

①

Přenosová rychlost je počet datových bitů za jednotku času.

1 baud je rovněž jednotka přenosu informace za jednotku času, tedy i včetně start/stop bits.

$$\frac{176}{8} = 22$$

Overhead linky činí $1 - \frac{8}{176}$

Pobud má linku propustnost 16 Mb/s datových, potřebuje $16 \cdot \frac{176}{8}$ Mb/s
Celkově tedy propustnost 22 Mb/s

Pobud budeme mít 32 datových, je overhead pouze $1 - \frac{32}{36}$

na přenosovou rychlost 22 Mb/s bude potřebovat $22 \cdot \frac{36}{32}$ Mb/s ≈ 24 Mb/s

Obecně platí, že čím menší overhead, tím méně musí být nadimenzovaná linka.

$\frac{2}{3} - 35 \approx 24$
opět přibližně
nemám kalkulaci

Pobud obnáší 1 Mb ≈ 1024 kb $\approx 1024 \cdot 1024$ b, takže rychlost v baudech

u prvního potřebujeme ≈ 22 000 000 baud

u druhého potřebujeme ≈ 24 000 000 baudů.

\\
nemám kalkulaci
a možná ztratit
čas

Což reflektuje skutečnost, že čím menší overhead, tím méně se musí linka předimenzovat.

Nonkrátké označení Master a slave (doch hold low etc.) je definováno protokolem komunikace.

⑤

Master a Slave je důležitá charakteristika na datové sběrnici. Aby byl provoz kontrolovatelný a usměrněný, vždy se dořizuje, s jakým cílem!! přenosem, kdo je aktuálně Master a kdo Slave.

při každém přenosu samozřejmě ne.

Existují multimaster i singlemaster sběrnice - tedy zde Master může být jen jedno zařízení či více zařízení.

Master je ten, kterým pro konkrétní přenos zadává pořadí (řídí komunikaci) a slave je ten, kterým odpovídá ves. vyhledává příkaz.

Typickým příkladem Masterem je procesor, kterým svůjm slovům, např.: registrujme definice, data ze kterých adres mu mají vrátit.

Obecně však platí, že pokud jde o data od mastera, je to write, pokud od slava k masterovi, je to read.

IEEE 754

⑥ - 1565, 625 uložit do paměti ve $0x000FA00$ jako single (32-bit float)

IEEE 754 float: 23 bitů (+ skrytá jednotka)
8 bitů exponent (bias [+127])
1 bit sign bit

1565 = 1100 0010011

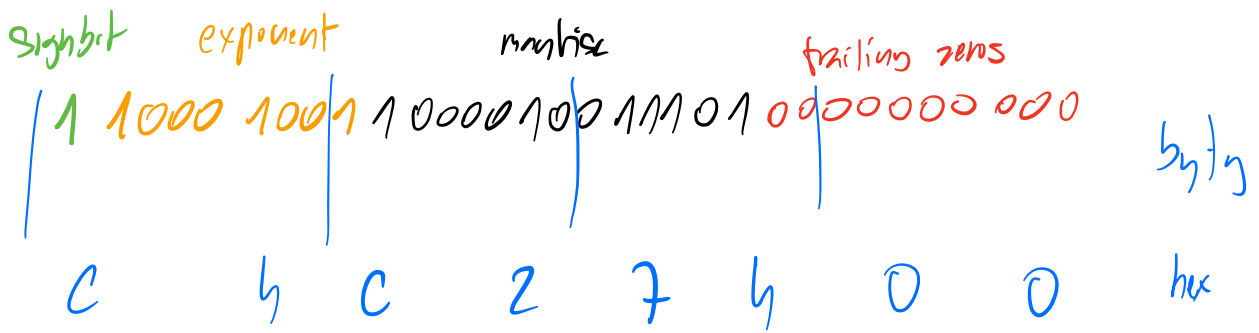
0,625 = 101

11000010011 101 00000000

skrytá jednotka

exp = 10

\downarrow bias + 127 $\rightarrow 137_{10} = 1000 1001_2$



0xChC27400

Takže postupně na adresu 0x0000FA00 a dále: 00 7h C2 C4
 - little endian (LSB first)

7) ŠEMÍK?

0x00001F02 addr.

- a) Windows 1250
- b) UTF-16 LE

S	=	UNI	WIN
E	=	0160	8A
M	=	0045	45
I	=	004D	4D
K	=	00CD	CD
?	=	004B	4B
.	=	003F	3F

LE!
 ←

UNI : 0160 0045 004D 00CD 004B 003F

WIN : 8A 45 4D CD 4B 3F

Hex dump ONI:

0x02001F02 | 3F 00 4B 00 CD 00 4B 00 | 45 00 60 01 00 00 00 00

Hex dump WIN:

0x02001F02 | 8A 45 40 CD 4B 3F 00 00 | 00 00 00 00 00 00 00 00

③ Zapište posloupnost bez ACK!

- 8 data bits, 1 ACK bit, Negative values in Two's Comp.

Addr: W = 42₁₆ R = 43_{hex} (7 bit) - LSB R/w

- 9th SCL pulse - ACK

Master to master X offset = 0x01 (MSB)

0x02 (LSB) (MSB first writing)

Musím poslat: 1 byte (address + R/w)

1 byte (command)

1 byte (address)

2 bytes (data)

2255₁₀ = 0000 1000 | 1100 1111₂
0 8 | C F

08 CF

Data:

□ 42 77 01 08 □

□ 42 77 02 CF □

↑
start condition

↑
address slave

↑ bez ACK
command

↑
address
in EkPROM

↑
data

↑
stop condition

h) (tabulka - page 6.)

```
def SetNewOpModeRate(opMode, newRate)
```

```
rates = { (1, uint8(0)), (5, uint8(1)), (10, uint8(2)), (20, uint8(3)) }
```

nejsun si jisti, jaky zivakny, oddelaji, dik v dictionary

```
check = opMode # Abych si nezmenil moje data
```

```
if check << 6 != uint(127): # When not in query mode
```

```
if check << 6 == uint(0) OR check << 6 == uint(256):
```

```
# When in StandBy or Continuous Mode
```

```
opMode = opMode & uint(159) # clearing
```

```
opMode = opMode | rates[newRate]
```

```
# setting up new rate from dict.
```

```
if check << 6 == 0:
```

```
# Set from StandBy to Continuous
```

```
opMode = opMode | uint(2) # setting mode
```

```
return opMode
```

check << 6 ?
 ?? 000000
 0 00 - StandBy
 127 01 - Query Mode
 256 10 - Continuous
 11 - Not Allowed

clearing rate bits to zero:

```

      ? 0 1 ? ? ? ?
mask: 1 0 0 1 1 1 1 1  &
      ? 0 0 ? ? ? ?
      ^
      159
  
```

setting Mode

```

      ? ? ? ? ? ? 0 0
mask: 0 0 0 0 0 0 1 0  OR
      ^
      2
  
```

8) $C = C + d + d + 25h$

16b! $C = 0x7000$

16b! $d = 0x7002$

proměnná v

$0x0000 - 0x00FF$

1) $var0 = d + d$

2) $C = C + var0$

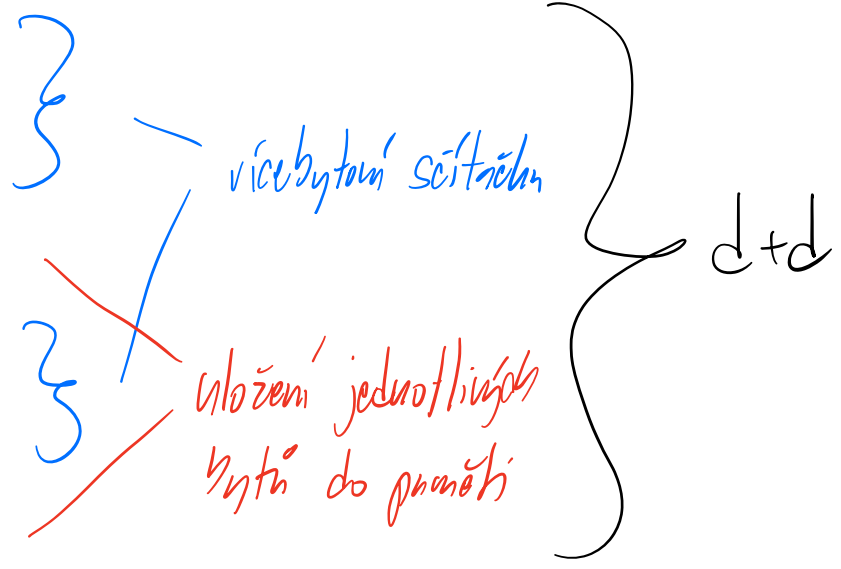
3) $C = C + 25h$

$25h = 0xFF$

$var0 = 0x0000$ (2d)

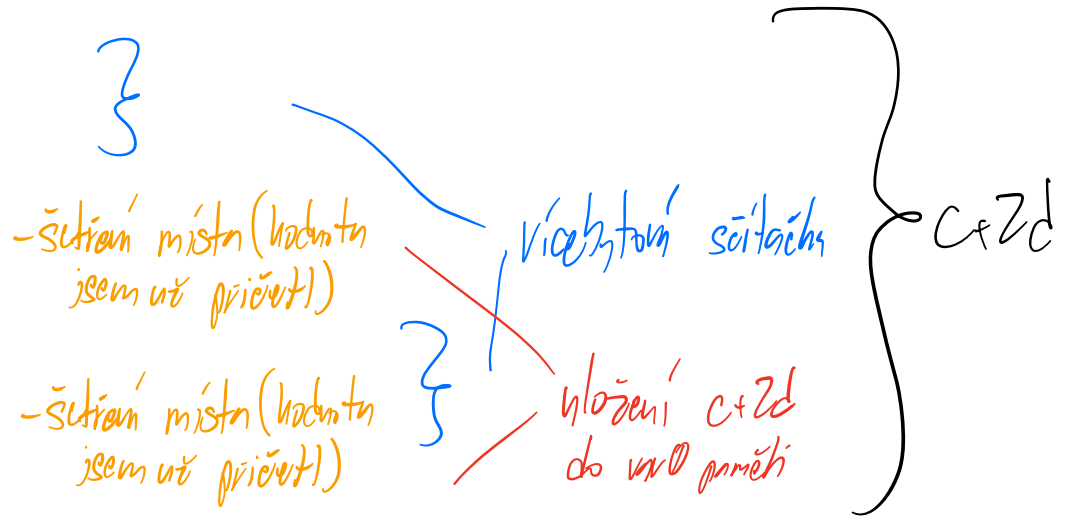
```

1) LDAA $7002
   CLC
   ADCA $7002
   STAA $0000
   LDAA $7003
   ADCA $7003
   STAA $0001
  
```



```

2) LDAA $7000
   CLC
   ADCA $0000
   STAA $0000
   LDAA $7001
   ADCA $0001
   STAA $0001
  
```



```

3) LDAA $0000
   CLC
   ADCA #$FF
   STAA $7000
   LDAA $0001
   ADCA #$00
   STAA $7001

```

více bytoví carry sítěch s 00FF
 uložení zpět do „C“
 pro případný carry přenos

⑨ offset = 80 00 00 00 = 00000080 = 128₁₀ → 0x000080

PE: 5D 45 00 00 → znak 20 bytů (velikost COFF Header),
 znak 4 byty offset a následující 4 byty „SizeOfCode“
 Signature

SizeOfCode: 00 00 08 00 = 8 · 16² = 2048

⑩ def IsPeExecutable (filename: str)

file = open(filename, "rb")

mz_sign = file.read(2)

→ první dva byty nás zajímají

if mz_sign[0] != 0x4D or mz_sign[1] != 0x5A:

return False

else:

size_of_file = os.path.getsize(filename)

if size_of_file > 0x3C:

file.seek(0x3C)

pe_sign_offset = file.read(2)

↳ vyhledání z hexdump souboru v příloze

if `pe_sign_offset + 0x30 < size_of_file`:

`file.seek(pe_sign_offset)`

`pe_sign = file.read(2)`

if `pe_sign[0] == 0x50` or `pe_sign[1] == 0x65`:

return True

else:

return False

else:

return False

else:

return False