Architectures des ordinateurs
Cours 2

Représentation et codage des données

Eric Garcia

2002

IUT GTR, Montbéliard

Références

- http://lifc.univ-fcomte.fr/~teifreto/ASI-Cours/index.htm
- http://icps.u-strasbg.fr/~vivien/Enseignement/Archi-2001-2002/Cours.pdf
- http://asi.insa-rouen.fr/enseignement/siteUV/se/cours.html
- http://www-gtr.iutv.univ-paris13.fr/Equipe/viennet/Enseignement/
- http://www.lifl.fr/~simplot/ens/archi/

Plan

- **Bases**
  - numération en base *b*
  - opération arithmétiques et conversions
  - représentation des nombres
- Codage
  - codage des entiers
  - codage des réels
  - codage des caractères

Numération en base *b*

- Numérotation romaine
  - MCMXCIX : M vaut toujours 1000, C vaut 100…
- Numérotation arabe : représentation positionnelle
  - la rang de chaque chiffre indique son poids
  - dans 1999 le 9 le plus à droite vaut 9, le deuxième 90…
  - la valeur d'un chiffre dépend de sa position
- Représentations courantes des nombres
  - plusieurs bases : XXX_b indique que le nombre XXX est écrit en base ***b***
  - représentation usuelle : base 10
  - représentations courantes en informatique : bases 2 (**binaire**, interne), 8 (**octal**) et 16 (**hexadécimal**)

Numération en base *2, 8, 10 et 16*

Bases	**ai ∈ {Chiffres}**
b = 2 ou Binaire	0,1
b = 8 ou Octal	0,1,2,3,4,5,6,7
b = 10 ou Décimal	0,1,2,3,4,5,6,7,8,9
b = 16 ou Hexadécimal	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

⇨En base ***b*** on utilise ***b*** chiffres

⇨ $x = a_n a_{n-1} \ldots a_1 a_0$

⇨ a_0 est le chiffre des unités

Numération : exemples

Bases	Exemples
2 ou Binaire	$1001_2 = 9_{10}$
8 ou Octal	$142_8 = 98_{10}$
10 ou Décimal	128_{10}
16 ou Hexadécimal	$E2A1_{16} = 58017_{10}$

⇨ Pour les bases supérieures à 10 on peut également introduire des symboles séparateurs :

⇨ système sexagésimal : longitude = 2°20’14’’

⇨ Exo1

Changement de base : b à 10

- $N_{(10)} = d_0.b^0 + \ldots + d_i.b^i + \ldots + d_n.b^n$
  - i : rang
  - b^i : poids du chiffre
  - le chiffre le plus à gauche est le chiffre de poids fort, celui le plus à droite le chiffre de poids faible
- Exemple :
  - $143_{(5)} = 3*5^0 + 4*5^1 + 1*5^2 = 3+20+25 = 48_{(10)}$
  - $AB_{(16)} = 11*16^0 + 10*16^1 = 11+160 = 171_{(10)}$
  - $1001_{(2)} = 1*2^0 + 0*2^1 + 0*2^2 + 1*2^3 = 1+0+0+8 = 9_{(10)}$
  - le poids des chiffres en binaire et 1, 2, 4, 8, 16, 32, 64…

Intervalle d'un nombre de *n* chiffres

⇨ Avec ***n*** chiffres on peut coder en base ***b***, la valeur maximale suivante (cas où tous les chiffres sont à la valeur maximale soit b-1)

$$
\begin{aligned}
Nmax(10) &= (b-1).b^0 + \ldots + (b-1).b^i + \ldots + (b-1).b^{(n-1)} \\
&= (b-1) \,.\, (1 + \ldots + b^i + \ldots + b^{(n-1)}) \\
&= b \,.\, (1 + \ldots + b^i + \ldots + b^{(n-1)}) - (1 + \ldots + b^i + \ldots + b^{(n-1)}) \\
&= b + \ldots + b^i + \ldots + b^n - (1 + \ldots + b^i + \ldots + b^{(n-1)}) \\
&= b^n - 1
\end{aligned}
$$

⇨Avec ***n*** chiffres on donc coder b^n valeurs en base ***b***

⇨ Dans un ordinateur on code les nombres sur 8, 16 ou 32 bits

- ⇨ 8 bits : Nmax = 28 – 1 = 255 on peut coder 256 valeurs
- ⇨ 16 bits : Nmax = 65 535 (65 000 couleurs…)
- ⇨ 32 bit : Nmax = 4 294 967 296

Kilo, Méga, Tera

- On utilise souvent en informatique Kilo, Mega at Tera.
  - traditionnellement un Kilo vaut 1000 : 1Km = 1000m
  - en base 2, 1000 n'est pas une puissance de 2, on prend la valeur immédiatement supérieure
  - 1 Kilo = 2^{10} = 1024
  - 1 Mega = 2^{20} = 1 048 576
  - 1 Tera = 2^{30} = 1 073 741 824
- Un octet est un regroupement de 8 bits
  - 1 Ko = 1024 octets
  - 1 Mo = 1024 Ko
  - 1 Go = 1024 Mo

Changement de base : 10 à b

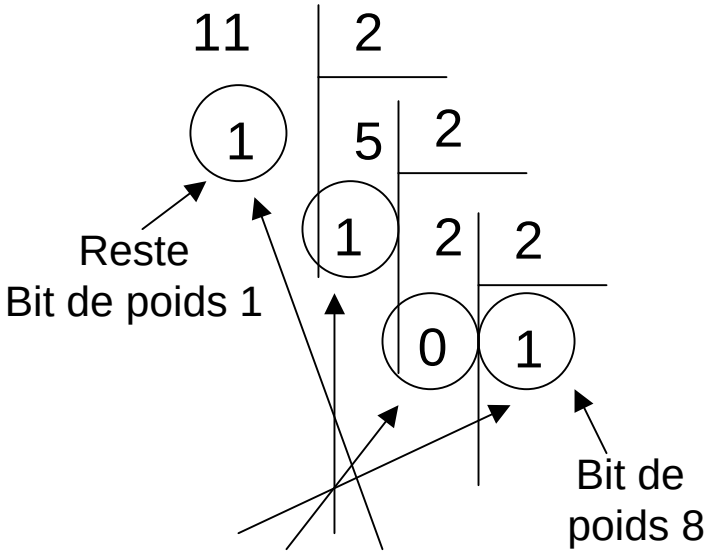
⇨ Algorithme des divisions successives

⇨ on divise le nombre par la base
⇨ puis le quotient par la base
⇨ ainsi de suite jusqu'à obtention d'un quotient nul
⇨ la suite des restes correspond aux chiffres dans la base visée
⇨ on obtient en premier le bit de poids faible et en dernier le bit de poids fort

⇨ Pour convertir un nombre d'une base b1 vers une base b2, il suffit de convertir le nombre de la base b1 en base 10, puis ce nombre en base 10 en base b2

Base 10 à 2 et 16 : exemples

conversion de $11_{(10)}$ en base 2

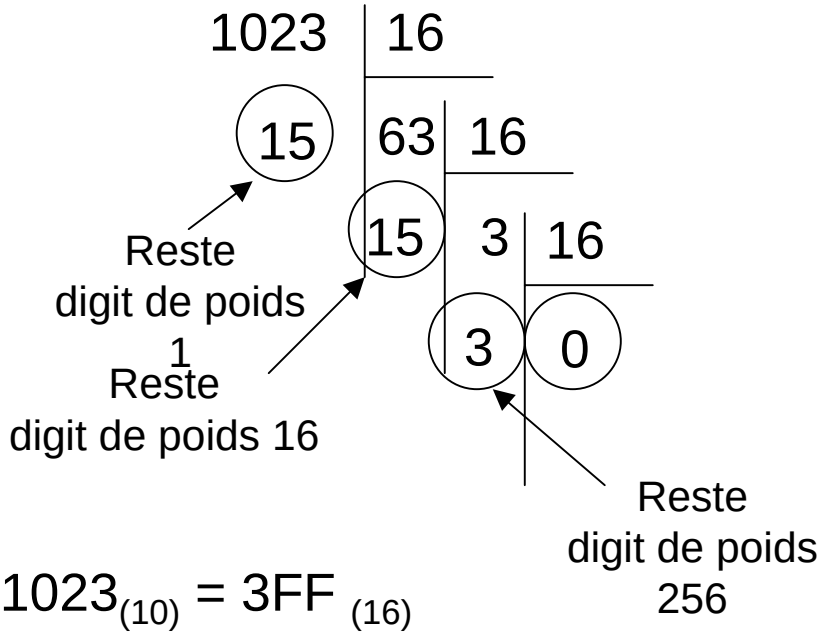


$11_{(10)} = 1\ 0\ 1\ 1\ _{(2)}$

Vérification :

$1*1 + 1*2 + 0*4 + 1*8 = 11$

conversion de $1023_{(10)}$ en base 16



$1023_{(10)} = 3FF\ _{(16)}$

Vérification :

$3*16^2 + 15*16 + 15*1 = 1023$

Particularité : bases 2, 8 et 16

⇨ Bases correspondant à des puissances de 2 : conversions très faciles

⇨ base 8 : découpage par blocs de 3 chiffres

⇨ 001 010 011 101
⇨ 1 2 3 5 = 1235_8

⇨ base 16 : découpage par blocs de 4 chiffres

⇨ 0010 1001 1101
⇨ 2 9 D = $29D_{16}$

⇨ Pour simplifier la notation binaire on utilise en informatique la notation hexadécimale

⇨ simplification de la lecture et écriture des adresses mémoires…

Opérations arithmétiques

- Mêmes principes que ceux utilisés en base 10

```
  1 1 1 1 1 1 1 1      255
+ 1 1 1 1 1 1 1 1    + 255
1 1 1 1 1 1 1 1 0    retenue
1 1 1 1 1 1 1 1 0      510

    F F              255
  + F F            + 255
    1 1 0          retenue
  -------          -------
    1 F E            510
```

```
          1 0 1
        X 1 0 1
  -------------
          1 0 1
        1 0 1
      1 0 1
  -------------
    1 1 0 0 1
```

A CORRIGER

- Dans l'ordinateur, les opérations sont faites en base 2
- Il existe aussi des opérations logiques (Cours n°3)

Représentation des nombres

- Les nombres sont codés en binaire
- Pour les nombres fractionnaires
  - on écrit la partie entière en binaire
  - on garde la virgule
  - la partie décimale se décompose en puissances négatives de 2
- Exemple : $0,75_{10}$
  - $0,75 * 2 = 1,5$ (on garde 1 reste 0,5)
  - $0,5 * 2 = 1$ (on garde 1 reste 0 : terminé)
  - $0,75_{10} = 0,11_2 = 1 * 2^{-1} + 1 * 2^{-2} = ½ + ¼$

Plan

- Bases
  - numération en base *b*
  - opération arithmétiques et conversions
  - représentation des nombres
- **Codage**
  - codage des entiers
  - codage des réels
  - codage des caractères

Types d'informations

⇨ Les types d'informations directement traitées par un processeurs sont :

⇨Les données

- ⇨ **entiers** : entiers naturels et entiers relatifs
- ⇨ **flottants** : décrivent les réels, simple et double précision
- ⇨ **caractères**
- ⇨ le codage de ces trois types est défini par des standards (normes spécifiées par des organisations internationales)

⇨Les instructions

- ⇨ leur codage est spécifique d'un processeur

L’information

- Dans un ordinateur, l’information est numérisée (digitale)
  - *L’information est la connaissance d’un état parmi un nombre fini d’états possibles*
  - Une information non numérisée est une information **analogique** : une grandeur physique continue (tension ou courant)
- L’unité de mesure de l’information est le bit
  - *un bit est la quantité d’information liée à la connaissance d’un état parmi deux*
  - un bit d’information peut être représenté par un digit binaire prenant les valeurs 0 ou 1
  - rappel : avec ***n*** bits, on peut représenter 2^n états

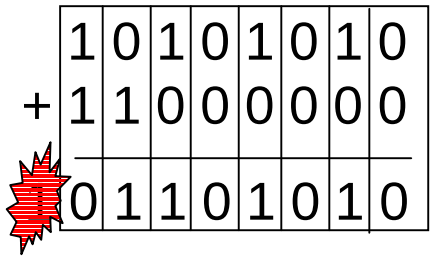
Codage : dépassement de capacité

- Codage = représentation avec un nombre limité de bits
  - Problème pour le dépassement de capacité
- Exemple pour les entiers naturels

Représentation

```
  1 0 1 0 1 0 1 0      170
+ 1 1 0 0 0 0 0 0    + 192
-----------------    -----
1 0 1 1 0 1 0 1 0      362
```

Codage sur 8 bits



362 est trop petit pour être codé sur 8 bits

Codage des entiers naturels

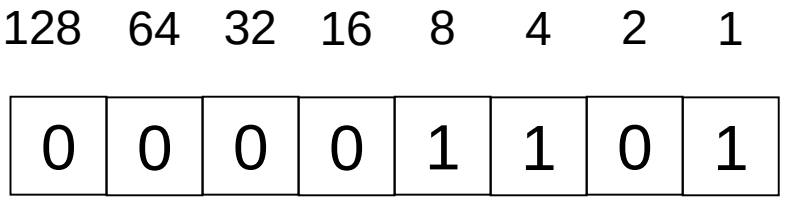
⇨ Entiers naturels (*unsigned* en C) codés sur un nombre d'octets fixé (un octet = 8 bits)

⇨ généralement 1, 2 (short en C) ou 4 (int ou long en C)

⇨ Sur un octet on peut coder les nombres de 0 à 255 (2^8 possibilités) : attentions aux dépassements (***overflow***)

⇨ deux possibilités : ignorer l'overflow (tourne) ou provoquer l'exécution d'une routine particulière

⇨ Nombre représenté en base 2, les bits sont rangés dans des cellules correspondant à leur poids, on complète à gauche par des 0



Codage des entiers relatifs

- Problème : coder le signe du nombre
- Idée simple : introduire un bit de signe (0 : + ; 1 : -)
  - sur 4 bits : 0110 = 6 et 1110 = - 6
  - sur n bits on code $[-(2^{n-1}-1),2^{n-1}-1]$: car on utilise n-1 bits pour coder le nombre
- Limitations de ce type de solution
  - deux représentations pour 0 : 0000 ou 1000
  - addition difficile alors que c'est l'opération la plus utilisée (extraction du signe, test pour faire une addition ou une soustraction, calcul du nouveau signe)
- Pour simplifier l'addition on représente les entiers négatifs en complément à 2

Complément à deux

⇨ La représentation de –x est obtenue par le complément à deux de x

- ⇨ on code en binaire sa valeur absolue
- ⇨ on complémente (inverse) tous les bits
- ⇨ on ajoute 1

⇨ Exemple : pour x = 22 sur 1 octet

- ⇨ X = 0 0 0 1 0 1 1 0 : on complémente
- ⇨ 1 1 1 0 1 0 0 1 : on ajoute 1
- ⇨ -X = 1 1 1 0 1 0 1 0

⇨ Le premier bit d'un nombre indique son signe, s'il est négatif, son complément permet de retrouver sa valeur absolue

Complément à 2 et entiers relatifs

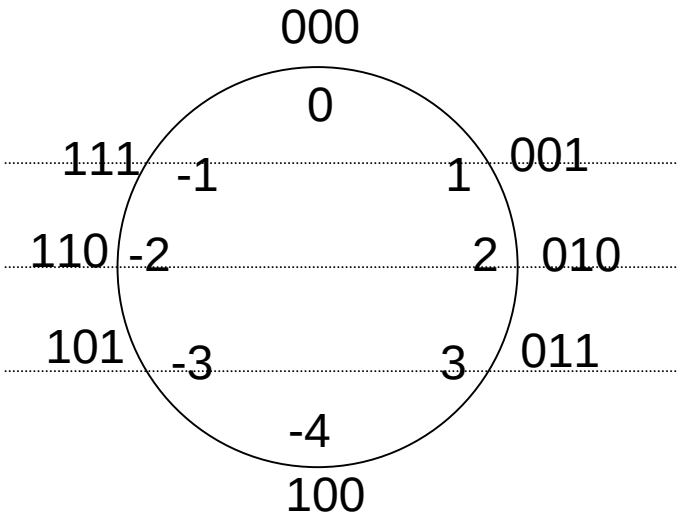
⇨ En machine les nombres négatifs sont directement codés en complément

- ⇨ soustraction : on complémente la deuxième opérande et on effectue une addition
- ⇨ addition et soustraction : un seul circuit réalisant l'addition
- ⇨ on ne considère pas le signe des nombres

⇨ Représentation non complètement symétrique : le plus petit nombre n'a pas d'opposé : sur n bits

- ⇨ le plus grand entier positif est $\mathbf{2^{n-1}-1}$
- ⇨ le plus petit entier négatif est $\mathbf{-2^{n-1}}$

Entiers relatifs : exemple



Algorithme d'addition traditionnel avec les deux méthodes

```
  0 0 0 0 0 0 0 1
+ 1 0 0 0 0 0 0 1
-----------------
  1 0 0 0 0 0 1 0
```

+1 –1 = -2 la méthode simple du bit de signe ne fonctionne pas

retenue → (1)

```
  0 0 0 0 0 0 0 1
+ 1 1 1 1 1 1 1 1
-----------------
  0 0 0 0 0 0 0 0
```

+1 –1 = 0 la méthode du complément à deux fonctionne

Codage des réels : virgule flottante

⇨ Par exemple codifié 3,25 qui s'écrit $(11,01)_2$

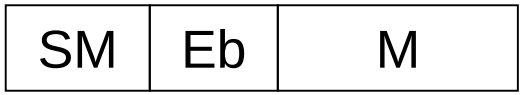
⇨ première machine = virgule fixe : nombre séparé en deux parties (chiffres avant et après la virgule)

⇨ Depuis les années soixante : virgule flottante

⇨ flottant stocké sous la forme $M * B^E$
⇨ M : Mantisse ; B : Base ; E : Exposant
⇨ exemple : $123 . 10^3 = 123\ 000$

⇨ Représentation IEEE 754 (signe 1 bit, exposant et mantisse sur 32 ou 64 bits pour simple et double précision)

⇨ SM : signe de la mantisse : 1 bit
⇨ Eb : exposant biaisé : 8 ou 11 bits
⇨ M : Mantisse : 23 ou 52 bits



Mantisse et exposant

- Signe : bit de poids fort (0 = + ; 1 = -)
- Exposant
  - placé avant la mantisse pour simplifier les comparaisons (pour ceci il ne doit pas être représenté en complément à deux : $2^{-1} > 2$)
  - sans signe mais biaisé de 127 (simple précision) :
    - Eb = 0 $\Rightarrow$ E = 0 – 127 = -127
    - Eb = 255 $\Rightarrow$ E = 255 – 127 = 128
  - les exposants 128 (erreur) et 0 (nb dénormalisé) sont interdits
- Mantisse
  - normalisée : bit de poids fort n'est pas 0 et un seul chiffre avant la virgule
  - ex : $11,01 = 1,101 * 2^1$

Évaluation d'un réel

SM	Eb	M f1f2…fn

⇨ Comme le bit de poids fort de la mantisse est nécessairement 1 : on ne l'indique pas (gaspillage de place), il est implicite

⇨ Mantisse

⇨ partie fractionnaire = $f_1f_2 \ldots f_n$ ⇒ $m = 1,f_1f_2 \ldots f_n$

⇨ nombre $x = (-1)^{SM} * 1,M * 2^{Eb-127}$

⇨ Exemple

⇨ $x = (-2,5)_{10} = -1,01*2^1{}_2$ (SM = 1 ; Eb = 1000 0000 ; M = 01)

Opérations

- Additions et soustractions
  - il faut que les exposants aient la même valeur
  - exemple : $3 * 10^3 + 9{,}98 * 10^5 =$
  - on dénormalise : $3 * 10^3 = 0{,}03 * 10^5$
  - on additionne les mantisses : $0{,}03 + 9{,}98 = 10{,}01$
  - on normalise : $10{,}01 * 10^5 = 1{,}001 * 10^6$
- Multiplications et divisions
  - on additionne ou on soustrait les exposants
  - on multiplie les mantisses
  - on normalise le résultat
- Nombre infini de valeurs dans un intervalle : codées avec un nombre fini de bits $\Rightarrow$ problème de précision (arrondi)

Codage des caractères

- Caractères : symboles alphanumériques (& , . A 5 …)
  - données non numériques (l'addition n'a pas de sens)
  - comparaison ou tri utile
  - table de correspondance entre les nombres et les caractères
- Code ASCII (American Standard Code for Information Interchange)
  - 7 bits pour représenter les caractères (ASCII étendu = 8 bits)
  - 0 (00) à 31 (1F) : caractères spéciaux : sauts, bips…
  - 48 à 57 : chiffres dans l'ordre (valeur = 4 bits de poids faible)
  - 65 à 90 les majuscules et 97 à 122 les minuscules (on modifie le 5ème bit pour passer des majuscules aux minuscules : +32)…
- En machine on accède aux mots par octet
  - ASCII étendu : caractères accentués…
  - Compatibilité : nécessaire d'expliciter le codage utilisé : *iso-latin-1*

Code ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com