

Correspondances hexadécimal / binaire / décimal, complément à 1 et complément à 2

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

Format des constantes, registres ou variables :

Octet = *Byte* = 1 octet = 8 bits = 2 digits hexa exemple : **0x2E**  
 Demi-mot = *Halfword* = 2 octets = 16 bits = 4 digits hexa exemple : **0x3EFF**  
 Mot = *Word* = 4 octets = 32 bits = 8 digits hexa exemple : **0x0300D010**

Conversion Hexa / Binaire : 1 digit hexa correspond à 4 digits binaires  
**0xAE3D** <-> **0b 1010 1110 0011 1101**

Conversion depuis ou vers les décimal : pas de correspondance simple entre chiffres.  
 Utiliser une table des puissances de 2 pour les nombres > 15.

n	2 <sup>n</sup>	n	2 <sup>n</sup>
0	1	9	512
1	2	10	1024
2	4	11	2048
3	8	12	4096
4	16	13	8192
5	32	14	16384
6	64	15	32768
7	128	16	65536
8	256	...	...
		32	~4,3 10 <sup>9</sup>

Complément à 1 (noté NOT pour les algorithmes)  
**NOT (0x6A)** -> **0x95** (utiliser la table)  
 Complément à 2 : c'est le complément à 1 +1  
**NOT (0xC2) +1** -> **0x3E** (attention hexa : 0x19+1 = 0x1A)

Notations pour préciser la base des constantes numériques :  
 - pas de préfixe -> décimal  
 - préfixe 0x -> hexa  
 - préfixe 0b -> binaire (seulement en assembleur)

Décalages et rotations

Les registres ARM sont 32 bits : les décalages et rotation se font sur 32 bits de large. Seul l'opérande 2 est concerné : voir doc <Oprnd2>.  
 Un décalage à gauche (resp. à droite) de n positions équivaut à multiplier (resp. diviser) par 2<sup>n</sup> (en beaucoup plus rapide pour le µprocesseur).

Opération	Terme anglais	Assembleur ARM	C	opérande	résultat
Décalage logique à gauche	<i>Logical shift left</i>	r0, LSL #3	x<<3	0b01010...11100 -> 0b10011...00011 ->	0b10...11100000 0b11...00011000
Décalage logique à droite	<i>Logical shift right</i>	r0, LSR #3	x>>3 x non signé	0b01010...11100 -> 0b10011...00011 ->	0b00001010...11 0b00010011...00
Déc. arithmétique à droite	<i>Arithmetic shift right</i>	r0, ASR #3	x>>3 x signé	0b01010...11100 -> 0b10011...00011 ->	0b00001010...11 0b11110011...00
Rotation à droite	<i>Rotate right</i>	r0, ROR #3	Pas d'opérateur de rotation en C	0b01010...11100 -> 0b10011...00011 ->	0b10001010...11 0b01110011...00

Opérateurs logiques

Opération	Terme anglais	Assembleur ARM	C	opérande1 (r1 ou y)	opérande2 (r2 ou z)	résultat (r0 ou x)
ET	<i>AND</i>	AND r0,r1,r2	x = y&z;	0b01010...	0b01100...	0b01000...
OU	<i>OR</i>	ORR r0,r1,r2	x = y z;	0b01010...	0b01100...	0b01110...
OU Exclusif	<i>Exclusive OR</i>	EOR r0,r1,r2	x = y^z;	0b01010...	0b01100...	0b00110...
NON (complément à 1)	<i>NOT</i>	MVN r0,r2	x = ~z;		0b01100...	0b10011...

Exemples d'utilisation :

ET pour masquer Ex. garder bit n°3	0b10010110 ET 0b00001000 0b00000000	0b11101001 ET 0b00001000 0b00001000	OU pour mettre à 1 Ex. bit n°5 à 1	0b10010110 OU 0b00100000 0b10110110	0b11101001 OU 0b00100000 0b11101001
OU Excl. pour inverser Ex. inverser bit n°4	0b10010110 OUx 0b00010000 0b10000110	0b11101001 OUx 0b00010000 0b11111001	ET pour mettre à 0 Ex. bit n°5 à 0	0b10010110 ET 0b11011111 0b10010110	0b11101001 ET 0b11011111 0b11001001

Exemples d'additions binaires sur format 8 bits

Arithmétique modulo 2<sup>8</sup> : 0xFF + 1 -> 0x00  
 u8 interprétation non signée / s8 interprétation signée

NZCV	Binaire	Hexa	u8	s8	Binaire	Hexa	u8	s8
	0b00000000	0x00	0	0	0b00000000	0x00	0	0
	0b01001110	0x4E	78	78	0b00000001	0x01	1	1
	0b00100101	0x25	37	37	0b00000010	0x02	2	2
0000	0b01110011	0x73	115	115	...	...	...	...
	0b01001110	0x4E	78	78	0b01111110	0x7E	126	126
	0b10011111	0x9F	159	-97	0b01111111	0x7F	127	127
1000	0b11101101	0xED	237	-19	0b10000000	0x80	128	-128
					0b10000001	0x81	129	-127
					...	...	...	...
					0b11111110	0xFE	254	-2
					0b11111111	0xFF	255	-1

Le complément à 2 correspond à la négation (inversion du signe) en signé :  
 62 (signé)=0x3E NOT(0x3E)+1 -> 0xC2=-62 (signé)  
 -41 (signé)=0xD7 NOT(0xD7)+1 -> 0x29= 41 (signé)

Le bit de poids le plus fort (le plus à gauche) indique le signe en interprétation signée :  
 0 positif, 1 négatif.  
 Si le nombre est positif, les interprétations signées et non signées sont identiques.  
 Si le nombre est négatif, la valeur absolue s'obtient en faisant le complément à 2.  
 Exemple : 0x9F=0b10011111 négatif  
 NOT(0x9F) + 1 -> 0x61 (97 en décimal)  
 En signé la valeur s'interprète comme -97