PROGRAM NEJSOU ŽÁDNÉ ČÁRY

(aneb převod dekadických čísel na binární a naopak)

Program, programování, počítače, mikroprocesory, to jsou všechno v poslední době velmi frekventovaná slova v našem životě. Běžně je používáme a považujeme to, co reprezentují, za samostatný obor, který zvládá ten, kdo se v něm „vyučil" a my ostatní smrtelníci samozřejmě nikoli. I když při tom základ programování – logiku, schopnost logicky uvažovat a řadit fakta a skutečnosti – máme přece každý vrozenou. A tak když vezmeme rozum do hrsti a seznámíme se s tím, jak zacházet s kalkulátorem, který máme k dispozici, může programovat každý z nás.

Co mě přimělo k této úvaze. Jsa poměrně čerstvým držitelem programovatelného kalkulátoru TI-58, učím se ve volných chvílích s ním zacházet. Nejlépe tak, že zatím „prověřuji" programy, které vymyslel někdo jiný. A tak se mi dostal do ruky program pro převod dekadických čísel na binární, což je jistě užitečná pomůcka. Byl navržen pro kalkulátor TI-57, který je sice jednodušší než TI-58, ale má vzhledem k TI-58 i některé přednosti. Program (který je rovněž uveřejněn v této ročence na str. 62) má 42 kroků (na TI-58 to dalo 58 kroků, protože nemá tzv. sdružené instrukce) a výsledky dával „po kouskách" – zmáčkne se tlačítko, objeví se číslice, zmáčkne se tlačítko, objeví se číslice, a to ještě ne číslice výsledku, ale číslice udávající, na kterém místě výsledku má být „jednička".

Nelíbilo se mi, že kalkulátor neukáže najednou celý výsledek a že je zapotřebí pořád něco „mačkat". Řekl jsem si, že program udělám jinak.

Vyšel jsem ze základního postupu, když převádíme dekadické číslo na binární s tužkou a papírem, bez elektroniky. Postupujeme asi takto (např. číslo 156):

1.	156:2 = 78, zbytek 0, píšeme	tedy	0	**0**
2.	78:2 = 39, zbytek 0, píšeme		0	**00**
3.	39:2 = 19, zbytek 1, píšeme		1	**100**
4.	19:2 = 9, zbytek 1, píšeme		1	**1100**
5.	9:2 = 4, zbytek 1, píšeme		1	**11100**
6.	4:2 = 2, zbytek 0, píšeme		0	**011100**
7.	2:2 = 1, zbytek 0, píšeme		0	**0011100**
8.	1:2 = 0, zbytek 1, píšeme		1	**10011100**

To samé teď musím naučit kalkulátor, včetně střádání výsledku tak, aby se na konci celý objevil na displeji.

Nejdříve mě napadlo, jak „střádat" celý výsledek. Když se podíváte, jak „přibýval" výsledek při ručním výpočtu (poslední sloupec vpravo), je to vlastně totéž, jako kdybychom k předchozímu mezivýsledku přičetli (dekadicky) vždy buď 10, 100, 1000 atd., podle toho, u kterého řádu jsme a je-li napsaná jednička či nikoli. Celý výsledek tedy dostaneme jako $\Sigma 10^{n-1}$ z těch míst binárního čísla, kde jsou jedničky. Číslo 10^{n-1} dostaneme tak, že vezmeme číslo 0,1 a před každým krokem výpočtu (před každým řádkem v ručním výpočtu) ho vynásobíme deseti. Pro první řádek potom dostaneme 1, pro druhý 10, pro třetí 100 atd. Toto číslo přičteme k předchozímu mezivýsledku, je-li zbytek dělení nenulový.

A teď jak upravit postup převodu čísel pro kalkulátor. Musí rozlišit dělení beze zbytku a dělení se zbytkem. Výsledkem dělení beze zbytku bude celé číslo, výsledkem dělení se zbytkem bude celé číslo a pět desetin (pětka za desetinnou tečkou). Část čísla za destinnou tečkou je tedy buď rovna nule a nebo ne, a to kalkulátor „umí" zjistit testovacím tlačítkem **x=t**, když (obsah registru) t = 0.

Postup bude tedy následující:

Obsah paměti, do které jsem vložil 0,1 (pro vytváření čísla 10^{n-1}) vynásobím deseti. Zadané číslo vydělím dvěma a nechám kalkulátor zjistit, zda je výsledek celé číslo či nikoli. Je-li výsledek celé číslo, vrátím se na začátek výpočtu, vynásobím deseti obsah paměti pro vytváření čísla 10^{n-1} atd. Není-li výsledek celé číslo, přičtu stávající obsah paměti s číslem 10^{n-1} k dosavadnímu výsledku a rovněž se vrátím zpět na začátek.

Nyní je zapotřebí stanovit, kdy je výpočet ukončen. Podíváme-li se na postup ručního výpočtu, zjistíme, že je to tehdy, když celá část výsledku dělení je rovna nule (řádek 8.). Protože výsledek dělení zapisujeme vždy hned do paměti, musíme před následujícím dělením zjistit, zda v paměti není již nula. Není-li, výpočet pokračuje dále, je-li v paměti nula, výpočet končí a na displeji se objeví celkový výsledek „vyzvednutý" z paměti, kam jsme ho po dobu výpočtu střádali.

Teď přeložíme takto logicky sestavený program do řeči a možností programovatelného kalkulátoru:

Zadané číslo uložíme do paměti č. 05:	**STO 05**
Budou se tam postupně ukládat i všechny výsledky dělení, které vždy nahradí předchozí číslo v této paměti.	
Zvolíme paměť č. 01 pro vytváření čísla 10^{n-1} a vložíme do ní (viz výše) číslo 0,1:	**0,1 STO 01**
A začíná výpočet – obsah paměti č. 01 (pro vytváření 10^{n-1}) vynásobíme deseti:	**10 *Prd 01**
Zadané číslo, které je uloženo v paměti č. 05, vyjmeme na displej:	**RCL 05**
Než budeme počítat, musíme zjistit, zda je jeho celá část rovna nule či nikoli. Celou část oddělíme pokynem:	***INT**
Zda je rovna nule, zjistí kalkulátor po stisknutí tlačítka: (za předpokladu, že obsah registru t=0, je to totéž jako x=0).	***x = t**
V případě, že odpověď zní „ano", pokračuje kalkulátor ve výpočtu na čísle programového kroku, zapsaném hned za rozhodovací instrukcí x=t. Výpočet by byl ukončen a bylo by zapotřebí vyvolat z paměti výsledek. Protože zatím nevíme, na kterém programovém kroku tuto operaci budeme mít, vynecháme si „okénko":	**□**
Zjistí-li kalkulátor, že x≠0, přeskočí toto okénko (i když už je v něm číslo) a pokračuje dál. Na displeji máme stále celou část čísla z paměti č. 05 a potřebujeme ji vydělit dvěma:	**: 2 =**
Výsledek si nejdříve uložíme zpět do paměti č. 05:	**STO 05**
a pak zjistíme, je-li výsledek celé číslo. Oddělíme od něj část za desetinnou čárkou:	**INV *INT**
a zjistíme, rovná-li se nule:	***x=t**
Je-li dělení beze zbytku, je odpověď *„ano"*, k výsledku připisujeme nulu a to tedy nemusíme dělat a vrátíme se opět na začátek výpočtu, před operaci 10 Prd 01, na programový krok č. 007	**007**
Není-li dělení beze zbytku, připisujeme k výsledku jedničku a musíme tedy do paměti střádající výsledek (č. 02) přičíst číslo 10^{n-1} z paměti č. 01:	**RCL 01 SUM 02**

a pak se zase vrátíme na začátek výpočtu, na programový krok č. 007, pokynem: **GTO 007**
Výpočet nyní probíhá „kolem dokola" tak dlouho, dokud se nezjistí, že celá část čísla v paměti č. 05 je rovna nule. V tom případě pokračuje program na kroku, pro jehož číslo jsme si nechali prázdné „okénko". Výpočet je ukončen a potřebujeme ukázat na displeji výsledek, střádaný v paměti č. 02. Napíšeme tedy: **RCL 02**
a zastavíme program: **R/S**
Pokyn RCL 02 je na 34. programovém kroku a toto číslo tedy musíme napsat do prázdného okénka. Celý program, seřazený tak, jak „mačkáme tlačítka", je v tab. 1 (označení hvězdičkou znamená, že musí předcházet stisknutí tlačítka 2nd). Pokud bychom chtěli teď počítat jiné číslo, musíme vymazat všechny paměti (aby se nové výsledky nepřičítaly ke starým) a vrátit se na začátek programu. Dopíšeme proto ještě poslední dva pokyny – vymazání pamětí ***CMs**
a *návrat na začátek* programu **RST**

Práce na programu a to, že „fungoval", mě natolik zaujalo, že jsem se rozhodl udělat ještě program pro opačný postup – převod binárního čísla na dekadické. Postup je velmi podobný a vycházel z následující úvahy:

Jednotlivá místa binárního čísla mají (je-li na nich jednička) hodnotu 1, 2, 4, 8. 16, 32, 64, 128 atd., obecně 2^{n-1}, kde *n* je pořadí místa zprava. Součet všech těchto čísel dává celkovou hodnotu odpovídajícího dekadického čísla.

V programu budeme tedy oddělovat po jednom místu zprava ze zadaného čísla a v případě, že na něm bude jednička, přičteme do „střádací paměti výsledku" příslušnou hodnotu 2^{n-1}. Místa budeme oddělovat tak, že číslo dekadicky vydělíme deseti, tím dostaneme jeho první číslici zprava za desetinnou čárku. Část čísla za desetinnou čárkou umí kalkulátor oddělit a umí také zjistit, zda je či není rovna nule. Číslo 2^{n-1} vytvoříme obdobně, jako v předcházejícím programu, výchozím číslem bude ale 0,5 a budeme je před každým cyklem výpočtu násobit dvěma.

Abychom poznali, kdy je výpočet ukončen, musíme zjistit, máme-li v základním čísle před desetinnou čárkou ještě nějakou jedničku, čili je-li celá část čísla rovna nule či nikoli. Tento test zařadíme do programu podobně jako v minulém případě.

Je-li výpočet ukončen, přečte se výsledek, „nastřádaný" opět v paměti č. 02.

Program vypadá následovně (už poněkud stručněji):
Do paměti č. 05 vložíme zadané číslo: **STO 05**
Do paměti č. 01 pro vytváření čísla 2^{n-1} vložíme 0,5 (aby tam pro první výpočet po vynásobení dvěma byla jednička): **0,5 STO 01**
A začíná výpočet (jsme na kroku č. 007). Obsah paměti č. 01 vynásobíme dvěma **2 *Prd 01**
Zjistíme, zda číslo z paměti č. 05 **RCL 05**
má část před desetinnou čárkou rovnou nule ***INT**
***x=t**
Je-li odpověď „*ano*", necháme si „okénko" **□**
je-li odpověď „*ne*", výpočet pokračuje tím, že číslo dělíme deseti: **: 10 =**
Výsledek uložíme zpět do paměti č. 05 **STO 05**
oddělíme z něho část za desetinnou čárkou **INV *INT**
a zjistíme, je-li rovno nule ***x=t**
V případě že ano, nic se nepřičítá a vracíme se na začátek výpočtu na programový krok č. 007 **007**
V případě že ne, je zapotřebí číslo 2^{n-1} z paměti č. 01 **RCL 01**
přičíst do paměti č. 02, střádající výsledek **SUM 02**
poté se opět vrátíme na začátek výpočtu, tj. na programový krok č. 007 **GTO 007**
Až bude výpočet ukončen, tj. před desetinnou čárkou již nebude žádná jednička (viz vynechané okénko), vyjmeme z paměti č. 02 výsledek **RCL 02**
a zastavíme program **R/S**
Protože pokyn RCL 02 je na 34. programovém kroku, dopíšeme toto číslo do vynechaného okénka.
Pro další výpočet vymažeme všechny paměti ***CMs**
a vrátíme program na začátek **R/S**

Celý program je po jednotlivých krocích seřazen v tab. 2.

I vy jistě vidíte, že oba prgramy jsou velmi podobné mají stejnou strukturu i stejný počet kroků. Přímo se nabízí možnost je sloučit a vytvořit jeden program pro oba převody (z dekadického na binární a z binárního na dekadický).

Podrobným srovnáním obou programů zjistíte, že se liší pouze v konstantách, tj. v tom, zda násobíme nebo dělíme dvěma nebo deseti. Jinak jsou programy úplně stejné.

Nebudeme proto vkládat konstanty do programů přímo číselně, ale uložíme je do paměti č. 03 a 04 kalkulátoru a v programu je nahradíme pokyny RCL 03 popř. RCL 04

Pro snadnou obsluhu programu použijeme zarážek (labelů) označených písmeny A (pro převod na binární čísla) a B (pro převod na dekadická čísla). Program je v tab. 3.

První část je zadání pro převod z dekadického na binární číslo:
Začíná zarážkou (labelem) **Lbl A**
vložením zadaného čísla do paměti č. 05 **STO 05**
uložením čísla 0,1 (nulu lze vynechat) do paměti č. 01 **.1 STO 01**
Uložíme do paměti č. 03 konstantu 10 **10 STO 03**
a do paměti č. 04 konstantu 2 **2 STO 04**
Nyní pokynem **GTO 33**
přejdeme na programový krok č. 33, kde začíná společná část programu pro výpočet.
Zarážkou (labelem) B začíná zadání pro převod z binárního čísla na dekadické ***Lbl B**
obdobně ukládáme do paměti číslo a konstanty **STO 05**
.5 STO 01
2 STO 03
10 STO 04
a pokračujeme výpočtem na právě následujícím programovém kroku č. 033.

Program výpočtu je shodný s těmi, které jsme již podrobně popsali. Liší se, jak bylo již řečeno, jen v tom, že místo konkrétních konstant 2 nebo 10 v něm najdete výrazy RCL 03 a RCL 04.

Program používáme tak, že zobrazíme na displeji zadané číslo, stiskneme tlačítko A a objeví se jeho binární ekvivalent. V případě zadání binárního čísla stiskneme tlačítko B a objeví se jeho dekadický ekvivalent. Zvolíme-li např. číslo 156, po stisknutí tlačítka A se objeví 100 111 00. Po stisknutí tlačítka B se objeví zpět 156.

V samostatných programech pro jednotlivé převody stiskneme pro získání výsledku tlačítko R/S.

Někomu se možná bude zdát celý tento výklad příliš „polopatický". V tom případě pro něj není určen. Je dost těch, kteří mají programovatelný kalkulátor a ještě příliš nevědí, co s ním. Pro ty je tento článek určen především. Ne snad proto, aby si mohli převádět dekadické číslo na binární nebo naopak, ale aby si uvědomili (z praktického hlediska), že lze tvořit program pouze logickým uvažováním při znalosti základních možností kalkulátoru a jeho obsluhy. **–ek**

Tab. 1.

000 42	STO	020 42	STO
001 05	05	021 05	05
002 00	0	022 22	INV
003 93	.	023 59	*Int
004 01	1	024 67	*x=t
005 42	STO	025 00	00
006 01	01	026 07	7
007 01	1	027 43	RCL
008 00	0	028 01	01
009 49	*Prd	029 44	SUM
010 01	01	030 02	02
011 43	RCL	031 61	GTO
012 05	05	032 00	00
013 59	*Int	033 07	7
014 67	*x=t	034 43	RCL
015 00	0	035 02	02
016 34	34	036 91	R/S
017 55	÷	037 47	*CMs
018 02	2	038 81	RST
019 95	=		

Tab. 2.

000 42	STO	020 42	STO
001 05	05	021 05	05
002 00	0	022 22	INV
003 93	.	023 59	*Int
004 05	5	024 67	*x=t
005 42	STO	025 00	00
006 01	01	026 07	7
007 02	2	027 43	RCL
008 49	*Prd	028 01	01
009 01	01	029 44	SUM
010 43	RCL	030 02	02
011 05	05	031 61	GTO
021 59	*Int	032 00	00
013 67	*x=t	033 07	7
014 00	0	034 43	RCL
015 34	34	035 02	02
016 55	÷	036 91	R/S
017 01	1	037 47	*CMs
018 00	0	038 81	RST
019 95	=		

Tab. 3.

000 76	*Lbl
001 11	A
002 42	STO
003 05	05
004 93	
005 01	1
006 42	STO
007 01	01
008 01	1
009 00	0
010 42	STO
011 03	03
012 02	2
013 42	STO
014 04	04
015 61	GTO
016 00	0

017 33	33
018 76	*Lbl
019 12	B
020 42	STO
021 05	05
022 93	
023 05	5
024 42	STO
025 01	01
026 02	2
027 42	STO
028 03	03
029 01	1
030 00	0
031 42	STO
032 04	04

033 43	RCL
034 03	03
035 49	*Prd
036 01	01
037 43	RCL
038 05	05
039 59	*Int
040 67	*x=t
041 00	0
042 61	61
043 55	÷
044 43	RCL
045 04	04
046 95	=
047 42	STO

048 05	05
049 22	INV
050 59	*Int
051 67	*x=t
052 00	0
053 33	33
054 43	RCL
055 01	01
056 44	SUM
057 02	02
058 61	GTO
059 00	0
060 33	33
061 43	RCL
062 02	02
063 47	*CMs
064 91	R/S