CSO

HW1

Q1 Conversion Practice

1. (a) $(28)_{10}$

2	28	
2	14	0
2	7	0
2	3	1
	1	1

$(28)_{10} = (11100)_2$

16	28	
	1	12

$(28)_{10} = (1C)_{16}$

12 is 'C' from the hexadecimal table.

(b) $(135)_{10}$

2	135	
2	67	1
2	33	1
2	16	1
2	8	0
2	4	0
2	2	0
	1	0

$(135)_{10} = (10000111)_2$

16	135	
16	8	7

$(135)_{10} = (87)_{16}$

(c) $(4096)_{10}$

2	4096	
2	2048	0
2	1024	0
2	512	0
2	256	0
2	128	0
2	64	0
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

$(4096)_{10} = (1000000000000)_2$

16	4096	
16	256	0
16	16	0
	1	0

$(4096)_{10} = (1000)_{16}$

2. (a) $(101010)_2$

$(101010)_2 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

$= 2^5 + 2^3 + 2^1 = 32 + 8 + 2 = (42)_{10}$

$\Rightarrow (101010)_2 = (42)_{10}$

$\overline{0010}\ \overline{1010} = (2A)_{16}$

$\Rightarrow (101010)_2 = (2A)_{16}$

(b) $(1101101)_2$

$(1101101)_2 = 1\times2^6 + 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$= 2^6 + 2^5 + 2^3 + 2^2 + 2^0$

$= 64 + 32 + 8 + 4 + 1$

$= (109)_{10}$

$\therefore (1101101)_2 = (109)_{10}$

$\overline{0110}\ \overline{1101} = \overline{6}\ \overline{D}$

$\therefore (1101101)_2 = (6D)_{16}$

(c) $(11110000)_2$

$(11110000)_2 = 1\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 0\times2^0$

$= 2^7 + 2^6 + 2^5 + 2^4$

$= 128 + 64 + 32 + 16 = (240)_{10}$

$(11110000)_2 = (240)_{10}$

$\overline{1111}\ \overline{0000} = (F0)_{16}$

$\Rightarrow (11110000)_2 = (F0)_{16}$

I use the self-written tables below for binary $\Leftrightarrow$ hexadecimal conversions in 2. & 3.
decimal $\Leftrightarrow$ hexadecimal (for letters)

Binary	HXD
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

HXD	Number
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

3. (a) $(2A)_{16}$

$\overline{2}\ \overline{A} = \overline{0010}\ \overline{1010}$

$\Rightarrow (2A)_{16} = (101010)_2$

$(2A)_{16} = 2 \times 16^1 + 10 \times 16^0$

$= 32 + 10 = (42)_{10}$

$\Rightarrow (2A)_{16} = (42)_{10}$

(b) $(B4)_{16}$

$\overline{B}\ \overline{4} = \overline{1011}\ \overline{0100}$

$(B4)_{16} = (10110100)_2$

$(B4)_{16} = 11 \times 16^1 + 4 \times 16^0$

$= 176 + 4 = (180)_{10}$

$\Rightarrow (B4)_{16} = (180)_{10}$

(c) $(1F8)_{16}$

$\overline{1}\ \overline{F}\ \overline{8} = \overline{0001}\ \overline{1111}\ \overline{1000}$

$\Rightarrow (1F8)_{16} = (111111000)_2$

$(1F8)_{16} = 1 \times 16^2 + 15 \times 16^1 + 8 \times 16^0$

$= 256 + 240 + 8 = (504)_{10}$

$\Rightarrow (1F8)_{16} = (504)_{10}$

Q2 Binary Addition and Subtraction

1. Binary Addition

(a) 1101 + 1011

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ \ } \\ 1101 \\ +\ 1011 \\ \hline 11000 \end{array}$$

∴ 1101 + 1011 = 1000 (with an overflow)

(b) 10010 + 11011

$$\begin{array}{r} {\scriptstyle 1\ \ \ \ \ } \\ 10010 \\ +\ 11011 \\ \hline 101101 \end{array}$$

∴ 10010 + 11011 = 01101 (with an overflow)

2. Binary Subtraction

(a) 10101 − 1101 =

$$\begin{array}{r} 10101 \\ +\ 10010 \\ \hline 100{\scriptstyle 1}1{\scriptstyle 1}1{\scriptstyle 1} \\ +\ 00001 \\ \hline 01000 \end{array}$$

∴ 10101 − 1101 = 1000

(b) 11100 − 1010 =

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ \ } \\ 11100 \\ +\ 10101 \\ \hline 1100{\scriptstyle 1}1 \\ +\ 00001 \\ \hline 10010 \end{array}$$

∴ 11100 − 1010 = 10010

I performed binary subtraction using two's complement (as the computer does) and have ignored any overflows.

Q3 Base 32

1. For base 32, we can use 0-9 as digits and A-V (representing numerical encoding of 10-31) as the Latin letters. In total, we have 32 different digit values to represent numbers in base-32.

2. $(90)_{32} = (9 \times 32^1 + 0 \times 32^0)_{10}$

$= (288)_{10}$

$= (100100000)_2 = (120)_{16}$

3. since $2^5 = 32$, I can think of a bit grouping algorithm like that used for binary to hexadecimal conversion. The algorithm:

1. Starting from the rightmost side of the binary number, I will group the digits into groups of 5.
2. If the leftmost group has less than 5 bit, then I will add zeroes to the beginning of that group.
3. Then, I'll use the following table to convert base-2 to base-32. (or, I can first convert each group to their decimal equivalent & assign each decimal value to its corresponding base-32-bit)

base-32/base-2 conversion table.

00000	0	01010	A/10	10100	K/20	11110	U/30
00001	1	01011	B/11	10101	L/21	11111	V/31
00010	2	01100	C/12	10110	M/22		
00011	3	01101	D/13	10111	N/23		
00100	4	01110	E/14	11000	O/24		
00101	5	01111	F/15	11001	P/25		
00110	6	10000	G/16	11010	Q/26		
00111	7	10001	H/17	11011	R/27		
01000	8	10010	I/18	11100	S/28		
01001	9	10011	J/19	11101	T/29		

11011 10000 01010 11011

→ Using my algorithm:

$(\overline{11011}\ \overline{10000}\ \overline{01010}\ \overline{11011})_2$

$= (RGAR)_{32}$

→ Using decimal middle-step conversion

$= 1 \times 2^{19} + 1 \times 2^{18} + 1 \times 2^{16} + 1 \times 2^{15} + 1 \times 2^{14} + 1 \times 2^8 + 1 \times 2^6 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = (901467)_{10}$

32	901467	
32	28170	27
32	880	10
32	27	16
32	0	27

$\therefore (901467)_{10} = (\overline{27}\ \overline{16}\ \overline{10}\ \overline{27})_{32}$

$= (RGAR)_{32}$

Hence, verified!

Numerical Encoding for base-32

10	A	21	L
11	B	22	M
12	C	23	N
13	D	24	O
14	E	25	P
15	F	26	Q
16	G	27	R
17	H	28	S
18	I	29	T
19	J	30	U
20	K	31	V

Q4 Floating Point Representation

1. $26 + 0.75$

2	26	
2	13	0
2	6	1
2	3	0
2	1	1
	0	1

$\Rightarrow (26)_{10} = (11010)_2$

$0.75 \times 2 = 1.50 \quad 1$
$0.5 \times 2 = 1.00 \quad 1$

$\Rightarrow (0.75)_{10} = (0.11)_2$

$(26.75)_{10} = (11010.11)_2$

$(11010.11)_2 = (1.101011)_2 \times 2^4$

Sign : 0
(1 bit)

Exponent : $(4 + 127)_{10} = (131)_{10}$
(8 bit)

2	131	
2	65	1
2	32	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

$= (10000011)_2$

Mantissa : .101011 00000000000000000
(23 bit) = 101 0110 0000 0000 0000 0000

$\therefore (26.75)_{10} = 0 - 10000011 - 10101100000000000000000$

2. (a) 0 - 10000000 - 10010000000000000000000

Sign Bit | Exponent (8 bit) | Mantissa (23 bit)

Sign Bit = 0 $\rightarrow$ +ve

Exponent = $(1000\ 0000)_2$
$= 1 \times 2^7 = (128)_{10} \Rightarrow (128 - 127) = 1$

Mantissa = $(10010000000000000000000)_2$
$= 1 \times 2^{-1} + 1 \times 2^{-4}$
$= 0.5 + 0.0625$
$= (0.5625)_{10}$

Overall : $(-1)^{sign} \times (1 + Mantissa) \times 2^{exponent}$
$= (-1)^0 \times (1 + 0.5625) \times 2^1$
$= 1.5625 \times 2 = (3.125)_{10}$

$\therefore (3.125)_{10}$

(b) 1 - 10000010 - 01010111000000000000000

Sign Bit	Exponent (8 bits)	Mantissa (23 bits)

Sign: 1 = -ve

Exponent : $(10000010)_2$

$= 1\times 2^7 + 1\times 2^1 = 128 + 2$

$= (130)_{10} \Rightarrow 130 - 127 = 3$

Mantissa : $(01010111000000000000000)_2$

$\Rightarrow 1\times 2^{-2} + 1\times 2^{-4} + 1\times 2^{-6} + 1\times 2^{-7} + 1\times 2^{-8}$

$= 0.25 + 0.0625 + 0.015625 + 0.0078125 + 0.00390625$

$= (0.33984375)_{10}$

Overall : $(-1)^{sign} \times (1 + \text{Mantissa}) \times 2^{exponent}$

$= (-1)^1 \times (1 + 0.33984375) \times 2^3$

$= -1.33984375 \times 8$

$= (-10.71875)_{10}$

$\therefore (-10.71875)_{10}$

Q5. Floating Point Precision

$(0.1)_{10} = (0 + 0.1)_{10}$

$(0)_{10} = (0)_2$

Sign: 0 $\Rightarrow$ +ve (1 bit)

for $(0.1)_{10}$,

1) 0.1 x 2 = 0.2 0
2) 0.2 x 2 = 0.4 0
3) 0.4 x 2 = 0.8 0
4) 0.8 x 2 = 1.6 1
5) 0.6 x 2 = 1.2 1
6) 0.2 x 2 = 0.4 0
7) 0.4 x 2 = 0.8 0
8) 0.8 x 2 = 1.6 1
9) 0.6 x 2 = 1.2 1
10) 0.2 x 2 = 0.4 0
11) 0.4 x 2 = 0.8 0
12) 0.8 x 2 = 1.6 1
13) 0.6 x 2 = 1.2 1
14) 0.2 x 2 = 0.4 0
15) 0.4 x 2 = 0.8 0
16) 0.8 x 2 = 1.6 1
17) 0.6 x 2 = 1.2 1
18) 0.2 x 2 = 0.4 0
19) 0.4 x 2 = 0.8 0
20) 0.8 x 2 = 1.6 1
21) 0.6 x 2 = 1.2 1
22) 0.2 x 2 = 0.4 0
23) 0.4 x 2 = 0.8 0
24) 0.8 x 2 = 1.6 1
25) 0.6 x 2 = 1.2 1
26) 0.2 x 2 = 0.4 0

27) 0.4 x 2 = 0.8 0

$(0.1)_{10} \simeq (0.0001\ 1001\ 1001\ 1001\ 1001\ 1001\ 100)_2$

$\simeq (1.10011001100110011001100)_2 \times 2^{-4}$

Exponent = $(-4 + 127)_{10} = (123)_{10}$

2	123	
2	61	1
2	30	1
2	15	0
2	7	1
2	3	1
	1	1

$(123)_{10} = (01111011)_2$ (8-bits)

Mantissa : 100 1100 1100 1100 1100 1100 (23 bits)

$\therefore (0.1)_{10} = (0-01111011-10011001100110011001100)_2$

To estimate the size of the error, I will be using an online calculator to give an approx result for the IEEE 32 bit floating single number. (www.binary-system.base-conversion.ru)

Converting the above RHS back to decimal gives

$= 0.099999994039535522460937S$

Error Size $= 0.1 - 0.099999994039535522460937S$

$= 5.960464483090 \times 10^{-9}$

Error % $= 5.960464483090 \times 10^{-6}$ %