Lecture 2
Data representation

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

Finite length binary numbers

- CdM-8 memory cells and registers have 8 bits
- They cannot represent arbitrary numbers
- Maximal unsigned number is 255
  - (we will discuss signed numbers later today)
- 255+1=0.
  - Actually, no. 255+1=0+carry bit
- This is very different from arithmetic you study in Calculus

32- and 64-bit computer are also finite

- Maximal unsigned 16-bit number is 65535
- Maximal unsigned 32-bit number is approximately 4 000 000 000
- Maximal unsigned 64-bit number is approximately 16 E18
- How to estimate this?

How to estimate big powers of two?

- Powers of two from 1 to 10 are easy to remember
  - And I think every IT specialist must remember them
- Powers from 1 to 6 are school multiplication table. 2**6=8**2=64
- 2**8 = 256 (maximal unsigned byte value+1) - useful to remember
- 2**10=1024 (approximately thousand) – also useful and easy to remember
- You can remember values of 2**9 and 2**7 or calculate them as needed
- 2**32=(2**2)*(2**30)=4*((2**10)**3)

  ~=4*(1000**3)=4 000 000 000
- 2**64~=(2**4)*(1000**6)=16*1E18

Megabytes, kilobytes, etc

- 2**10 bytes = 1024 bytes = 1 kilobyte
- Normal people think kilobyte has 1000 bytes, programmers think kilogram has 1024 grams
- 1024 kilobytes = 1Megabyte ~= 1 000 000 bytes
- 1024 Megabytes = 1 Gigabyte
- 1024 Gigabytes = 1 Terabyte
- Some bad people (like HDD makers) use decimal Mega.. Giga and Tera prefixes instead of binary. Sometimes they designate this by using Tib instead of Tb. Sometimes they not. Beware

Can we work with long numbers on CdM8?

- Yes we can.
- There is a carry bit in CdM8 PS register.
- I mentioned it when we discussed what 255+1 means.
- 255+1=127+128=...=0+C bit
- In most modern CPUs it can be used for branch conditions
- adc instruction, which adds two registers and a carry bit
- You can use it to implement an arbitrary length integer calculation
  - Well, not bigger than 8096 bits

What about negative numbers?

?

Simple idea: sign bit

- Use high bit to represent a sign
- 24=00011000, -24=10011000
- Aka signed magnitude or sign-and-magnitude
- Was popular in early computers
- Biggest number is 127, smallest number is -127
- Two representations of 0: 00000000 and 10000000
- We will understand later why it got out of fashion

More complex idea: two complement

- To calculate the 2's complement of an integer,
- Pad it to given word length (8 bits in CdM-8)
- Invert all bits
  by changing all of the ones to zeroes
  and all of the zeroes to ones
  (also called **1's complement**),
- And then add one.
- 2’complement of 1 is ^(00000001)+1=11111110+1=11111111
- Why?

Really, why use 2’complement?

- 1+2’complement(1)=0
- N+2’complement(N)=0 for all -1<N<128
- K+2’complement (N)=K-N for most N and K fitting in 7 bits
- So, if we treat 2’complement(N) as –N, we can add negative numbers as we do with positive (unsigned)
- Great simplification of hardware
- No -0 value
- Extra value for negative numbers (smallest possible is -128)

$$5 + (-3) = 2$$

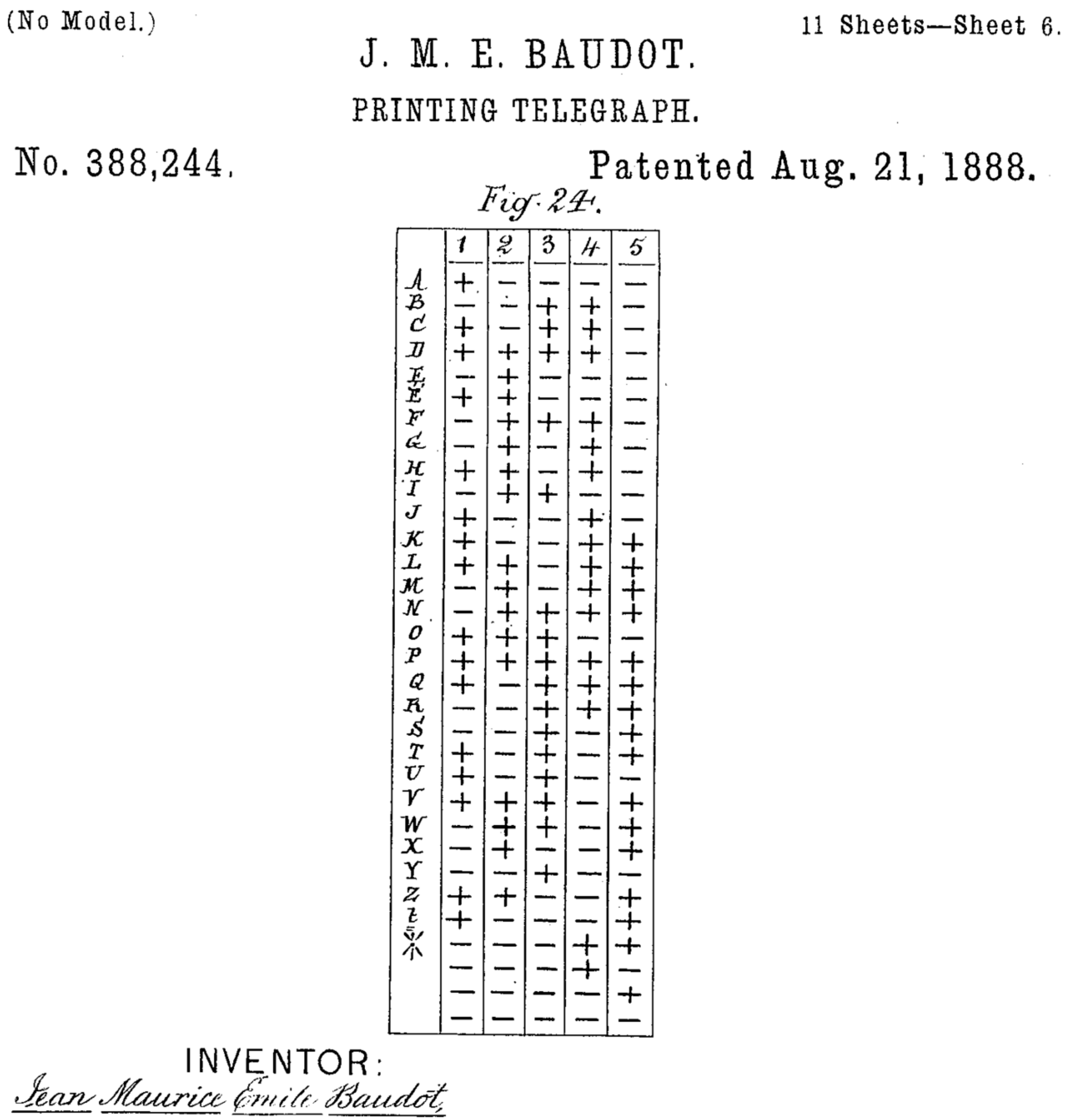
```
  0000 0101 = +5
+ 1111 1101 = -3
  ---------
  0000 0010 = +2
```

So, what is exact value of N and V flags in PS?

- For most CdM-8 commands, Z is 1 iff the operation result is 00000000
- N flag is = topmost bit of the result (bit 8)
- C flag is equal to carry to bit 9
- V flag is 1 if you added two positive 2’complement numbers and got negative
  or if you added two negative 2’complement numbers and got positive
- It is known as *overflow* or *sign loss*
- Or this can be expressed in other way:
  carry to bit 9 is **not** equal to carry to bit 10.
- V is for oVerflow. In some other CPUs it is called O.

Text representation

- First widely used binary communication system was Baudot printing telegraph
- Baudot code was used until 1924 when if was superseded by 6-bit ITA2 encoding
- Modulation unit (Baud) is named after Baudot


(No Model.)

11 Sheets—Sheet 6.

J. M. E. BAUDOT.

PRINTING TELEGRAPH.

No. 388,244.

Patented Aug. 21, 1888.

Fig. 24.

	1	2	3	4	5
A	+	−	−	−	−
B	−	−	+	+	−
C	+	−	+	+	−
D	+	+	+	+	−
E	−	+	−	−	−
É	+	+	−	−	−
F	−	+	+	+	−
G	−	+	−	+	−
H	+	+	−	+	−
I	−	+	+	−	−
J	+	−	−	+	−
K	+	−	−	+	+
L	+	+	−	+	+
M	−	+	−	+	+
N	−	+	+	+	+
O	+	+	+	−	−
P	+	+	+	+	+
Q	+	−	+	+	+
R	−	−	+	+	+
S	−	−	+	−	+
T	+	−	+	−	+
U	+	−	+	−	−
V	+	+	+	−	+
W	−	+	+	−	+
X	−	+	−	−	+
Y	−	−	+	−	−
Z	+	+	−	−	+
t	+	−	−	−	+
[illegible]	−	−	−	+	+
	−	−	−	+	−
	−	−	−	−	+
	−	−	−	−	−

INVENTOR:

Jean Maurice Émile Baudot

American Standard Code for Information Interchange

- ASCII
- 7-bit code standardized by American Standard Association (now ANSI) in 1963.
- Latin encoding used by most modern computers
- Had 8-bit extensions to support national scripting systems (European characters, Greek, Cyrillic, Hebrew)

ASCII table

ASCII Code Chart

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

Unicode

- Designed to represent all writing systems known to humanity
- Including historical, like Egyptian hierogliphs
- Including fictionary, like Klingon and Quenya
- Several translation formats, including UTF-32, UTF-16 and UTF-8
- UTF-32 can represent any Unicode codepoint directly
- UTF-16 uses so called ”surrogate pairs” to represent some characters
- UTF-8 – ASCII-compatible prefix encoding

UTF-8

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx