# O strojovém kódu.

Vladimír Vérosta, *příspěvek pro konferenci Technického muzea v Brně*

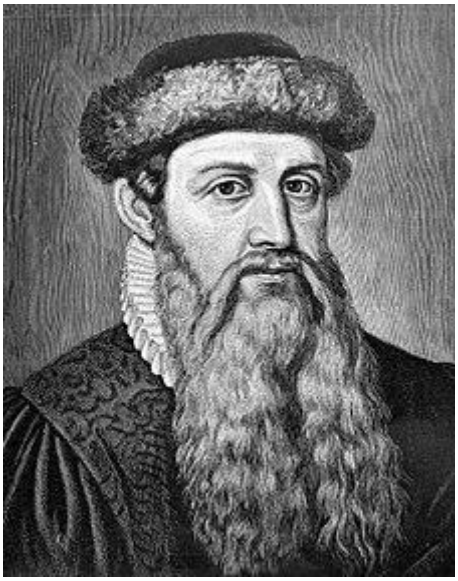
*"Od strojového kódu k programování neuronových sítí"*
*v září 2018 a pro web [prog-story.technicalmuseum.cz](prog-story.technicalmuseum.cz)*

Dovolím si nejprve krátce připomenout život a dílo lidí, kteří nepochybně velmi přispěli k soudobému stavu vzdělanosti, vědy a tím pádem i naší oblíbené výpočetní techniky!

Johannes Gensfleisch, řečený **Gutenberg**, nar. 1394 v Mohuči (+1468), zlatník, brusič drahokamů, výrobce puncovních razidel, přišel na myšlenku odlévání písmen z kovu, vynalezl knihtisk!

Tzv. 42řádková Gutenberghova bible o 1282 stranách byla vytištěna v nákladu 180 exemplářů, z toho 30 luxusnějších na pergamenu, ostatní na papíru. Ač jde o teprve první velké dílo knihtisku, je technicky dokonalé, s harmonickou sazbou, zarovnanými pravými okraji sloupců a stejnoměrně sytou černí tisku.

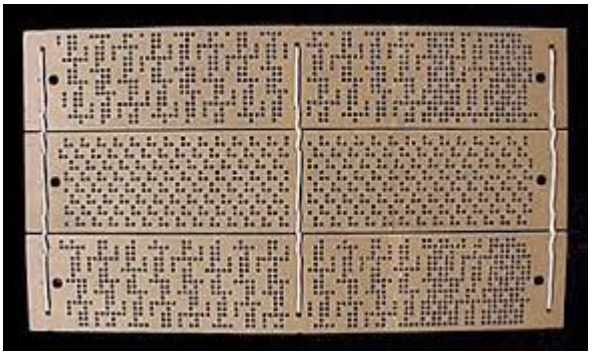
Ve velkém počtu a za nízkou, dostupnou cenu průmyslově vyráběné knihy významně pozvedly úroveň vzdělávání, vědy, kultury, význam nedozírný. Informační revoluce.



Gutenbergova Bible z roku 1454/55 o 42 řádcích

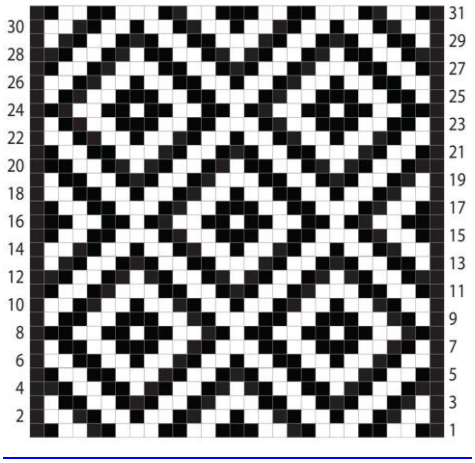
https://wikisofia.cz/wiki/Joseph_Marie_Jacquard

**Joseph Marie Jacquard** vlastním jménem Joseph Marie Charles (* 7. července 1752 Lyon, Francie, † 7. srpna 1834 Oullins, Francie). Jacquard byl francouzský vynálezce, který **sestrojil první programovatelný tkalcovský stav**, který se programoval pomocí děrných štítků



Otvor ve štítku znamenal nazvednutí určité nitě a tím postupně vytváření složitých vzorů na tkané látce. Štítky byly spojovány do dlouhých pásů a jejich výrobou, programováním se zabývali první developeři!! 😊


Vzorování na látkách vyrobených na žakárových strojích už opravdu stálo za to!!



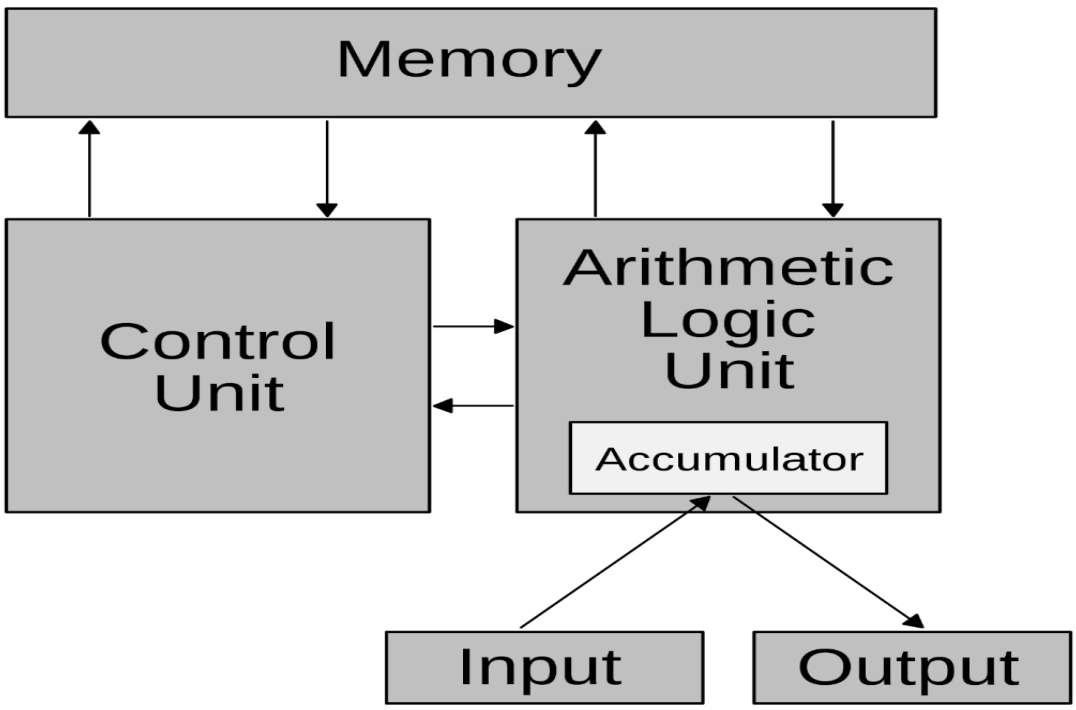

https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/VNEUM.HTML


Za třetím mužem mého přehledu se vydáme do jedné temné prosincové noci roku 1903, na březích Dunaje v krásné Budapešti, se 28. prosince narodilo dítě mužského pohlaví, kterému rodiče při křtu dali dvě jména: János a Lájos (čti Jánoš, Lájoš) Chlapeček velmi prospíval byl nesmírně nadaným studentem a když se později dostal do USA, anglická podoba jeho jména zněla

## John von Neumann!

V roce 1945 navrhl strukturu moderního počítače, která je v podstatě platná dodnes. Jánošovi připadalo, že počítače podle jeho architektury nebudou mít pro svět valného významu, tehdy to totiž byly rozměrné skříně ve velkých sálech; mýlil se, kdykoliv uchopíte svůj mobil, vzpomeňte na tohoto muže, protože bez něj bychom technologicky nebyli tam , kde jsme!

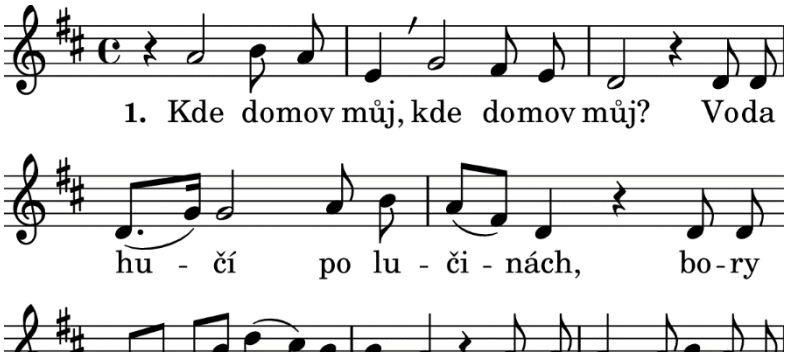
Von Neumannovo schéma:



Zatímco u Jacquardova  stroje jsou děrné štítky, obsahující řídící vzor textilie, zřetelně uloženy vně stroje, John pochopil, jak to má být správně a program, řídící výpočetní procesy, uložil do vnitřní paměti počítače. Ano tedy tam, kde jsou uložena i zpracovávaná data. Nárůst výkonnosti systému přímo astronomický!

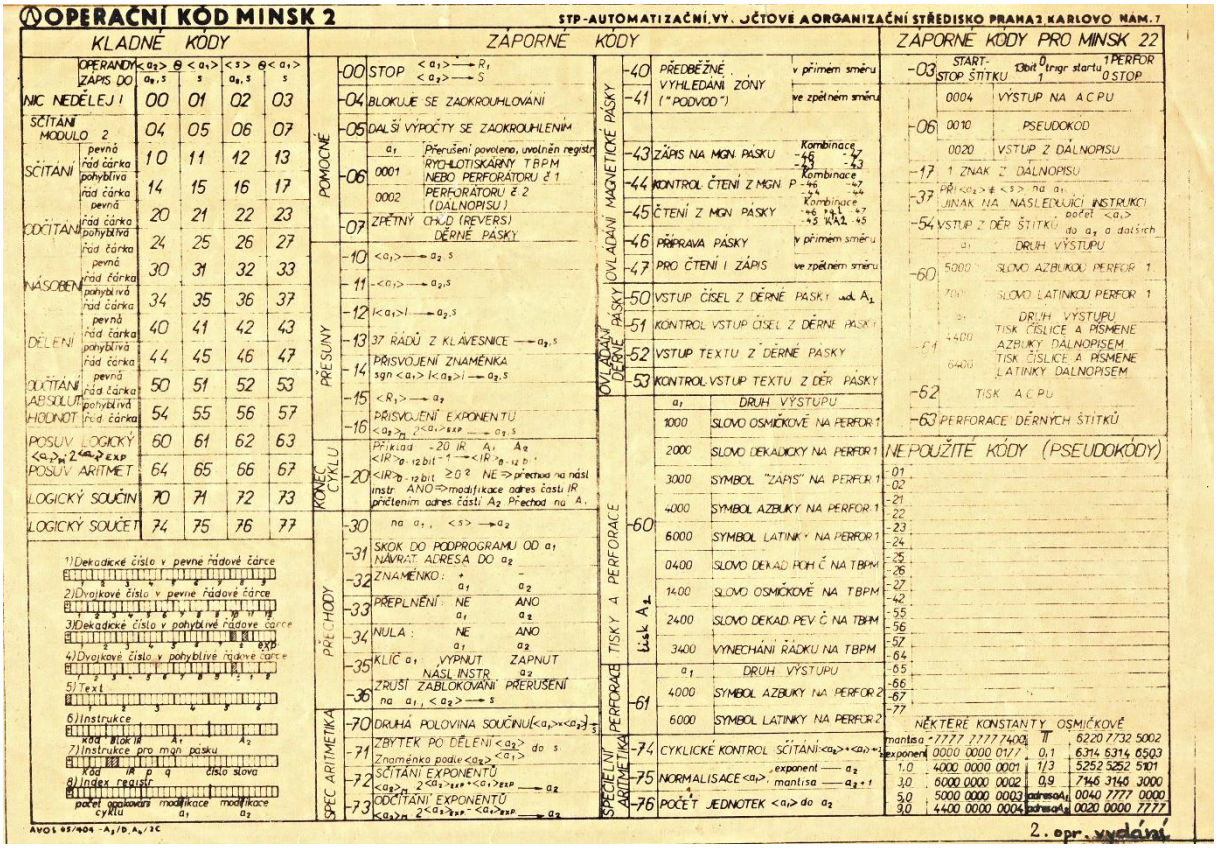
A takhle prakticky vypadají všechny moderní počítače, včetně mobilních telefonů.

Řídicí jednotka počítače (control unit) čte postupně jednu po druhé instrukce uložené v paměti a provádí je, jinak řečeno program se interpretuje. Postupuje tedy úplně stejně jako v případě, kdy je interpretace předepsána jiným způsobem:




Notový zápis je taky určen pro interpretaci, a opakovací značky v něm vytvářejí něco jako cyklus v programu. 😊

Při prvních pokusech na počítači Minsk-2 jsme ovšem strojový kód zapisovali do paměti počítače přímo z ovládacího pultu, kde byla klávesnice pro čistě binární vstup.



Instrukce strojového kódu se dají dělit podle jejich určení do různých skupin,

Přesuny slov/půlslov mezi pamětí a registry, oběma směry

Přesuny obsahů paměti z jednoho místa na druhé,

Operace pro provádění logických operací, logický součet, součin, …

Testování podmínkového kódu, jakožto produktu vznikajícího po provedení nějaké instrukce

A na základě jeho hodnoty větvení programu,

Operace pro výpočty s pohyblivou řádovou čárkou.

Firma **IBM** byla proslulá svou precizností a svoje sálové, tedy mainframe stroje doprovodila vyčerpávající kultovní příručkou „Principles of operation“ se systematickým popisem počítačového systému včetně strojového kódu. Po jistou dlouho dobu to byla náplň mých pracovních dní.


Práce přímo ve strojovém kódu, byť v „pohodlném“ hexadecimálním vyjádření je velmi obtížná, jakákoliv změna kódu, vložení nové instrukce, změna stávající znamená ruční práci na přebudování celého „textu“, a lze ji akceptovat jedině u velmi, velmi omezeného okruhu úloh, kde jiná varianta programování není možná, třeba IPL z děrné pásky.

Pro rozumnou práci většího rozsahu se proto používá vyšší, symbolický stupeň vyjádření jednotlivých instrukcí strojového programu, tzv. **Assembler**. Efektivita práce vývojáře tím mnohonásobně vzroste:

Např. jednoduchý přesun znakového řetězce na počítačích IBM 360/370 může například vypadat v Assembleru takto:

**MVC JMENO(22), VSTUP**

V hexadecimálním zápisu ovšem např. jako

**D215B0179F37**

A při binárním pohledu je to změť nul a jedniček dosti nečitelná:

**1101 0010 0001 0101 1011 0000 0001 0111 1001 1111 0011 0111**

(Hmm, ta dvojakost nuly a jedničky vzbuzuje asociace na prastaré východní koncepty jin a jang, ze kterých jedině prý se skládá tento svět! Tohle bych bral jako důkaz! 😊 )

Strojové instrukce systémů IBM se podle účelu dělí do skupin:

… hlavní instrukce

… dekadické instrukce

… instrukce v pohyblivé řádové čárce

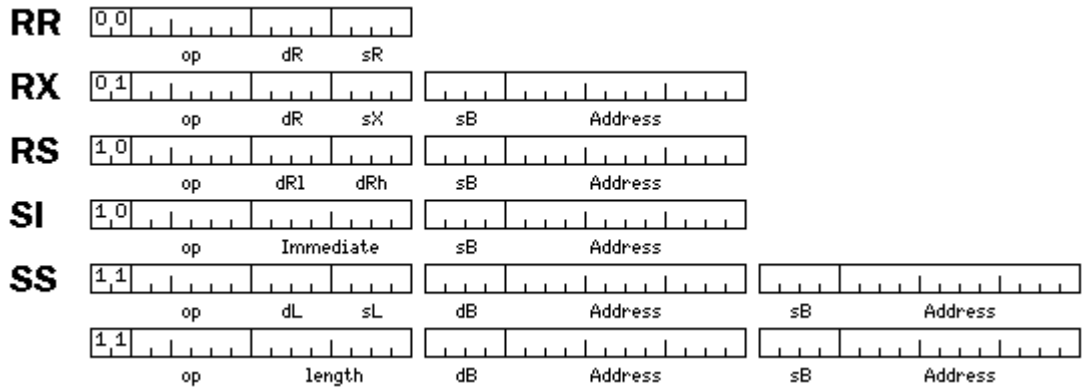
… řídící instrukce

… instrukce vstupu/výstupu

Podle formátu se pak instrukce dělí na typy RR (délka 2 bajty), RS (4 bajty), RX (4 bajty), SI (4bajty) a SS (6 bajtů).

Pomocí nich se manipuluje s obsahem hlavní paměti, každým ze šestnácti univerzálních registrů, s registry pro pohyblivou řádovou čárku, testuje se tzv. podmínkový kód signalizující, jak dopadla bezprostředně předcházející operace.

K dispozici je ovšem i sada čistě systémových instrukcí, určených pro řídící programy operačního systému, supervizor a jeho příbuzenstvo.




Adresování operandů

Pokud je operand právě prováděné instrukce uložen *v univerzálním registru*, tak je v instrukci na jeho

„adresu“ určeno čtyřbitové pole, kam se právě tak dají uložit čísla 0, 1, 2 až 15 znamenající číslo dotyčného registru.

Například instrukce  LR 15,11 přesouvá obsah registru 11 do registru 15, formát instrukce hexadecimálně potom zní

**18FB**

binárně 0001 1000 1111 1011 😊

Číslo 18 je kód instrukce LR (na 8 bitů), to F je adresa cílového registru 15, B pak adresa zdrojového registru 11 (obě adresy jsou po 4 bitech) čili 8 + 4 + 4 = 16, tedy tahle instrukce zabere v paměti počítače přesně 16 bitů.

O dost složitější je princip *adresování paměťových operandů*. Přiznám se, že mi trvalo delší dobu, než jsem ho vstřebal natolik a byl schopen v jeho rámci uvažovat, smysluplně a efektivně použít.

Potíž je v tom, že paměť už tehdy měla řádový rozsah několik megabajtů, absolutní adresa operandu sice byla známá, určitelná, ale její zápis by vyžadoval třeba 24 bitů paměti. A to by ve strojovém kódu zabralo příliš velký prostor, nehledě na další nevýhody.

Adresa operandu v hlavní paměti se proto (na strojích ibm 360 a novějších) konstruuje jako 16ti-bitové pole v instrukci. Má formát

**BAAA**

Kde B (4 bity), označuje číslo tzv. bázického registru (1 – 15, nulu nelze použít)

AAA (12 bitů) je pak kladná relativní adresa vzhledem k této bázi.

Adresa operandu v hlavní paměti se musí při prováděné kódu vždy znovu a znovu vypočítávat jako součet obsahu univerzálního registru B a daného 12-ti bitového offsetu.

Pokud tedy například budeme mít v univerzálním registru 7 hodnotu 00285BC4, tak adresa operandu v instrukci tvaru **741A** povede k výpočtu 00285BC4 + 41A = **00285FDE,** tohle je tedy výsledná adresa operandu v hlavní paměti (virtuální paměť nechme nyní spát!)

Offset na 12 bitů ovšem nedá moc nadechnout, protože na 12 bitů jsme schopni zapsat maximálně 4096 čísel (000 až FFF), tedy píseček na hraní v rámci jedné hodnoty bázického registru je poměrně malý.

Instrukce MVC (kód D2) pro přesun 27 bajtů (27 = 1B hex.) z jedné části paměti do druhé, v rámci jednoho bázického registru může například vypadat takto:

**D2 1A 741A 7A22**

Pozor, údaj o délce přesouvaného znakového řetězce je v instrukci kódován číslem o jedničku menším, tedy místo 1B je tam uvedeno 1A!

Je ovšem možné mít připraveno více bázických registrů, pomocí kterých jsme schopni operovat ve větším rozsahu paměti, pak by instrukce MVC mohla vypadat třeba takto:

**D2 1A 741A 322E**

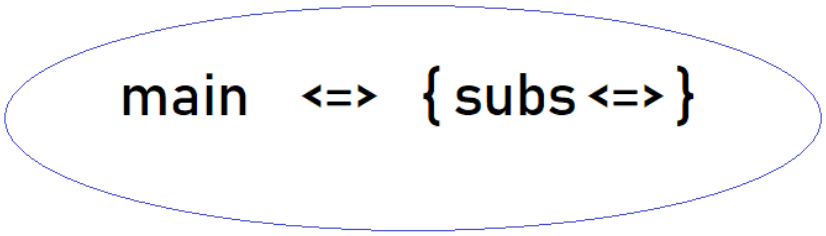
Adresa cílového operandu je pořád 741A, pro zdrojový operand jsme ovšem, lišáci, použili adresu 322E s jiným bázickým registrem. Muselo to tak být, protože tento operand zřejmě leží hoooodně daleko od cílového, takže jeden bázický registr nestačí.

Třídy, instance??

V té době samozřejmě nikdo neměl ani potuchy o paradigmatech moderního programování, teď ovšem při pohledu zpět vidíme, že bázický registr jakoby reprezentuje ukazatel na instanci nějakého objektu (o max. velikost 4096 bajtů ovšem), takže ta druhá instrukce MVC z dnešního pohledu reprezentuje přesun hodnoty z jednoho objektu do druhého, resp. mezi dvěma instancemi jedné a téže třídy. 😊

Struktura aplikace

V assembleru, tedy už na symbolické úrovni, vypadá struktura aplikace celkem jednoduše. Hlavní program, main, má k dispozici řadu podprogramů, subs, které realizují příslušné části algoritmu, main volá subs, případně jednotlivé subs se volají mezi sebou.




Vyvolání a návrat z podprogramu se děje na půdorysu dobře navrženého schématu, kde rozhodující roli hrají univerzální registry.

Registr 15 – báze volaného podprogramu,

Registr 14 – návratová adresa z volání,

Registr 1 – ukazatel na pole případných argumentů volání,

Registr 13 – adresa úklidového pole volatele pro uložení aktuálních hodnot registrů volatele

```
USING *,15
STM 14,12,8(13)
..... instrukce podprogramu
.....
LM 14,12,8(13)
BR 14
```

Ladění

Dnes, např. na úrovni jazyka C# nebo Java, je syntaxe a sémantika navržena tak dokonale, že jen velmi zřídka nastane případ, kdy by program nefungoval nebo dokonce havaroval. Mít možnost pracovat na projektech v jazyce C# pokládám proto za odměnu!

Havárie programu psaného v assembleru byly ovšem na denním pořádku. Nejčastějšími případy byla chybná hodnota bázického registru, nejčastěji touto chybnou hodnotou byla nula, při provádění instrukce se pak CPU pokusil pracovat s pamětí o adrese jenom o něco větší než ta nula, což vedlo k přerušení z důvodu ochrany paměti a program šel do kytek, havaroval.

Hlavním pomocníkem developera při analýze příčiny chyby byl tzv. velký dump neboli výpis obsahu paměti v okamžiku havárie. Dala se z něj adresa programu, kde došlo k chybě, byly tam vypsány obsahy počítačových registrů a pak obsah uživatelské části paměti, kde byl uložen program. Dump se tiskl na široké tiskárně (120 nebo později i 132 nebo 160 znaků široké), a těch stránek výpisu bývalo i několik desítek. Lesy plakaly.

Po nalezení příčiny programátor příslušným způsobem opravil text programu a spustil další kolo ladění.

Něco takového jako interaktivní ladění bylo natolik vzdálené našim fantaziím, že jsme po něm ani nezatoužili!

Paměti

Paměti, které bývaly k dispozici pro provozování programů jsou z dnešního hlediska naprosto komické.

Vnitřní paměť luxusního provedení počítače EC 1021 měla celých 64 (slovy šedesátčtyři) kB, přičemž 16 kB si zabral operační systém a pro aplikační úlohy tedy zbývalo pouhých 48 kB!

Paměť RAM mého zcela konfekčního, obyčejného telefonu má 2 GB. Zkusme tedy podělit 2 000 000 000 / 65 536 = 30 517! Nemohu tomu uvěřit, paměť mého telefonu by dokázala pokrýt paměťovou potřebu 30.517 strojů EC 1021!!

Diskové paměti u strojů EC 1021 měly kapacitu 7,25 MB a bývaly tam 4 diskové jednotky tohoto typu, které poskytovaly ohromující kapacitu 4 x 7,25 = 29 MB!

Dnes mívají běžné fotoaparáty snímače, které produkují snímky o velikosti tak asi 3,5 Mb, čili na tamten disk by se vešly asi 2 obyčejné fotky! Dnešní běžná paměťová SD karta má kapacitu 16GB, což po vydělení 16GB / 29MB ~= 551! Neboli jedna karta do foťáku má dnes paměť jako tehdejších 551 sálových počítačů.

Technologický vývoj udělal od těch dob neuvěřitelný pokrok, dobře tomu tak, jenom mám podezření, že příčinou tohoto pokroku není člověk, ale …. čert!!! 😉