A brief history of Unicode

😊 happens

Alex Blewitt
@alblue

What is Unicode?

- Unicode is an industry standard for representing text
- Defines a number of code points that map to characters
  - Not all characters are visible (control characters)
  - Not all characters are standalone (accents)
  - Not all code points refer to characters (some are undefined)
  - Does include all major ideographs from a variety of languages
  - `U+0041` == ‘A’, `U+20AC` == ‘€’
- Pop quiz: what size are Unicode code points?
  - 8-bit
  - 16-bit
  - 32-bit

Unicode: a 21-bit code point

- All characters in Unicode are logically 21-bits wide
  - Not a great format for encoding data in computers!
  - How did we end up with a 21-bit character set?
- To explain that, we have to look backwards in time …
- Before Unicode …
  - Many variations of character sets with different meanings
  - Single-byte
    - ISO-8859-1 (CP-1252), ISO-8859-2, … ISO-8859-9
    - ASCII, EBCDIC
  - Multi-byte
    - ISO-2202-CN, ISO-2202-JP, ISO-2202-KR (CJK)

What does all of this mean?

- Character sets and code pages assigned meanings
  - 0x41 = ‘A’
  - 0xD0 = ?
    - ISO-8859-1 = ‘Đ’
    - ISO-8859-3 = <missing>
    - ISO-8859-9 = ‘Ğ’
    - EBCDIC = ‘}’
  - All based on ASCII (well, except EBCDIC …)

- Pop quiz: what size are ASCII code points?
  - 8-bit
  - 16-bit
  - 32-bit

ASCII is a 7-bit code point

- Who needs power-of-two?
  - American Standard Code for Information Interchange
  - Defined to harmonise existing incompatible encodings
    - ASCII was the Unicode of the telegraph era
- First 128 characters of ASCII are same as
  - Unicode
  - ISO-8859-1 (aka Latin-1)
  - CP1252 (Windows)
  - …
- Where did ASCII come from?

ASCII

USASCII code chart

Upper Lower

Bits b4	b3	b2	b1	b7 b6 b5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
				Row ↓ / Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Numbers Control Punctuation

http://en.wikipedia.org/wiki/ASCII#/media/File:ASCII_Code_Chart-Quick_ref_card.png

ASCII control characters

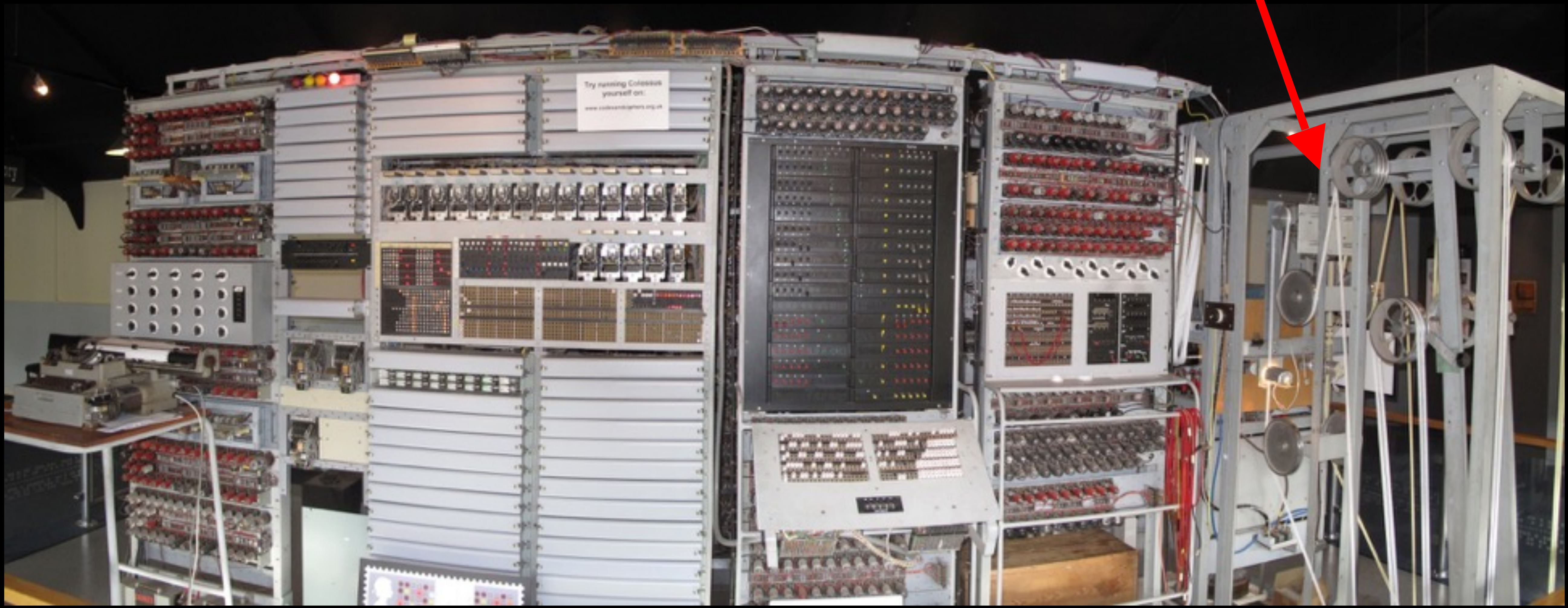
- Many are now obsolete but stem from telegraph days
  - XML disallows control characters other than CR, LF, HT
- Some were used for printer control mechanisms
  - HT/VT – horizontal or vertical tab (^I/^K)
  - LF/FF – line feed/form feed (^J/^L)
  - CR – carriage return (^M)
- Some are used for notification
  - BEL – ring the bell (^G is beep in Unix terminals)
- Some were used for notification
  - ACK/NAK/STX/ETX/SYN
  - ESC/NUL

Telegraphs and teletypes

- Telegraphs revolutionised communication
  - Characters sent as an electric encoding of bits
  - Various encoding supported characters
  - Needed standardisation …
- Teletype printers would print out punched paper tapes
  - Paper tapes could be optically read
  - /dev/tty in Unix stands for ‘teletype’
  - /dev/ttyS1 stands for ‘teletype on serial port 1’
- Punched cards and tapes were common

Colossus computer

Used to crack codes from the Lorenz telegraph with paper tape



http://en.wikipedia.org/wiki/Colossus_computer

Baudot, Murray and ITA2

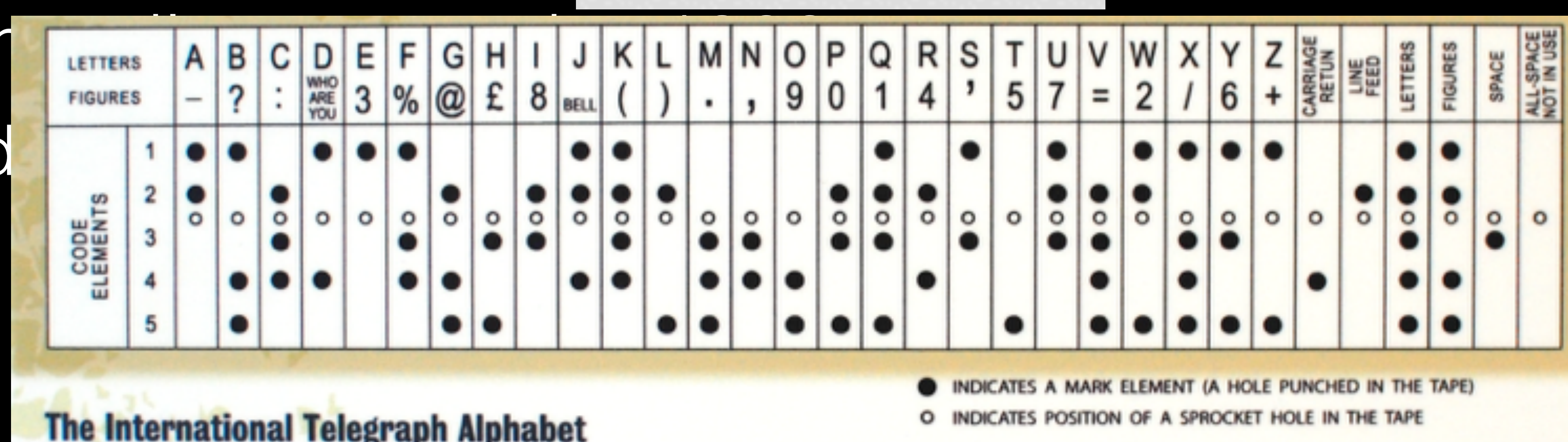
- Baudot created first fixed length 5-bit encoding
  - Also gave name to ‘baud’ as symbols-per-second (not bits)
  - Became known as ITA1
  - Created ~ 1870
- Murray encoding created ~ 1900
  - Modified patterns to minimise wear on punches
  - Defined NUL as 0, introduced CR and LF, Backspace
  - Evolved to ITA2 ~ 1930

Baudot, Murray and ITA2

- Baudot created first fixed length 5-bit encoding
  - Also gave name to ‘baud’ as symbols-per-second (not bits)
  - Became known as ITA1
  - Created ~ 1870

← Sprocket drive holes

- Murray en
  - Modified
  - Defined
  - Evolved



LETTERS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
FIGURES	–	?	:	WHO ARE YOU	3	%	@	£	8	BELL	(	)	.	,	9	0
CODE ELEMENTS 1	●	●		●	●	●				●	●					
2	●		●				●		●	●	●	●				●
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
3			●			●		●	●		●		●	●		●
4		●	●	●		●	●			●	●		●	●	●	
5		●					●	●				●	●		●	●

LETTERS	Q	R	S	T	U	V	W	X	Y	Z	CARRIAGE RETUN	LINE FEED	LETTERS	FIGURES	SPACE	ALL-SPACE NOT IN USE
FIGURES	1	4	’	5	7	=	2	/	6	+						
CODE ELEMENTS 1	●		●		●		●	●	●	●			●	●		
2	●	●			●	●	●					●	●	●		
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
3	●		●		●	●		●	●				●		●	
4		●				●		●			●		●	●		
5	●			●		●	●	●	●	●			●	●		

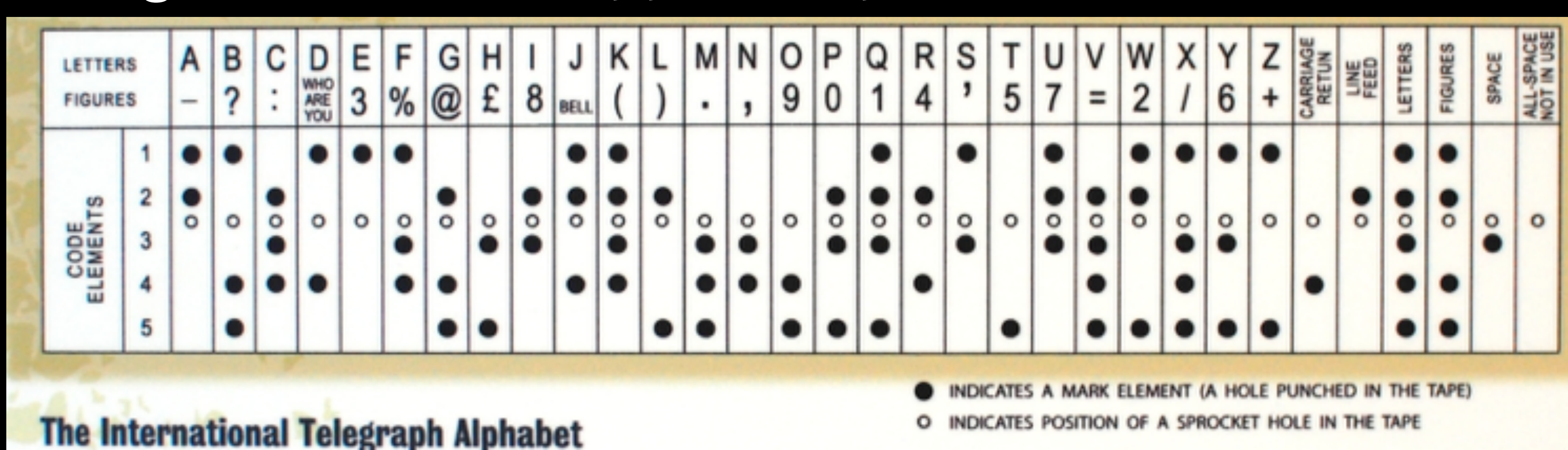
● INDICATES A MARK ELEMENT (A HOLE PUNCHED IN THE TAPE)
○ INDICATES POSITION OF A SPROCKET HOLE IN THE TAPE

The International Telegraph Alphabet

http://en.wikipedia.org/wiki/Baudot_code

Shifting in Baudot code

- The astute of you will notice 5 bits isn't enough
  - 26 letters + 10 digits > 2^5 (32)
- This was solved with the idea of a shift
  - Based on idea of typewriters
  - Meant that decoding was based on state
    - **Letter mode** – `Hello World`
    - **Figures mode** – `£3))9 294)`



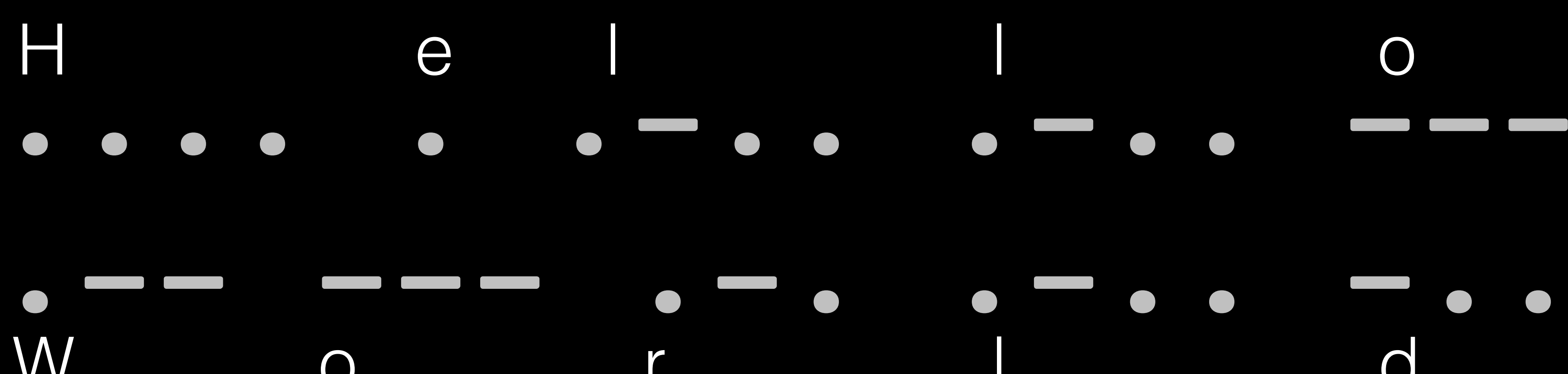
LETTERS	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	CARRIAGE RETUN	LINE FEED	LETTERS	FIGURES	SPACE	ALL-SPACE NOT IN USE
FIGURES	–	?	:	WHO ARE YOU	3	%	@	£	8	BELL	(	)	.	,	9	0	1	4	'	5	7	=	2	/	6	+						
CODE ELEMENTS 1	●	●		●	●	●				●	●						●		●		●		●	●	●	●			●	●		
2	●		●				●		●	●	●	●				●	●	●			●	●	●					●	●	●		
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
3			●			●		●	●		●		●	●		●	●		●		●	●		●	●				●		●	
4		●	●	●		●	●			●	●		●	●	●			●				●		●			●		●	●		
5		●					●	●				●	●		●	●	●			●		●	●	●	●	●			●	●		

● INDICATES A MARK ELEMENT (A HOLE PUNCHED IN THE TAPE)
○ INDICATES POSITION OF A SPROCKET HOLE IN THE TAPE

The International Telegraph Alphabet

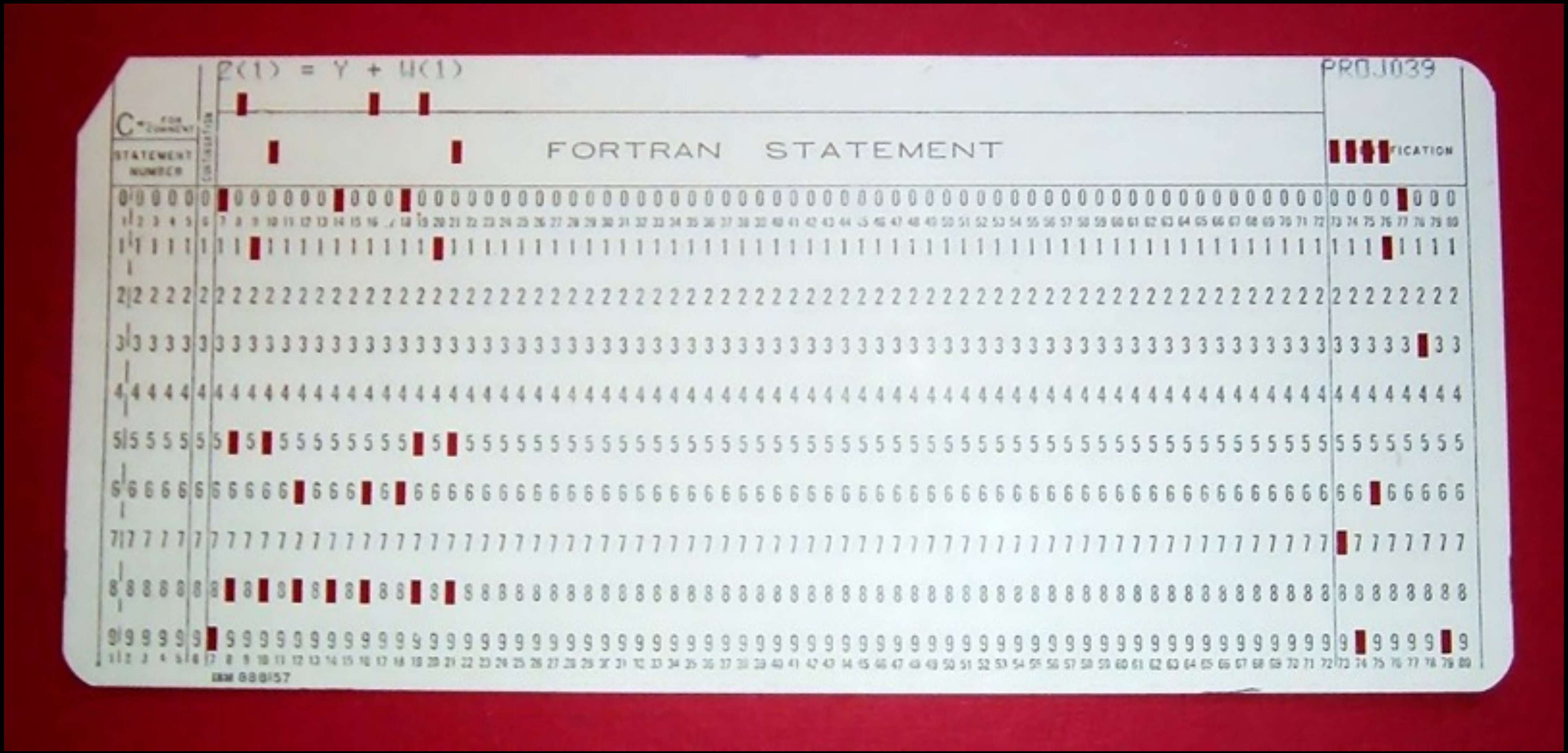
Morse Code

- Morse code is a variable length encoding
  - Dots or dashes to represent characters
  - Initial encoding for radio with human operators
  - Invented in ~1840
- Practical for humans to hear and decode / send



Punched Cards

- Punched tape itself was an evolution of cards
  - Each card represented a 'line', each column a letter



http://en.wikipedia.org/wiki/Punched_card

Punched Cards

- Punched tape itself was an evolution of cards
  - Each card represented a ‘line’, each column a letter

http://en.wikipedia.org/wiki/Punched_card
http://en.wikipedia.org/wiki/Silver_certificate_(United_States)

When were punched cards used?

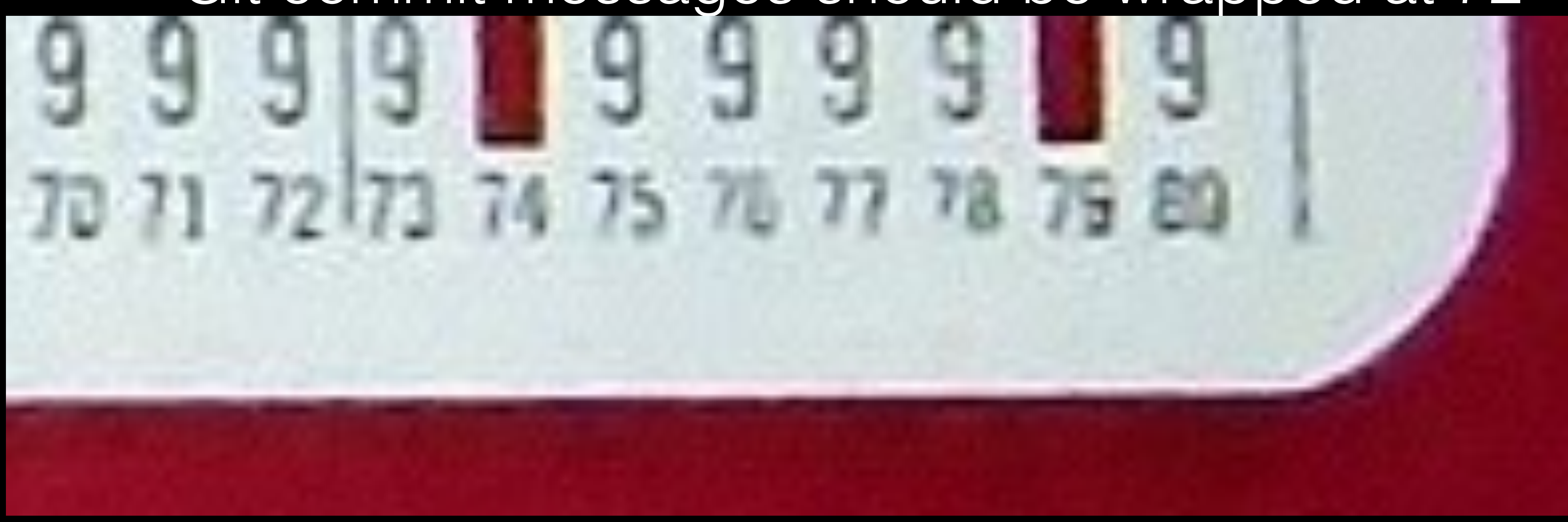
- When were punched cards
  - 1960
  - 1950
  - 1940
  - 1930
  - 1920
  - 1910
  - …

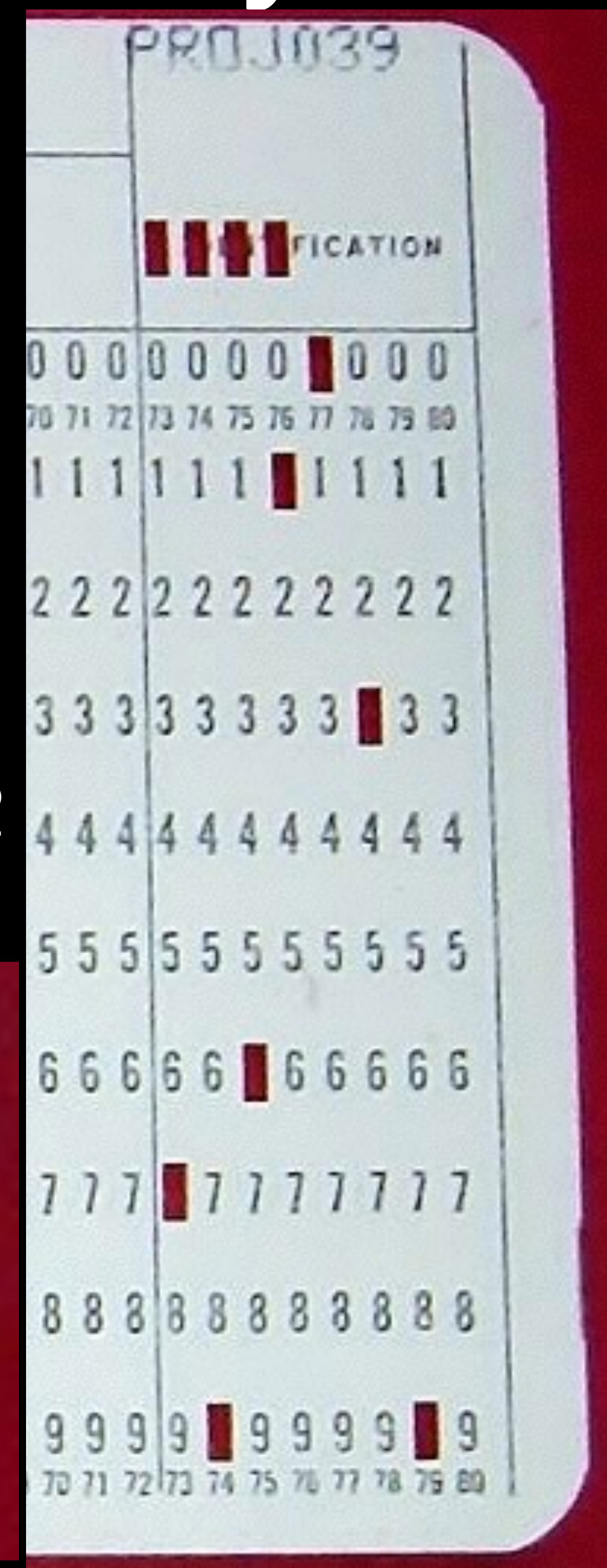
Jaquard Loom
1800

US Census
1890

Punched cards legacy

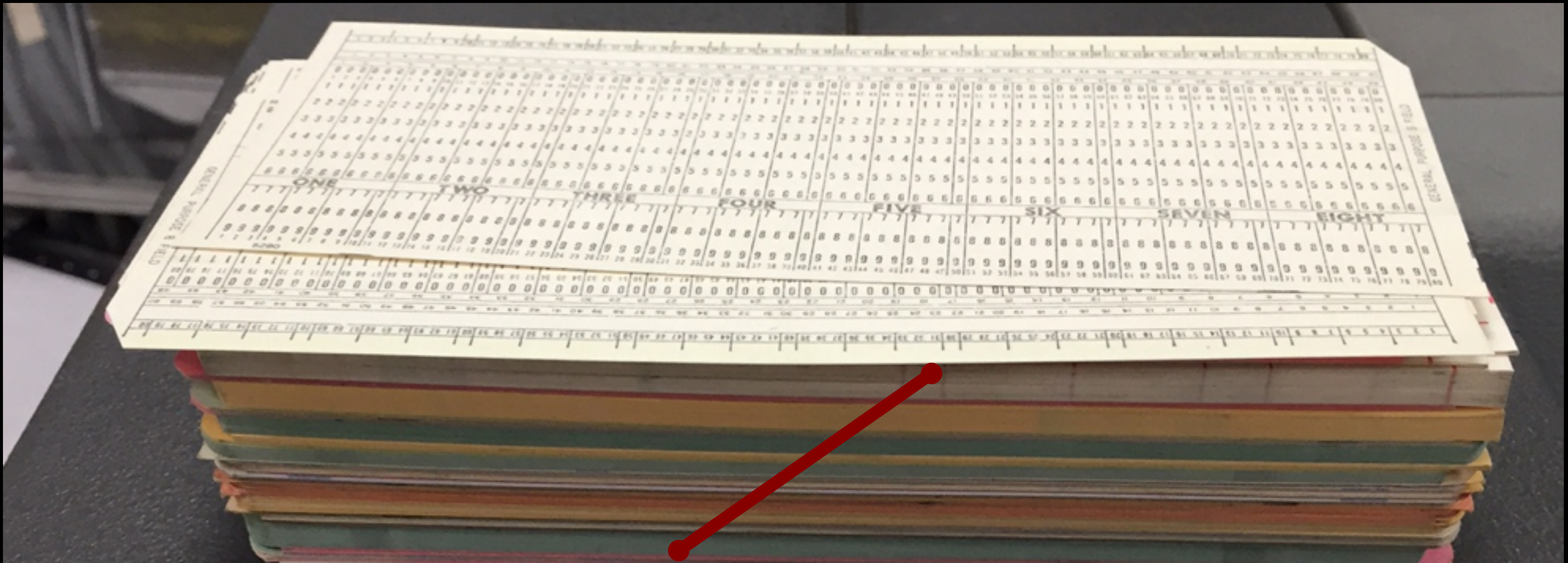
- Legacy of punched cards still with us
  - Cards were 80 columns wide
    - Led to early terminals having an 80 col display
    - Some IDEs and text editors have a wrap at 80
  - 8 characters were often used for numbering
    - Fortran ignored characters in columns 73-80
    - Some text editors will wrap /warn after column 72
    - Git commit messages should be wrapped at 72





Punched cards and line numbers

- Dropping a stack of cards was an expensive operation …
  - Radix sort of columns 73-80 can be used to fix
  - Or just put a diagonal line through them …



EBCDIC

- EBCDIC is the Extended BCD Interchange Code
  - BCD is Binary Coded Decimal, e.g. 0x12 is 12 decimal

0-9 in BCD is 0000..1010

Figure 4. Card Codes and Graphics for 64-Character Set

http://www.columbia.edu/cu/computinghistory/

EBCDIC

0-9 in BCD is 0000..1010

EBCDIC Code Table

B4	B3	B2	B1	B8 / B7 / B6 / B5	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
				HEX-1 \ HEX-0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NUL	DLE	DS		SP	&	-									0
0	0	0	1	1	SOH	SBA	SOS				/		a	j			A	J		1
0	0	1	0	2	STX	EUA	FS	SYN					b	k	s		B	K	S	2
0	0	1	1	3	ETX	IC							c	l	t		C	L	T	3
0	1	0	0	4	PF	RES	BYP	PN					d	m	u		D	M	U	4
0	1	0	1	5	PT	NL	LF	RS					e	n	v		E	N	V	5
0	1	1	0	6	LC		ETB	UC					f	o	w		F	O	W	6
0	1	1	1	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
1	0	0	0	8		CAN							h	q	y		H	Q	Y	8
1	0	0	1	9		EM							i	r	z		I	R	Z	9
1	0	1	0	A	SMM	CC	SM		¢	!	¦	:								
1	0	1	1	B	VT				.	$	,	#								
1	1	0	0	C	FF	DUP		RA	<	*	%	@								
1	1	0	1	D	CR	SF	ENQ	NAK	(	)	_	'								
1	1	1	0	E	SO	FM	ACK		+	;	>	=								
1	1	1	1	F	SI	ITB	BEL	SUB	|	¬	?	"								

http://ferretronix.com/march/computer_cards/ebcdic_table.jpg

EBCDIC challenges

- Not all was well with the EBCDIC character set
  - Rarely used outside of IBM mainframes
  - Different sort ordering to ASCII
    - ASCII has 0-9, A-Z, a-z
    - EBCDIC has a-z, A-Z, 0-9 (and not contiguous; ‘a’-‘z’ != 25)
  - Created around same time (1963)
    - IBM’s mainframes had peripherals using punched cards
    - Easier to translate punched cards into EBCDIC
    - Mainframes could be switched into ASCII but programs failed
  - Shares similar control characters to ASCII
    - Form Feed, Tab, Escape …

Putting history together

Telegraph

Morse Code (1840)

Baudot Code (1870)

Murray/ITA2 (1900)

Automation

Jacquard Loom (1800)

Hollerith Card (1890)

Computing

Fortran (1960)

ASCII (1963)

EBCDIC (1963)

ISO-8859-* (1985)

Unicode 1.0 (1991) – 16 bit

Unicode 2.0 (1996) – 21bit

Why a 21 bit code, though?

- Unicode 1.x was a 16-bit code
  - Not enough to store everything
  - Needed to have additional ‘planes’
- Plane 0: “Basic Multilingual Plane” was most of 1.x
- Plane 1: “Supplemental Multilingual Plane” added
  - Emoji
  - Egyptian Hieroglyphs
  - Graphics characters such as dominoes and playing cards
- Plane 2 .. 16: “Supplementary planes” of various types

Still doesn’t explain 21 bit

- To represent additional planes requires encoding
- Two main Unicode encodings are widely used
  - UTF-8
  - UTF-16 (formerly UCS-2)
- Unicode Transformation Format says how to encode point
  - Logical code point for € is U+20AC
  - May be written out in different ways
    - 0x20 0xAC
    - 0xAC 0x20
- UTF-16 uses 2 octets (16-bits) to represent content
- UTF-8 uses octets (bytes/8-bit) to represent content

UTF-16

- UTF-16 uses two octets to represent content
  - Can be ‘big endian’ or ‘little endian’
    - 0x20 0xAC is ‘big endian’
    - 0xAC 0x20 is ‘little endian’
  - Byte Order Mark (BOM 0xFE 0xFF) often written out at front
    - 0xFE 0xFF – ‘big endian UTF-16 BOM’ – þÿ in ISO-8859-1
    - 0xFF 0xFE – ‘little endian UTF-16 BOM’ – ÿþ in ISO-8859-1

- Still only 16 bit – how are planes 1..16 represented?
    - Surrogate pairs allow encoding 20 bits worth of data in 4 octets
    - High surrogate pair (10 bits)
    - Low surrogate pair (10 bits)

But 10 + 10 != 21 …

- No, but there's no need to use them for plane 0 (BMP)
  - So, take away 1 and you have planes 0..15 which is 4 bits
  - 4 bits + 16 bits (65536 in each plane) = 20 bits
- Consider 7 o'clock symbol 🕖
  - `U+1F556` (The leading 1 indicates it is in plane 1)
  - Plane 1 is encoded as `0000`
  - F5 is `1111 0101`
  - 56 is `0101 0110`
- UTF-16 for U+1F556 is
  - `110110 0000 1111 01 == 0xD83D`
  - `110111 01 0101 0110 == 0xDD5A`

UTF-8 stores 21 bits in 4 octets

- UTF-8 is a variable length encoding
  - ASCII bytes (<= 127, <= `U+007F`) are encoded as one octet
  - `U+0080..U+07FF` are encoded as two octets
  - `U+0800..U+FFFF` are encoded as three octets
  - `U+10000..U+1FFFFF` are encoded as four octets
- Single octets
  - Always start with a `0`
- Multi octets
  - Start with `11`
  - Continuation octet starts with `10`

Designed by Ken Thompson and Rob Pike

UTF-8 examples

- `U+0041` A
  - `0x41`
- `U+1F556` 🕖
  - U+1 is `00001`
  - F5 is `1111 0101`
  - 56 is `0101 0110`
  - Encoded as 4 octets `0xF09F9596`
    - `11110  000 == 0xF0`
    - `10 01 1111 == 0x9F`
    - `10 0101 01 == 0x95`
    - `10 010110  == 0x96`

ï»¿ is the UTF-8 encoded UTF-16 byte order mark

Doesn't make sense

Generated by Windows

The number of bits in
the first part shows number
of bytes in code

Flags of all nations

- How are flags represented? 🇬🇧🇪🇺🇺🇸
  - Extensible way without adding new data
  - Regional indicator symbols **A … Z**

G B 🇬🇧 U+1F1EC U+1F1E7

E U 🇪🇺 U+1F1EA U+1F1FA

Symbols replaced with flag as standard font ligatures

U S 🇺🇸 U+1F1FA U+1F1F8

UTF-8: 0xF09F 87BA F09F 87B8

UTF-16: 0xFE FF D83C DDFA D83C DDF8

Unicode: a 21-bit code point

- Expanded from 16 bits with 1.x to 21 bits with 2.x
- Encodings for UTF-8 provide a way to store 21 bits
  - Can scan through string to count code points
  - Octets starting with 0 or 11 are start of character
  - Octets starting with 10 are continuation characters
  - Self synchronizing
- Encodings for UTF-16 use surrogate pairs
  - Surrogate pairs can store 20 bits of data
  - Define plane 0 to not use surrogate pairs and this gives 21
- Evolving over the last 200 years …

A brief history of Unicode

🏁 happens

Alex Blewitt
@alblue