Architecture des ordinateurs

TD 3 : Représentation des réels et des caractères

Exercice 1 Représentation d'une partie fractionnaire

1. Coder sur 8 bits les parties fractionnaires suivantes :
   - $(0.578125)_{10} = (0.10010100)_2$

     - $0.578125 \times 2 = 1.15625 \rightarrow 1$
     - $0.15625 \times 2 = 0.3125 \rightarrow 0$
     - $0.3125 \times 2 = 0.625 \rightarrow 0$
     - $0.625 \times 2 = 1.25 \rightarrow 1$
     - $0.25 \times 2 = 0.5 \rightarrow 0$
     - $0.5 \times 2 = 1.0 \rightarrow 1$
     - $0.0 \times 2 = 0.0 \rightarrow 0$

   - $(0.85)_{10} = (0.11011001)_2$

     - $0.85 \times 2 = 1.7 \rightarrow 1$
     - $0.7 \times 2 = 1.4 \rightarrow 1$
     - $0.4 \times 2 = 0.8 \rightarrow 0$
     - $0.8 \times 2 = 1.6 \rightarrow 1$
     - $0.6 \times 2 = 1.2 \rightarrow 1$
     - $0.2 \times 2 = 0.4 \rightarrow 0$
     - $0.4 \times 2 = 0.8 \rightarrow 0$
     - $0.8 \times 2 = 1.6 \rightarrow 1$
     - ça boucle ...

2. Décoder :
   - $(0.10110000)_2 = (2^{-1} + 2^{-3} + 2^{-4})_{10} = (0.5 + 0.125 + 0.0625)_{10} = (0,6875)_{10}$
   - $(0.11011001)_2 = (2^{-1} + 2^{-2} + 2^{-4} + 2^{-5} + 2^{-8})_{10} = (0.5 + 0.25 + 0.0625 + 0.03125 + 0.00390625)_{10} = (0,84765625)_{10}$
     Remarque : dans la question précédente on avait codé $(0.85)_{10}$ par $(0.11011001)_2$ ...

Exercice 2 Représentation des réels en norme IEEE 754

Rappeler l'étendue des valeurs des nombres normalisés en simple et double précision.

Exercice 3 Codage en IEEE 754 Coder les réels suivants (représentés en base 10) en simple précision :

- 40
  - **signe** 1 bit : $+ \rightarrow 0$
  - représentation binaire : 101000 (valeur absolue)
  - normalisation : 1.01000×2^5
  - exposant réel (exp effectif) : 5
  - **exposant décalé (exp codé)** 8 bits : $(5 + 127)_{10} = (132)_{10} = (10000100)_2$
  - **f** 23 bits : 01000000000000000000000
  - $(01000010001000000000000000000000)_2 = (42200000)_{16}$

– -0.078125
  – **signe** 1 bit : - $\rightarrow$ 1
  – représentation binaire : 0.0001010 (valeur absolue)
    – $0.078125 \times 2 = 0.15625 \rightarrow 0$
    – $0.15625 \times 2 = 0.3125 \rightarrow 0$
    – $0.3125 \times 2 = 0.625 \rightarrow 0$
    – $0.625 \times 2 = 1.25 \rightarrow 1$
    – $0.25 \times 2 = 0.5 \rightarrow 0$
    – $0.5 \times 2 = 1.0 \rightarrow 1$
    – $0.0 \times 2 = 0.0 \rightarrow 0$
    – pfff ... c'est le même calcul que dans le premier exercice ...
  – normalisation : 1.010×2^{-4}
  – exposant réel : -4
  – **exposant décalé** 8 bits : $(-4 + 127)_{10} = (123)_{10} = (01111011)_2$
  – **f** 23 bits : 01000000000000000000000 (la même que pour 40 !)
  – $(10111101101000000000000000000000)_2 = (BBA00000)_{16}$

– 13.625
  – **signe** 1 bit : + $\rightarrow$ 0
  – représentation binaire : 1101.101 (valeur absolue)
  – normalisation : 1.101101×2^3
  – exposant réel : 3
  – **exposant décalé** 8 bits : $(3 + 127)_{10} = (130)_{10} = (10000010)_2$
  – **f** 23 bits : 10110100000000000000000
  – $(01000001010110100000000000000000)_2 = (415A0000)_{16}$

– -87.375
  – **signe** 1 bit : - $\rightarrow$ 1
  – représentation binaire : 1010111.011 (valeur absolue)
  – normalisation : 1.010111011×2^6
  – exposant réel : 6
  – **exposant décalé** 8 bits : $(6 + 127)_{10} = (133)_{10} = (10000101)_2$
  – **f** 23 bits : 01011101100000000000000
  – $(11000010101011101100000000000000)_2 = (C2AEC000)_{16}$

– 0 : définition du cours
  – **signe** 1 bit : + ou - $\rightarrow$ 0 ou 1
  – **exposant décalé** 8 bits : $(0)_{10} = (00000000)_2$
  – **f** 23 bits : 00000000000000000000000
  – $(10000000000000000000000000000000)_2 = (80000000)_{16}$
  – $(00000000000000000000000000000000)_2 = (00000000)_{16}$

– NaN : définition du cours

– **signe** 1 bit : + ou - $\rightarrow$ 0 ou 1
– **exposant décalé** 8 bits : $(127)_{10} = (11111111)_2$
– **f** 23 bits : au moins un bit à 1 mais quelconque
– $(111111111xxxxxxxxxxxxxxxxxxxxxx)_2 = (FF\{y \geq 8\}xxxxx)_{16}$
– $(011111111xxxxxxxxxxxxxxxxxxxxxx)_2 = (7F\{y \geq 8\}xxxxx)_{16}$

$+\infty$: définition du cours
– **signe** 1 bit : + $\rightarrow$ 0
– **exposant décalé** 8 bits : $(127)_{10} = (11111111)_2$
– **f** 23 bits : 00000000000000000000000
– $(01111111100000000000000000000000)_2 = (7F800000)_{16}$

Exercice 4 Décodage de réels exprimés en norme IEEE 754 Décoder les réels suivants, donnés en simple précision :

Remarques :
– pour décoder le signe et le dernier bit de l'exposant décalé (exposant codé) il suffit de constater que seuls les symboles ≥ 8 en hexadécimal représentent une séquence de 4 bits commançant par 0 ...
– on peut décoder en décimal directement 1.f (ou 0.f) et multiplier par $2^{expeffectif}$ après.

– $41FE8000_{16}$
  – **signe** : $4 < 8$ donc signe +
  – exposant décalé : 100 0001 1 = $(131)_{10}$
  – **exposant réel** : $(131 - 127)_{10} = (4)_{10}$
  – f : 111 1110 1000 0000 0000 0000
  – 1.f : 1.111 1110 1
  – **1.f dénormalisé** : 11111.1101
  – $(2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4})_{10} = 2^5 - 1 + 2^{-1} + 2^{-2} + 2^{-4} = (31.8125)_{10}$

– $3EA80000_{16}$
  – **signe** : $3 < 8$ donc signe +
  – exposant décalé : 011 1110 1 = $(125)_{10}$
  – **exposant réel** : $(125 - 127)_{10} = (-2)_{10}$
  – f : 010 1000 1000 0000 0000 0000
  – 1.f : 1.0101
  – **1.f dénormalisé** : 0.010101
  – $(2^{-2} + 2^{-4} + 2^{-6} = (0.328125)_{10}$

– $C5E00000_{16}$
  – **signe** : $C \geq 8$ donc signe -
  – exposant décalé : 100 0101 1 = $(139)_{10}$
  – **exposant réel** : $(139 - 127)_{10} = (12)_{10}$
  – f : 110 0000 0000 0000 0000 0000
  – 1.f : 1.11
  – **1.f dénormalisé** : 1 1100 0000 0000
  – $(2^{10} + 2^{11} + 2^{12} = (7168)_{10}$

– 00380000_{16}

- **signe** : $0 < 8$ donc signe +
- exposant décalé : 000 0000 0 = $(0)_{10}$

⇒ soit c'est zéro, soit c'est un nombre dénormalisé, comme la partie f n'est pas nulle c'est un nombre dénormalisé.

- **expostant réel** : $(-127)_{10}$
- f : 011 1000 0000 0000 0000 0000
- **0.f** : 0.0111
- $(2^{-2} + 2^{-3} + 2^{-4}) \times 2^{-127} = (0.4375 \times 2^{-127})_{10}$

Exercice 5 Additions Effectuer les additions suivantes :

Avant de procéder à l'addition on doit mettre le plus petit des deux nombres au même exposant que le plus grand, et vérifier les signes.

- $C0880000_{16}$ + 41680000_{16}
  - un positif et un négatif : on devra faire une soustraction
  - exposant décalé de C0880000 = 100 0000 1 = $(65)_{10}$
  - exposant décalé de 41680000 = 100 0001 0 = $(66)_{10}$
  - le plus petit (valeur absolue) est donc : C0880000, on doit ajouter 1 = 10000010-10000001 à son exposant
  - l'opération à effectuer est donc : 1.1101 - 0.10001 = 1.01001, le résultat est positif car 1.1101 > 0.10001, et l'exposant (décalé) est 100 0001 0, ce qui donne 0100 0001 0010 0100 0000 0000 0000 0000 en binaire, soit 41 24 00 00 en hexadécimal.
- $815A6000_{16}$ + $015B1000_{16}$
  - un positif et un négatif : on devra faire une soustraction
  - exposant décalé de 815A6000 = 000 0001 0 = $(2)_{10}$
  - exposant décalé de 015B1000 = 000 0001 0 = $(2)_{10}$

  ⇒ les deux exposants sont égaux donc on peut faire l'opération directement sur les 1.f
  - 1.f pour 815A6000 = 1.1011010011
  - 1.f pour 015B1000 = 1.1011011001

  ⇒ le plus grand (valeur absolue) est donc 015B1000, on fait l'opération 1.1011011001-1.1011010011 et le résultat final sera positif.
  - 1.1011011001-1.1011010011 = 0.0000 0001 1, on doit renormaliser, il faut multiplier ce résultat par 2^{-8} pour obtenir 1.1, ce qui donne un nouvel exposant (décalé) de 2-8 = -6 < 0 (en exposant réel ca donne -6-127 = -133) donc le résultat est trop petit pour être codé en simple précision, le mieux qu'on puisse faire est de l'arrondir à zéro ... ou d'utiliser la notation dénormalisée ! Pour cela on doit exprimer le nombre avec un exposant égal à -127, il faut le multiplier par 2^6 et en même temps décaler la virgule de 6 positions vers la gauche. On obtient : 0.0000 0000 0000 011 avec comme exposant effectif à -127, soit un exposant codé à 0 ce qui correspond au codage dénormalisé. Le résultat est donc :
    - signe : positif = 0
    - exposant : 000 0000 0 (dénormalisé)
    - f : 000 0000 0000 0011 0000 0000

    ⇒ 0000 0000 0000 0000 0000 0011 0000 0000 = $(00\ 00\ 03\ 00)_{16}$

ASCII								
$(.)_{10}$	$(.)_{16}$	char	$(.)_{10}$	$(.)_{16}$	char	$(.)_{10}$	$(.)_{16}$	char
32	0x20	<SPACE>	64	0x40	@	96	0x60	‘
33	0x21	!	65	0x41	A	97	0x61	a
34	0x22	¨	66	0x42	B	98	0x62	b
35	0x23	#	67	0x43	C	99	0x63	c
36	0x24	$	68	0x44	D	100	0x64	d
37	0x25	%	69	0x45	E	101	0x65	e
38	0x26	&	70	0x46	F	102	0x66	f
39	0x27	’	71	0x47	G	103	0x67	g
40	0x28	(	72	0x48	H	104	0x68	h
41	0x29	)	73	0x49	I	105	0x69	i
42	0x2A	*	74	0x4A	J	106	0x6A	j
43	0x2B	+	75	0x4B	K	107	0x6B	k
44	0x2C	,	76	0x4C	L	108	0x6C	l
45	0x2D	-	77	0x4D	M	109	0x6D	m
46	0x2E	.	78	0x4E	N	110	0x6E	n
47	0x2F	/	79	0x4F	O	111	0x6F	o
48	0x30	0	80	0x50	P	112	0x70	p
49	0x31	1	81	0x51	Q	113	0x71	q
50	0x32	2	82	0x52	R	114	0x72	r
51	0x33	3	83	0x53	S	115	0x73	s
52	0x34	4	84	0x54	T	116	0x74	t
53	0x35	5	85	0x55	U	117	0x75	u
54	0x36	6	86	0x56	V	118	0x76	v
55	0x37	7	87	0x57	W	119	0x77	w
56	0x38	8	88	0x58	X	120	0x78	x
57	0x39	9	89	0x59	Y	121	0x79	y
58	0x3A	:	90	0x5A	Z	122	0x7A	z
59	0x3B	;	91	0x5B	[	123	0x7B	}
60	0x3C	<	92	0x5C	\	124	0x7C	|
61	0x3D	=	93	0x5D	]	125	0x7D	{
62	0x3E	>	94	0x5E	ˆ	126	0x7E	∼
63	0x3F	?	95	0x5F	_	127	0x7F	<DEL>

Exercice 6 Représentation des caractères

1. En utilisant le code ASCII, écrire votre nom et votre prénom sans oublier de mettre les initiales en majuscule.
2. En utilisant le code ASCII, décoder la phrase suivante : 76 39 97 114 99 104 105 44 32 99 39 101 115 116 32 102 97 99 105 108 101 46

   L’archi c’est facile.

3. En utilisant le code ASCII, décoder la phrase suivante donnée en hexadécimal : 4A 27 41 49 20 54 52 4F 55 56 45 20 21

   J’AI TROUVE !