Leaning Aims

ASCII and Unicode

- Define a bit as a 1 or a 0, and a byte as a group of eight bits
- Know that 2^n
- different values can be represented with n bits
- Use names, symbols and corresponding powers of 2 for binary prefixes e.g. Ki, Mi
- Differentiate between the character code of a denary digit and its pure binary representation
- Describe how character sets (ASCII and Unicode) are used to represent text

Unit nomenclature

- Although we frequently refer to 1024 bytes as a kilobyte, it is in fact a kibibyte.
- To avoid any confusion between references to 1024 bytes rather than 1000 bytes, an international collaboration between standards organisations decided in 1996 that kibi would represent 1024, and kilo would represent 1000.
- Kibi is a combination of the words kilo and binary.
- The same is true of the other familiar names Mega, Giga and Tera being replaced by mebi, gibi and tebi.
- The table below outlines the nomenclature for increasing quantities of bytes, in which a KiB is a kibibyte and a MiB, a mebibyte.

Name	Symbol	Power	Value
kibi	Ki	2^{10}	1024
mebi	Mi	2^{20}	1,048,576
gibi	Gi	2^{30}	1,073,741,824
tebi	Ti	2^{40}	1,099,511,627,776
pebi	Pi	2^{50}	1,125,899,906,842,624
exbi	Ei	2^{60}	1,152,921,504,606,846,976
zebi	Zi	2^{70}	1,180,591,620,717,411,303,424
yobi	Yi	2^{80}	1,208,925,819,614,629,174,706,176

Name	Symbol	Power
Kilo	K or k	10^{3}
Mega	M	10^{6}
Giga	G	10^{9}
Tera	T	10^{12}
Peta	P	10^{15}
Exa	E	10^{18}
Zetta	Z	10^{21}
Yotta	Y	10^{24}

The ASCII code

Historically, the standard code for representing the characters on the keyboard was ASCII (American Standard Code for Information Interchange).

This uses seven bits which form **128 different bit combinations**, more than enough to cover all of the characters on a standard English-language keyboard.

The first 32 codes represent non-printing characters used for control:

- backspace (code 8)
- Enter or Carriage Return key (code 13)
- Escape key (code 27).
- Space character code 32
- Delete as code 127.

ASCII	DEC	Binary	ASCII	DEC	Binary	ASCII	DEC	Binary	ASCII	DEC	Binary
NULL	000	000 0000	space	032	010 0000	@	064	100 0000	`	096	110 0000
SOH	001	000 0001	!	033	010 0001	A	065	100 0001	a	097	110 0001
STX	002	000 0010	"	034	010 0010	B	066	100 0010	b	098	110 0010
ETX	003	000 0011	#	035	010 0011	C	067	100 0011	c	099	110 0011
EOT	004	000 0100	$	036	010 0100	D	068	100 0100	d	100	110 0100
ENQ	005	000 0101	%	037	010 0101	E	069	100 0101	e	101	110 0101
ACK	006	000 0110	&	038	010 0110	F	070	100 0110	f	102	110 0110
BEL	007	000 0111	'	039	010 0111	G	071	100 0111	g	103	110 0111
BS	008	000 1000	(	040	010 1000	H	072	100 1000	h	104	110 1000
HT	009	000 1001	)	041	010 1001	I	073	100 1001	i	105	110 1001
LF	010	000 1010	*	042	010 1010	J	074	100 1010	j	106	110 1010
VT	011	000 1011	+	043	010 1011	K	075	100 1011	k	107	110 1011
FF	012	000 1100	,	044	010 1100	L	076	100 1100	l	108	110 1100
CR	013	000 1101	-	045	010 1101	M	077	100 1101	m	109	110 1101
SO	014	000 1110	.	046	010 1110	N	078	100 1110	n	110	110 1110
SI	015	000 1111	/	047	010 1111	O	079	100 1111	o	111	110 1111
DLE	016	001 0000	0	048	011 0000	P	080	101 0000	p	112	111 0000
DC1	017	001 0001	1	049	011 0001	Q	081	101 0001	q	113	111 0001
DC2	018	001 0010	2	050	011 0010	R	082	101 0010	r	114	111 0010
DC3	019	001 0011	3	051	011 0011	S	083	101 0011	s	115	111 0011
DC4	020	001 0100	4	052	011 0100	T	084	101 0100	t	116	111 0100
NAK	021	001 0101	5	053	011 0101	U	085	101 0101	u	117	111 0101
SYN	022	001 0110	6	054	011 0110	V	086	101 0110	v	118	111 0110
ETB	023	001 0111	7	055	011 0111	W	087	101 0111	w	119	111 0111
CAN	024	001 1000	8	056	011 1000	X	088	101 1000	x	120	111 1000
EM	025	001 1001	9	057	011 1001	Y	089	101 1001	y	121	111 1001
SUB	026	001 1010	:	058	011 1010	Z	090	101 1010	z	122	111 1010
ESC	027	001 1011	;	059	011 1011	[	091	101 1011	{	123	111 1011
FS	028	001 1100	<	060	011 1100	\	092	101 1100	|	124	111 1100
GS	029	001 1101	=	061	011 1101	]	093	101 1101	}	125	111 1101
RS	030	001 1110	>	062	011 1110	^	094	101 1110	~	126	111 1110
US	031	001 1111	?	063	011 1111	_	095	101 1111	DEL	127	111 1111

Character form of a denary digit

Although numbers are represented within the code, the number character is not the same as the actual
number value.

The ASCII value 0110111 will print the character ‘7’, even though the same binary value equates to the denary number 55.

Therefore ASCII cannot be used for arithmetic and would use unnecessary space to store numbers.

Numbers for arithmetic are stored as pure binary numbers.

‘7’ + ‘7’ (i.e. 0110111 + 0110111 in ASCII) would be 77, not 14 or 110.

The development of ASCII

ASCII originally used only 7 bits, but an 8-bit version was developed to include an additional 128 combinations to represent symbols such as æ, © and ƒ.

You can try holding down the ALT key and typing in the code number using the number pad to type one of these symbols.

For example, ALT+130 will produce é, as used in café.

The 7-bit ASCII code is compatible with the 8-bit code and simply adds a leading 0 to all binary codes.

Unicode

- By the 1980s, several coding systems had been introduced all over the world that were all incompatible with one another.
- This created difficultly as multilingual data was being increasingly used and a new, unified format was sought. As a result, a new **16-bit code** called Unicode (UTF-16) was introduced.
- This allowed for 65,536 different combinations and could therefore represent alphabets from dozens of languages including Latin, Greek, Arabic and Cyrillic alphabets.
- The first 128 codes were the same as ASCII so compatibility was retained.
- A further version of Unicode called **UTF-32** was also developed to include just over a million characters, and this was more than enough to handle most of the characters from all languages, including Chinese and Japanese.
- This meant that whilst there is now just one globally recognised system to maintain, one character in this scheme uses four bytes instead of two, significantly increasing file sizes and data transmission times.