16^2	16^1	16^0
(1×16^3)	(10×16^2)	(11×16^1)	(8×16^0)

- 2) Sum the result
- $(1 \times 16^3) + (10 \times 16^2) + (11 \times 16^1) + (8 \times 16^0) = 6840$

Hexadecimal and binary

- Hexadecimal is used to represent binary numbers as there is an easy conversion method,
- We have a 1:1 mapping of binary to hex and vice versa
- and so the values of the registers for example represented by the assembler are represented in hexadecimal.

Hexadecimal and binary

Base	**Hex/bin value**	**Hex/bin value**	**Hex/bin value**	**Hex/bin value**	**Hex/bin value**	**Hex/bin value**	**Hex/bin value**	**Hex/bin value**
Hex.	0	1	2	3	4	5	6	7
Bin.	0000	0001	0010	0011	0100	0101	0110	0111
Hex	8	9	A	B	C	D	E	F
Bin.	1000	1001	1010	1011	1100	1101	1110	1111

To covert hexadecimal into binary

- Simply swap the corresponding hexadecimal digit with that of the 4 bit binary representation.
- e.g. Convert $A127_{16}$ into binary
- A 1 2 7
- 1010 0001 0010 0111

- So $A127_{16}$ translates to 1010000100100111_2

To convert binary into hexadecimal

- Group the binary digits (4 bits at a time) starting from the right hand side of the binary number, moving to left
- Then swap each of the nibbles (4 bits) with its corresponding hexadecimal character
- e.g. Convert 10101010010001_2 into hexadecimal

- Step 1: 10 1010 1001 0001
- Step 2: 2 A 9 1

- $10101010010001_2 = 2A91_{16}$

Representing Negative Numbers on a computer

- Method 2: Using 2s complement notation

  Use the MSB but with a negative weighting

For an 8 bit representation:

	MSB							LSB
	-128	64	32	16	8	4	2	1
+44	0	0	1	0	1	1	0	0
-44	1	1	0	1	0	1	0	0

converting a binary number into a twos complement number

- Step1: Take Binary number and flip the bits (0 -> 1 and 1 -> 0)
- Generate the value of -32 in 2's complement notation
- +32 using 8 bits : 00100000_2
- Flip bits : 11011111_2
- Step 2: Add 1

```
 11011111
+       1
 11100000   == -32
```

IEEE Floating Point Number representation

- Standard Form
- We can write out a number in two parts

1. The Fraction part
2. The magnitude or exponent part

- $e.g$ 356 can be re-written as 3.56 x 10^2

Fraction part

Exponent part

IEEE Floating Point Number representation

- $x = (-1)^{S} \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-\text{Bias})}$
- S: sign bit (0 = non-negative, 1 = negative)
- **Fraction** part -**Normalized** significand:
- 1.0 ≤ |significand| < 2.0
- We divide by a number so the Fraction ranges from 1.0 to up to but not including 2.0
- no need to represent the value of 1 explicitly (hidden bit) – that's why we add 1 to the fraction

IEEE Floating Point Number representation

- $x = (-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-\text{Bias})}$
- **Exponent**: excess representation: actual exponent + Bias
- Ensures exponent is unsigned to ensure this we use a Bias($2^{(N-1)} - 1$)

    - Single precision: Bias = 127;
    - Double precision: Bias = 1023
    - Quadruple precision: Bias = 16383

IEEE Floating Point Number representation

- Generate – -45.375 in IEEE format using 32 bit representation(single precision)
- Step 1 – the sign : The s bit will be set to 1 as our value is negative
- Step 2 – Generate the EXPONENT
- convert the absolute value of -45.375 to binary == 101101.011
- Shift the decimal point so that we obtain 1.something
- 1.01101011 (count the number of shifts) in our case we have + 5
- Note if we shift from left to right the the sign will be -ve

IEEE Floating Point Number representation example cont. -45.375

- EXPONENT
- Add the number of shifts (in our case +5) to the BIAS (for 32 bit representation it is 127)
- 5 + 127 = 132
- Convert the result into binary 132 = 10000100; this should be 8 bits if less add preceding 0s to make it upto 8 bits
- This is the exponent part of the number

IEEE Floating Point Number representation example cont. -45.375

- Step 3 – Generate the FRACTION or MANTESSA
- Take the binary number 1.01101011
- Ignore the whole number bit we have 01101011
- This is the fraction part make it upto 23 bits (length of fraction part by adding trailing 0s giving:
- 01101011000000000000000
- Putting it all together
- SignExponentFraction
- 11000010001101011000000000000000
-