COMMPAS

vydavatelství a služby
uživatelům mikropočítačů COMMODORE

Programujeme Commodore 64

Aleš Vychodil

COMMODORE 64

O B S A H

1. Úvod

Úvodem bychom vám měli sdělit, proč tato kniha vlastně vznikla. Nikdo, ani ten nejlepší programátor, nemůže vědět vše a zde právě zasahuje naše kniha. Budou vám na následujících stranách objasněny všechny příkazy standardního BASICU V 2.0, vylepšené verze SIMONS BASICU, najdete zde výpis BASIC ROMKY a KERNAL ROMKY, seznámíme Vás se základy PASCALU, uveřejníme užitečné TIPY a TRIKY pro programování, naučíme vás základy ASSEMBLERU, objeví se zde i popis výstupních konektorů a nakonec výpis jednoduchých programů, které by vám neměly chybět. Tak tedy s chutí do toho a půl je hotovo. Přeji Vám za celý vydavatelský tým aby C64 byla vaším pomocníkem, který za vás ochotně odřře všechu práci, se kterou byste se jinak lopotili vy. Tak tedy už nezbývá než vám popřát

hodně štěstí a úspěchu!

2. Na co použít C64

Vyjmenovat všechny možné aplikace se nám jistě nepodaří a i kdyby, zabralo by to nejméně polovinu knihy, a to by asi nikdo nechtěl. Stejně jako u většiny jiných počítačů lze C64 používat k hraní her. S tím se setkáme většinou u uživatelů začátečníků, ale není to pravidlo. Spousta her podporuje fantazii, schopnost rychlé reakce, občas i logiku a různé další vlastnosti. Samostatnou kapitolou jsou simulátory, které mohou byt i užitečné. Rozhodně hry k počítačům neodmyslitelně patří, a kdyby k ničemu jinému nebyly, ukazují čeho všeho je váš počítač schopen.

Počítač se však může stát i učitelem. Vyukové programy jsou velice oblíbené, zvlaště pokoušejí-li se s žákem nějakým způsobem komunikovat.

Některé pomocí klávesnice, jiné přes zvukový výstup - tedy např. hlasem. Předmět výuky je různý: počínaje výukou programování a konče výukou jazyků a pod.

Počítač za nás může řešit i různé, zdánlivě od počítačů odvislé úlohy. Dokáže nakreslit graf zadané funkce,

umí ho zvětšit, zmenšit a nevím co ještě. Umí udělat optimální rodinný rozpočet, tedy takový, který nejlépe vyhovuje zadaným podmínkám. S jeho pomocí můžeme snadno řešit logické i jiné problémy. Zkusme třeba tohle: číslo 25 lze bez obtíží napsat jako 5+5+5+5+5=25, tedy pěti pětkami. Dokážete napsat číslo 100

a) pěti jedničkami
b) pěti trojkami
c) pěti pětkami
d) pěti devítkami

Věřím, že se vám to povede dříve, nebo později. Zkrátka myslím, že bychom se mohli shodnout na tom, že počítač je velmi uzitečná věc, která je tu proto, aby nám pomáhala. C64 je ve své třídě jedním z nejlepších. Pro domácí účely myslím, plně postačuje. Jeho vnitřní struktura je logická a snadno pochopitelná.

Příkazy

Příkazy jsou naším prostředníkem při zapisování programu. Čím více jich známe, tím snadněji se nám programuje. Na následujících několika stránkách najdete seřazené příkazy a standardní proměnné podle abecedy. Ještě bych měl dodat, že k většině příkazů existuje také jejich zkrácený zápis. Oceníte ho, až budete psát nějaký delší program. A nyní k jednotlivým příkazům a proměnným:

ABS (absolute - absolutní)

```
A=ABS(-324):B=ABS(X*2-12*Y):IF C=ABS(C) THEN ?"OK"
```

Numerická funkce, která vypočte absolutní hodnotu členu v závorce.

AND (and - a)

```
AA=12678AND1563:IF(B*3(10)AND(B*3)=0)THEN B=B+11
```

Operátor, provádějící logický součin dvou výrazů

ASC (ASCII = American Standard Code for Information

Interchage - soubor alfanumerických znaků ve formně osmibitoveho kódu)

```
PRINT ACS("a"):ON ASC(A$)-64 GOTO 300:XY%=ASC("*")
```
Funkce, která ke znaku vyhledá příslušný ASCII kód. Kódy všech znaků se pohybují od 0 do 255. V žádném případě nemuže být menší nebo větší.

ANT (arcustangens)

```
XY=ANT(AA):AB=ANT(12):BN=ANT(X+34*Y-12)
```
Numerická funkce, která je opačná k funkci tangens. Její význam oceníme při matematických výpočtech.

CHR$ (character - znak)

```
A$="OKA"+CHR$(89)+CHR$(13):PRINT A$:BB$=CHR$(66)
```
Funkce, která ASCII kódu přiřadí příslušný znak. Je opakem funkce ASC.

CLOSE (uzavřít)

```
OPEN1,4,7:PRINT#1,"test uzavírání souboru":CLOSE 1
```
Příkaz k uzavření souboru. V praxi to znamená konec práce s nějakým periferním zařízením.

CLR (clear - nulovat, čistit)

```
A%=5:A$"zkouška clr":CLR:PRINT A%,A$,"tot vše"
```
Nulování, mazání hodnot všech proměnných.

CONT (continue - pokračování)

```
PRINT A,B$:A=##$:B$="":CONT
```
Příkaz, který po zastavení programu příkazem STOP nebo klávesou STOP, spustí program znovu od zastaveného místa. Hodnoty proměnných se zachovávají a můžeme je nejen nechat vypsat, ale i měnit.

COS (cosinus)

Matematická funkce, která vypočte kosínus argumentu v závorce. Nutno dodat, že pracuje s argumentem zadaným v radiánech. Platí jednoduchý vztah, podle kterého je 108 stupňů rovno 3.14159265 (pí) rad.

DATA

```
DATA Ringo,John,Paul,George,4:read
A$,B$,C$,D$:E$="BEATLES="+A$+B$+C$+D$:READ   A:PRINT   E$,A
```

Příkaz umožňující uložení čísel nebo řetězců, oddělených čárkou, do programu.

DEF FN (define funktion - definuj funkci)

Příkaz pro definování vlastní funkce. Výhoda tohoto příkazu spočívá v tom, že nemusíme opakující se výpočty psát stále znovu, ale stačí "zavolat" jenom tu správnou funkci. Pro název funkce platí stejná pravidla jako pro název proměnné.

DIM (dimension - rozměr)
Příkaz, který zavede jedno, dvou nebo třírozměrové číslo podle čísel nebo řetězců. Index jednotlivých polí se pohybuje od 0 k "nadimenzované hodnotě". Zbývá dodat, že pole o rozměru 0-10 je dimenzováno automaticky, tedy bez použití příkazu DIM.

END (konec)
Zakončení programu. Běh programu končí na posledním programovém řádku. Nicméně následují-li podprogramy, nezbývá nic jiného, než poslední řádek hlavního programu opatřit tímto příkazem.

EXP (exponential function - exponeciální funkce)

```
E=EXP(1):PRINT E
```

Matematická exponenciální funkce. Uvedený příklad můžeme matematicky zapsat jako E=e1, kde e je Eukleova konstanta, základ přirozených logaritmů a je rovna přibližně číslu 2,71828183.

FN (funktion - funkce)
Po nadefinování můžeme funkci volat klíčovým slovem FN a jménem, popř. parametrem. Ulehčuje práci při různých výpočtech a šetří místo v paměti.

FOR...TO...(STEP...) (Pro..do..(krok..))

Tento soubor příkazů nám pomůže vytvářet tzv. cykly, tedy úseky programu, které se opakují tak dlouho, dokud není splněna podmínka udaná v záhlaví toho kterého cyklu. Ještě je důležité vědět, že cykly se mohou prolínat, takže můžeme udělat cyklus v cyklu, a to tolikrát, kolikrát nám to dovolí kapacita zásobníku, ale pro většinu běžných aplikací jejich počet stačí.

FRE (volný, neobsazený)

Funkce, která nám vypiše, kolik místa v paměti nám ještě zbývá.

GOSUB (go to subroutine - jdi do podprogramu)

Příkaz umožňující skok do podprogramu. Správný, dobře strukturovaný program by měl mít 2 části: podprogramy a hlavní program, který je používá. Jako podprogram vytvoříme tu část programu, která by se mohla opakovat. Jednak tím šetříme paměť a pak taky sebe při případném oživování. Na konec podprogramu dáváme příkaz RETURN, který způsobí návrat na řádek, odkud byl podprogram vyvolan, a to hned za příkaz GOSUB. S počtem podprogramů v programu je to asi tak jako s počtem cyklů v cyklu.

GET (vezmi, seber)

Příkaz k přečtení znaku, popřípadě i čísla. Znak muže být v klavesnicovém bufferu (GET) nebo v nějakém souboru (GET #).

GOTO (go to - jdi na)

Příkaz k pokračování běhu programu na stanoveném řádku.

IF...THEN (jestliže...pak)

```
100 INPUT"X":X:IF X<10THEN 100
560 IF POS(0)>20 THEN PRINT "konec řádku"
```

Tyto dva příkazy nám dávají možnost měnit chod programu v závislosti na splnění dané podmínky.Po příkazu THEN může následovat buď číslo řádku, na kterém se má pokračovat,

nebo jiné příkazy, které mohou vyplnit zbytek řádku za THEN. Nedoporučuji dělat pomocí těchto příkazů cykly. Jsou asi 10x pomalejši než FOR-TO-NEXT.

INPUT (vstup)

Příkaz vstupu. Umí přečíst jak řetězec, tak i hodnotu čísla. V případě, že místo čísla zadáme znak, hlásí to a opakuje se tak dlouho, dokud nedostane to, co žádá. V zadávaném textu se nesmí vyskytovat čárka, protože ji chápe jako oddělovač. Opět se liší INPUT z klávesnicového bufferu od INPUT# ze souboru.

INT (integer - celá hodnota)

Funkce, která od desetinného čísla "odsekne" jeho desetinnou část, a tím z něj udělá číslo celé. Nemá nic společného se zaokrouhlováním. Vychází z principu uložení kladných a záporných čísel ve dvojkové soustavě.

LEFT$ (left - levý)

Tento příkaz pracuje s řetězcem a dokáže z něj oddělit x znaků zleva.

LEN (lengh - délka)

Tato funkce dokáže spočítat počet znaků zadaného řetezce.

LET (nechť, ať)

Přiřazovací příkaz. Obvykle se používá, ale nic se tím nezkazí.

LIST (seznam)

Příkaz pro vypsání programu. Může se vypsat jeden řádek, určitý úsek nebo celý program. Každý, kdo chce ochránit svůj program v BASICU před očima rozebíračů, se ho snaží nějak blokovat. Jak, to si povíme později.

LOAD (zavést, naplnit)

Příkaz pro čtení dat, např. programu z periferie (magnetofon, disketa).

LOG (logarithm - logaritmus)

Matematická funkce. Jedná se o logaritmus (v základě má číslo e). Používá se při různých matematických operacích.

MID$ (middle of string - střed řetězce)

Příkaz, který z daného řetězce vytvoří řetězec nový tím způsobem, že si z něj prostě kousek vezme.

NEW (nový)

Tento příkaz přestaví některé systémové proměnné a za 2048 uloží několik nul, takže program se jeví jako vymazaný. Oživit ho lze malým trikem. Jinak po zadání tohoto příkazu můžeme začít psát nový program.

NEXT (příští, následující)

Tento příkaz je součástí cyklu FOR TO. Způsobí přičtení kroku k dané proměnné. Pak ji porovná s cílovou hodotou a je-li menší nebo rovna, opakuje se úsek programu mezi FOR-TO a NEXT. Je-li větší, program pokračuje dále.

NOT (negace - ne)

Logický operátor negace. Myslím, že není třeba k ní cokoliv dodávat.

ON (v závislosti, podle)

Výhodný příkaz k větvení programu v závislosti na splnění podmínky nebo podle hodnoty proměnné. Rozhodování je několikrát rychlejší, než při klasickém IF-THEN. Je-li proměnná v záhlaví (P) nulová, pokračuje program na následujícím řádku.

OPEN (otevři)

OPEN1,4,7:PRINT#1,"zkouška":CLOSE1
Příkaz k otevření souboru. Číslo souboru 1. Číselný kód periferie, se kterou budeme pracovat, je 4. Sekundární adresa je 7.

OR (nebo)

Operátor logického součtu.

PEEK (pokukovat)

Tento příkaz slouží k přečtení obsahu paměťového místa a může mít hodnotu v rozmezí od 0 do 255. Pomocí tohoto příkazu můžeme číst z paměti RAM, BASIC ROM, Character a Kernal ROM. Nelze číst obsah paměti RAM pod ROM.

POKE (vrtat se do něčeho)

Příkaz sloužící k zápisu čísla do paměti RAM. Hodnota, kterou připisujeme na danou adresu, se musí pohybovat od 0 do 255.

POS (position - pozice)

Pomocí tohoto příkazu můžeme zjistit polohu kurzoru na obrazovém řádku.

PRINT (tisknout)

Tento příkaz je jedním z nejpoužívanějších. Tiskne na obrazovku nebo na příslušné místo údaje, zapisuje do souboru (PRINT#). Zvláštní úlohu má u tohoto příkazu čárka a středník (velké rozestupy po čárce a malé po středníku).

READ (čti)

Příkaz pro načtení položky z datařádku. Položky na datařádku jsou odděleny čárkami.

REM (remark - poznámka)

Tento příkaz umožňuje vkládat do programu komentáře, které celý program zpřehledňují. Zbytek řádku za REM je tedy určen pro vlastní poznámky.

RESTORE (obnovit)

Příkaz, který způsobí, že ukazatel na aktuální položku datařádku se nastaví opět na začátek. Data se začínají číst od začátku.

RETURN (návrat)

Tento příkaz dáváme na konec podprogramu. Způsobí návrat za příkaz GOSUB, který volal poslední podprogram.

RIGHT$ (pravý)

Příkaz, který je velice podobný LEFT$. Parametry má dva. Jednak je to řetězec a pak číslo nebo proměnná, určující, z kolika znaků se utvoří nový řetězec.

RND (random - náhoda)

Příkaz, který generuje náhodné číslo. Velikost tohoto čísla se pohybuje od 0 do 0,99999999, proto se obvykle násobí nějakou konstantou a pak se z něj dělá celá hodnota. Tím dostaneme číslo, které je v praxi mnohem více použitelné.

RUN (běžet)

Příkaz pro spuštění programu. Nuluje proměnné. Funguje jenom tehdy, je-li na úplném počátku RAM pro BASIC uložena 0.

SAVE (spasit, střádat)

Příkaz pro uložení programu do periferního zařízení. Uložení programu na magnetofon je poměrně spolehlivé, ale velmi zdlouhavé. Proto se používají urychlovače (turba). Záznam na pásce je kratší, ale méně spolehlivý.

SGN (signum)

Matematická funkce, která nabývá hodnot -1, 0 a 1. Její hodnota závisí na znaménku argumentu. Pro 0 je nulová.

SIN (sinus)

Matematická funkce sinus. Argument je třeba udat v radiánech.

SPC (space - prostor)

Funkce, která mezi jednotlivé řetězce vkládá stanovený počet mezer. První mezera následuje za posledním znakem prvního řetezce a poslední je před prvním znakem druhého řetězce. Tím se tento příkaz liší od příkazu TAB.

SQR (square root - druhá odmocnina)
Pomocí tohoto příkazu můžeme vypočítat druhou odmocninu. Parametrem je číslo od 0 výš.

STATUS (stav)

Standardní vstupně/výstupní proměnná, ukazuje konec bloku, odvozuje se z ní hláška LOAD ERROR a pod. Je v podstatě obrazem adresy 144. Protože je to proměnná, berou se v úvahu pouze první dva znaky. Stačí tedy psát ST.

STEP (krok)

Tento příkaz lze použít jako součást cyklu FOR-TO-NEXT. K proměnné se u NEXT přičte hodnota uvedená za STEP. K funkci cyklu však není bezpodmínečně nutný. Není-li uveden, je hodnota kroku rovna 1.

STOP (stát)

Stop je velmi užitečný příkaz. Napíšeme-li ho do programu, způsobí jeho zastavení a vypíše číslo řádku, kde k zastavení došlo. Hodnoty proměnných se zachovávají a můžeme s nimi různě manipulovat. Program znovu spustíme příkazem CONT.

STR$ (straight - přímo)

Pomocí tohoto příkazu snadno převedeme číslo do řetězce. Přímo to totiž nejde.

SYS (systém)

Příkaz pro skok do programu ve strojovém kódu. Je-li zakončen instrukcí RTS, program v BASICU pokračuje dále za SYS.

TAB (tabulace - sestavit do tabulky)

Příkaz k posunutí kurzoru o (X) znaků od okraje. X se pohybuje, podobně jako u SPC, od 0 do 255. Není možné přepsat text napsaný tímtéž příkazem PRINT, který obsluhuje TAB. Pokud bychom se o to pokusili, příkaz TAB se nebude brát v úvahu.

TAN (tangenta)

Goniometrická funkce tangens. Argument funkce zadáváme v radiánech (180 stupňů = 3.14 (pí))

TIME (čas, doba)

Standardní proměnná ukazující čas od zapnutí počítače. TI je numerická a po vydělení 60 ukazuje čas od zapnutí počítače v sekundách. TI$ je řetězcová a ukazuje čas v hodinách, minutách a sekundách.

USR (use - užívat)

Příkaz, který umožňuje užití programu ve strojovém kódu s přenosem parametrů. Na adresy 785, 786 uložíme adresu počátku programu, který chceme volat.

VAL (value - hodnota)

Funkce, která z řetězce vybere zleva vše co se může vyskytnout v číselném oboru, a přiřadí to dané proměnné. Ostatního si nevšímá.

VERIFY (ověřit)

Tento příkaz po uložení dat (na magnetofon) zkontroluje celý záznam, je-li opravdu v pořádku. Jak test dopadl, to se dozvíme z hlášky, kterou vypíše.

WAIT (čekat)

WAIT 198,1:GET A$: PRINT A$:
Tento příkaz zastaví běh programu a čeká, až se na dané adrese (198) objeví požadovaná hodnota (1). V jiném případě můžeme za první hodnotu zapsat ještě hodnotu druhou. Obě samozřejmě oddělené čárkami. Vztah mezi těmito hodnotami je EXCLUSIVE OR. V uvedeném příkladu se čeká, až bude v klávesnicovém bufferu první znak. Může se ovšem stát, že buffer obsahuje více než 1 znak a program se v tomto místě zakousne. Proto bychom měli buffer nastavit do výchozí polohy, kdy v něm je 0 znaků. Lze to udělat např. příkazem POKE 198,0.

Tak toto byl poslední příkaz, který C64 zná. Zdá se vám to málo? Možná máte trochu pravdu. Chybí příkazy pro grafiku, zvuk apod. Ale není vůbec žádný problém si některé příkazy přidělat. Stačí sehnat např. SIMONS BASIC a hned máte o 114 příkazů navíc. Ulehčuje odlaďování programů, práci s grafikou a pod.

Co se týče příkazů měli byste si každý vyzkoušet, abyste zjistili, co je a co není možné provádět. Určitě se vám to vyplatí.

OBRAZ

Jednou z předností C64 je jeho video obvod. V této kapitole bychom se chtěli podívat na některé jeho vlastnosti.

Hrubá grafika, znakové sady

Pod pojmem hrubá grafika si můžeme představit třeba obrázek

nakreslený pomocí jedné ze standardních znakových sad. Kód znaků se ukládá do obrazovkové RAM, která má 40 zn./řád. * 25 řád. = 1000 Bytů. Kódy znaků můžeme do této paměti zapsat buď přímo nebo pomocí prostředníka v BASICU, příkazem PRINT. VIC si přečte kód a podle něj vyhledá ve znakové ROM tvar příslušného znaku a ten přenese na obrazovku. Aby obvod věděl, kde má co hledat, musíme mu dát potřebné informace. Informace potřebuje také rutina, která zajišťuje tisk na obrazovku. Tyto informace zapíšeme do dvou registrů a na jednu systémovou proměnnou.

V registru CIA#2 nastavujeme, ve které části paměti bude ležet obrazovková a znaková paměť. Jak jsem se již zmínil, VIC byl udělán na zakázku, proto je částečně předprogramován. Ve standardním režimu začíná obrazovková RAM na adrese $0400 a znaková ROM na $D000. Pokud umístíte obrazovkovou paměť jinam, ihned proveďte stisk CHR$ (147)- mazání obrazovky. Tím se aktivují systémové proměnné pro KERNAL. Pokud bychom to neudělali, znaky by se dále zapisovaly do předchozí obrazovkové paměti. Dělá to dojem, že se počítač zakousnul.

V obrazovkové paměti jsou kódy znaků uloženy po jednotlivých obrazovkových řádcích, ale jak jsou uložena data jednotlivých znaků?
Podívejme se, jak jsou uložena data pro znaky A, B ve znakové ROM:

adresa	znak	76543210	adresa	znak	76543210
$D008	**	00011000	$D010	*****	01111100
$D009	****	00111100	$D011	** **	01100110
$D00A	** **	01100110	$D012	** **	01100110
$D00B	******	01111110	$D013	*****	01111100
$D00C	** **	01100110	$D014	** **	01100110
$D00D	** **	01100110	$D015	** **	01100110
$D00E	** **	01100101	$D016	*****	01111100
$D00F		00000000	$D017		00000000

Tedy tímto způsobem můžeme vyrábět své vlastní znaky. Použití je široké. Můžeme si udělat české znaky nebo různé grafické symboly popřípadě pozadí k nějaké hře. Záleží už jen, a jen na naší fantazii. Kolik Bytů zabírá plná (256 znaků) znaková sada? 8*256=2048, tedy 2KB. V C64 jsou sady dvě, proto zabírají 4KB paměti.

Pokud se chceme blíže podívat na paměť našeho Commodoru, musíme si ujasnit některé pojmy.

Nejprve k té paměti. Paměť si můžeme rozdělit do dvou skupin RWM a ROM. Označené RWM znamená, že paměti tohoto typu jsou určeny pro zápis a čtení informace uživatelem. V praxi se tento typ paměti vžil poněkud nesprávně jako RAM. Paměti typu ROM jsou určeny pouze pro čtení jejich obsahu. Jsou naprogramovány už při výrobě.

Protože informací v paměti bývá velké množství, je třeba se dostat nějakým způsobem k té, kterou potřebujeme. K tomu nám stačí znát adresu té paměťové buňky, která nás zajímá. Data jsou uložena v paměti po jednotlivých slovech-Bytech, a to ve dvojkové soustavě. Každý Byte se skládá z 8 bitů. Podívejme se na obrázek:

```
Adr.B. 0 76543210
Adr.B. 1 76543210
Adr.B. 2 76543210
   .  .     .
   .  .     .
   .  .     .
Adr.B.65535 76543210
```

Každý Byte obsahuje 7 bitů. Bit může mít hodnotu 0 nebo 1. Má-li hodnotu 0, říkáme, že je vypnutý, 1 - zapnutý. Počet možných kombinací v Bytu je 2-1 (počítáme od 0). První buňka paměti má adresu 0, poslední 65535. Adresa se někdy skládá ze dvou Bt. nižšího (Low) a vyššího (High). Tomu nižšímu říkáme adresa ve stránce a tomu vyššímu stránka. K dispozici tedy máme 256 stránek (0-255) a paměť má kapacitu 65536 Bytů, tedy 64 KB. Zkusme se nyní zamyslet nad tím, jak bychom mohli rozložit adresu do jednotlivých Bytů. Pokud máme adresu menší než 256 je to jednoduché, ale mějme třeba adresu 2049. Jak na to? Velice jednoduše. V BASICU bychom to zapsali:

```
HI=INT (adresa/256): LO = adresa-256*HI
```

Některé obvody mají v paměti vyhrazena svá místa. Na tato místa ukládáme informace, které si přečteme a podle jejich obsahu zaměřujeme dále svou činnost. Anebo je tam naopak uloží a je na nás, jak s nimi naložíme. Těmto místům říkáme registry. Do některých registrů zapisujeme parametry zvuku, z jiných registrů můžeme naopak cosi vyčíst.

A nakonec ještě pro ty, kdo stále nevěří, že na 8 bitech může nastat právě 256 kombinací, uvádím část z nich. Ostatní je snadné si odvodit.

Kombinace	Hodnota	Hex.tvar	76543210
1	0	$00	00000000
2	1	$01	00000001
3	2	$02	00000010
4	3	$03	00000011
5	4	$04	00000100
6	5	$05	00000101
7	6	$06	00000110
8	7	$07	00000111
9	8	$08	00001000
10	9	$09	00001001
11	10	$0A	00001010
12	11	$0B	00001011
13	12	$0C	00001100
14	13	$0D	00001101
15	14	$0E	00001110
16	15	$0F	00001111
17	16	$10	00010000
18	17	$11	00010001
19	18	$12	00010010
20	19	$13	00010011
21	20	$14	00010100
22	21	$15	00010101
23	22	$16	00010110
24	23	$17	00010111
25	24	$18	00011000
26	25	$19	00011001
27	26	$1A	00011010
28	27	$1B	00011011
29	28	$1C	00011100
30	29	$1D	00011101
31	30	$1E	00011110
:	:	:	:
256	255	$FF	11111111

Systémové proměnné

Adresa (Hex)	Adresa (Dec)	Popis
$0000	0	řídicí registr 6510
$0001	1	in/out registr 6510 (obsluha paměti)
$0002	2	nepoužito
$0003-$0004	3-4	adresa procedury pro převod reál na integer
$0005-$0006	5-6	adresa procedury pro převod integer na real
$0007	7	hledaný znak
$0008	8	příznak při hledání řetězce
$0009	9	číslo sloupce před dalším TAB
$000A	10	příznak BASICU: 0 LOAD, 1 VERIFY
$000B	11	ukazatel vstupního bufferu
$000C	12	příznak DIM
$000D	13	typ: $00 numerický, $FF řetězce
$000E	14	typ: $00 real, $FF integer
$000F	15	příznak LIST
$0010	16	příznak pro ident. funkce pole
$0011	17	$00 INPUT, $40 GET, $98 READ
$0012	18	místo pro goniometrické funkce
$0013	19	argument CMD
$0014-$0015	20-21	číslo řádku při GOTO, LIST, ON, GOSUB
$0016	22	ukazatel místa v řetězcích
$0017-$0018	23-24	adresa předchozího řetězce
$0019-$0021	25-33	zásobník pro 3 řetězce (délka, adresa)
$0022-$0025	34-37	místo sloužící různým cílům
$0026-$002A	38-42	místo pro násobení s pohyblivou desetinnou tečkou
$002B	43	ukazatel počátku programu v BASICU
$002D	45	ukazatel konce programu v BASICU
$002E	46	počátek proměnných
$002F	47	ukazatel počátku polí
$0031	49	ukazatel konců polí

Adresa (Hex)	Adresa (Dec)	Popis
$0033	51	ukazatel počátku řetězců
$0035	53	pomocný ukazatel řetězců
$0037	55	ukazatel konce paměti v BASICU
$0039	57	číslo překládané řádky programu
$003B	59	číslo předchozí řádky programu
$003D	61	ukazatel následující instrukce (CONT)
$003F	63	číslo aktuálního datařádku
$0041	65	adresa aktuální položky datařádku
$0043	67	ukazatel na vstupní rutinu
$0045	69	jméno aktuální proměnné v BASICU
$0047	71	adresa aktuální proměnné
$0049	73	adresa aktuální proměnné cyklu (FOR)
$004B	75	pomocný ukazatel při překladu
$004D	77	maska při operaci relace
$004E	78	místo sloužící různým cílům
$0054	84	vektor skoku pro funkci
$0057	87	místo pro numerické operace
$0061	97	Real akumulátor 1 (exponent)
$0062	98	Real akumulátor 1 (mantisa)
$0066	102	Real akumulátor 1 (označení)
$0067	103	počet elementů výrazu
$0068	104	přestupný Byte z akumulatoru 1
$0069	105	Real akumulátor 2
$006F	111	výsledek porovnání akumulatoru 1, 2
$0070	112	zaokrouhlený Byte akumulátoru 1
$0071	113	délka kazetového bufferu
$0073	115	procedura CHRGET: čtení znaku
$008B	139	báze pro generování náhodných čísel
$0090	144	stavový byte přenosu in/out

Adresa (Hex)	Adresa (Dec)	Popis
$0091	145	příznak stisknuté klávesy STOP
$0092	146	časová konstanta LOAD z magnetofonu
$0093	147	příznak pro Kernal: 0 LOAD, 1 VERIFY
$0094	148	sériová sběrnice znaků v bufferu
$0095	149	znak před vysíláním na seriový bus
$0096	150	synch. na konci pásky
$0097	151	pomocná paměť pro registry X, Y
$0098	152	počet otevřených souborů

I $0099 I	153 I	kód vstupu: 0 klávesnice	I
I $009A I	154 I	kód výstupu: 3 obrazovka	I
I $009B I	155 I	parita při čtení z pásky	I
I $009C I	156 I	příznak při odebírání Bytu	I
I $009D I	157 I	typ: $00 programový, $80 přímý	I
I $009E I	158 I	kontrolní suma	I
I $009F I	159 I	korekce chyb při nahrávání	I
I $00A0 I	160 I	aktuální čas od zapnutí počítače	I
I $00A3 I	163 I	místo pro bity (pro sériový bus)	I
I $00A4 I	164 I	obsluha impulsu	I
I $00A5 I	165 I	zápis bitu na pásku	I
I $00A6 I	166 I	ukazatel bufferu magnetofonu	I
I $00A7 I	167 I	obsah sloužící pro práci s RS 232	I
I $00AC I	172 I	ukazatel na počátek načítání dat	I
I $00AE I	174 I	ukazatel konce pro LOAD, SAVE, VERIFY	I
I $00B0 I	176 I	časová konstanta při čtení z pásky	I
I $00B2 I	178 I	ukazatel poč. bufferu při čtení z mag.	I
I $00B4 I	180 I	příznak pro čtení z pásky: RS 232	I
I $00B5 I	181 I	příznak konce pásky: násl. bit RS 232	I
I $00B6 I	182 I	Byte před vysláním RS 232	I
I $00B7 I	183 I	délka názvu souboru	I
I $00B8 I	184 I	číslo aktuálního kanálu	I
I $00B9 I	185 I	sekundární adresa	I
I $00BA I	186 I	kód periferie	I
I $00BB I	187 I	ukazatel na adresu názvu souboru	I
I $00BD I	189 I	znak čtení/zápis na pásku: Parita RS	I
I $00BE I	190 I	počet bloků dat	I
I $00BF I	191 I	buffer pro čtení Byte z pásky	I
I $00C0 I	192 I	příznak ovládání magnetofonu	I
I $00C1 I	193 I	počáteční adresa při oper. in/out	I
I $00C3 I	195 I	pomocný příkaz při práci s páskou	I
I $00C5 I	197 I	kód dříve stisknuté klávesy	I
I $00C6 I	198 I	počet znaků v bufferu klávesnice	I
I $00C7 I	199 I	příznak: zapnutá inverze tisku	I
I $00C8 I	200 I	počet znaků na počítačovém řádku	I
I $00C9 I	201 I	číslo řádky s kurzorem	I
I $00CA I	202 I	pozice kurzoru ve sloupci	I
I $00CB I	203 I	kód právě stisknuté klávesy	I
I $00CC I	204 I	příznak kurzoru: $00...bliká	I
I $00CD I	205 I	typ blikání kurzoru	I
I $00CE I	206 I	znak, nad kterým je kurzor	I
I $00CF I	207 I	negativ/pozitiv znaků pod kurzorem	I

I	$00D0	I	208	I	znak z klávesnice/obraz	I
I	$00D1	I	209	I	adresa aktuálního řádku v obraz. RAM	I
I	$00D3	I	211	I	sloupec, ve kterém je kurzor	I
I	$00D4	I	212	I	příznak (úvozovky): 0 nebyly	I
I	$00D5	I	213	I	max. délka řádku z obrazovky	I
I	$00D6	I	214	I	řádek, na kterém je kurzor	I
I	$00D7	I	215	I	kód aktuálního znaku	I
I	$00D8	I	216	I	počet znaků před přečtením	I
I	$00D9	I	217	I	pomoc. údaje pro obrazové řádky	I
I	$00F3	I	243	I	adresa aktuální řádky v paměti barev	I
I	$00F7	I	247	I	adresa poč. vstup. bufferu pro RS 232	I
I	$00FB	I	251	I	čtyři volné Byty pro uživatele	I
I	$00FF	I	255	I	převod čísel na řetězce	I
I	$0100	I	256	I	zásobník procesoru	I
I	$0200	I	512	I	vstupní buffer BASICU	I
I	$0259	I	601	I	tabulka Kernalu akt. čísla souboru	I
I	$0263	I	611	I	tabulka Kernalu akt. čísla periferií	I

IAdresa I(Hex)	IAdresa I(Dec)	I	Popis	I
I	$0277	I	631	I	vstupní buffer klávesnice	I
I	$0281	I	641	I	ukazatel počátku paměti pro BASIC	I
I	$0283	I	643	I	ukazatel konce paměti pro BASIC	I
I	$0285	I	645	I	příznak pro IEC-625	I
I	$0286	I	646	I	aktuální kód barvy znaku	I
I	$0287	I	647	I	aktuální kód barvy pozadí	I
I	$0288	I	648	I	Hi Byte adresy obrazovkové RAM	I
I	$0289	I	649	I	velikost kláv. bufferu	I
I	$028A	I	650	I	opakovací funkce kláves	I
I	$028B	I	651	I	zpoždění mezi opětovným čtením	I
I	$028C	I	652	I	zpoždění pro potvrzení	I
I	$028D	I	653	I	příznak (stisk CONTROL, C=, SHIFT)	I
I	$028E	I	654	I	předchozí stav příkazu SHIFT	I
I	$028F	I	655	I	ukazatel na dekódování klávesnice	I
I	$0291	I	657	I	přepnutí tisk. módu ($00zap., $80vyp.)	I
I	$0292	I	658	I	rolování textu: $00 zap. +-$00 vypnuto	I
I	$0293	I	659	I	RS 232: 6551 povelový registr	I
I	$0294	I	660	I	RS 232: 6551 řídicí registr	I
I	$0295	I	661	I	RS 232: nestandardní par. přenosu	I
I	$0297	I	663	I	RS 232: 6551 stavový registr	I

$0298	664	RS 232: počet bitů před vysláním	
$0299	665	RS 232: rychlost přenosu (bit/mikros.)	
$029B	667	RS 232: ukaz. v bufferu odebrání znaku	
$029C	668	RS 232: poč. bufferu na odebír. znaku	
$029D	669	RS 232: ukaz. poč. v bufferu vysl. znaku	
$029E	670	RS 232: ukaz. konce v bufferu vysl. znak	
$029F	671	stav IRQ vek. před spoluprací s mag.	
$02A1	673	příznak NMI pro CIA 2	
$02A2	674	parametry pro CIA 1 s magnetofonem	
$02A5	677	pomocný uk. ráfku při rolování	
$02A6	678	příznak: 0 NTSC, 1 PAL	
$02A7	679	volný prostor pro uživatele	
$0300	768	procedura pro tisk chybových hlášek	
$0302	770	hlavní část překladače (teplý start)	
$0304	772	přepis textu na příkazy	
$0306	774	rutina tisknoucí příkazy (LIST)	
$0308	776	zpracování následujícího příkazu	
$030A	778	procedůra pro aritmetické vyjádření	
$030C	780	stav A (akumulátoru) před SYS	
$030D	781	stav X před SYS	
$030E	782	stav Y před SYS	
$030F	783	stav SP před SYS	
$0310	784	kód $4C JMP Hi Lo (pro USR)	
$0311	785	Lo skok do programu ve stroj. kódu	
$0313	787	volný Byte	
$0314	788	hardware přerušení (IRQ-stand.: $EA31)	
$0316	790	přerušení po instrukci BRK ($FE66)	
$0318	792	nemaskovatelné přerušení (NMI-FE47)	
$031A	794	vektor: rutina OPEN	
$031C	796	vektor: rutina CLOSE	
$031E	798	vektor: rutina CHKIN	
$0320	800	vektor: rutina CHKOUT	
$0322	802	vektor: rutina CLRCHN	
$0324	804	vektor: rutina CHRIN	
$0326	806	vektor: rutina CHROUT	
$0328	808	vektor: rutina STOP	
$032A	810	vektor: rutina GETIN	
$032C	812	vektor: rutina CLALL	
$032E	814	použitý IRQ ($FE66)	
$032F	816	vektor: rutina LOAD	
$0332	818	vektor: rutina SAVE	
$0334	820	osm volných Bytů	

$033C	828	buffer užívaný pouze při práci s mag.
$03FC	1020	čtyři volné Byty
$0400	1024	obrazová paměť RAM
$07E8	2024	šestnáct volných Bytů
$07F8	2040	nastavení Spritu

Adresa (Hex)	Adresa (Dec)	Popis
$8000	32768	toto místo zaujímá vnější paměť ROM
$A000	40960	BASIC ROM (nebo 8K RAM)
$C000	49152	4K RAM (na programy v Assembleru)
$D000	53248	in/out+color RAM, znak. ROM, 4K RAM
$E000	57344	Kernal ROM (nebo 8K RAM)
$0000	0	MOS 6510 řídící registr
$0001	1	MOS 6510 registr
		bit význam
		--- -------
		0...LORAM (0 - Basic ROM vyřazena)
		1...HIRAM (0 - Kernal ROM vyřazena)
		2...CHARAM (0 - znak ROM vyřazena)
		3...data na magnetofon - výstup
		4...spínač v magnetofonu (1 - sepnutý
		5...ovládání motoru (0 - zap, 1 - vyp)
		6...nevyužit
		7...nevyužit

$D000	53248	MOS 6566 Video obvod
$D02E	54271	

$D000	53248	sprite 0, souřadnice X
$D001	53249	sprite 0, souřadnice Y
$D002	53250	sprite 1, souřadnice X
$D003	53251	sprite 1, souřadnice Y
$D004	53252	sprite 2, souřadnice X
$D005	53253	sprite 2, souřadnice Y
$D006	53254	sprite 3, souřadnice X
$D007	53255	sprite 3, souřadnice Y
$D008	53256	sprite 4, souřadnice X
$D009	53257	sprite 4, souřadnice Y
$D00A	53258	sprite 5, souřadnice X
$D00B	53259	sprite 5, souřadnice Y

I $D00C I 53260 I	sprite 6, souřadnice X	I
I $D00D I 53261 I	sprite 6, souřadnice Y	I
I $D00E I 53262 I	sprite 7, souřadnice X	I
I $D00F I 53263 I	sprite 7, souřadnice Y	I
I $D010 I 53264 I	sprite 0-7: 9-tý bit X-ové souřadnice	I
I $D400-I 54272-I	MOS 6581 Zvukový obvod	I
I-$D7FF I-55295 I		I
I $D500-I 54528-I	obrazy adres zvukového obvodu	I
I-$D7FF I-55295 I		I
I $D800-I 55296-I	paměť barev jednotlivých pixelů	I
I-$DBFF I-56319 I		I
I $DC00-I 56320-I	MOS 6526 Mezistykový obvod #1	I
I-$DCFF I-56575 I		I
I $DD00-I 56576-I	MOS 6526 Mezistykový obvod #2	I
I-$DDFF I-56831 I		I
I $DD00 I 56576 I	datový port A (sér.sběrnice,RS232,VIC)	I
I $DD01 I 56577 I	datový port B (RS 232)	I
I $DD02 I 56578 I	řídicí registr portu A	I
I $DD03 I 56579 I	řídicí registr portu B	I
I $DD04-I 56580-I	totéž jako $DC04- 56324-	I
I-$DD0F I-56591 I	-$DC0F -56335	I
I $DE00-I 56832-I	rezervováno pro další možné	I
I-$DEFF I-57187 I	rozšíření obvodu in/out	I
I $DF00-I 57088-I	rezervováno pro další možné	I
I-$DFFF I-57343 I	rozšíření obvodu in/out	I

ULOŽENÍ BASICU a proměnných

Jak je možné, že jeden program dokáže vyrobit jiný? Jak lze takový program sestavit a hlavně vložit do paměti? Na tyto a mnohé jiné otázky vám dokáže odpovědět tato kapitola.

Programy v BASICU mají v paměti vyhrazenou určitou oblast (2048 - 40959). V ní tedy je program zapsán (počínaje 2048). Podívejme se na příklad.

```
10 PRINT "AHOJ": POKE 198,1
20 GETA$: PRINTA$: GOTO 10
```

Tento program se uloží takto:

```
$0800(2048) $00 (pokud zde není0,nejedná se o prg.v BASICU)
$0801(2049) $04 Lo ukazatel na počátek dalšího prog. řádku
$0802(2050) $08 Hi  -//-          -//-           -//-
$0803(2051) $0A Lo číslo programového řádku 10
$0804(2052) $00 Hi  -//-          -//-
$0805(2053) $99 PRINT (kód příkazu)
$0806(2054) $22 " (kód uvozovek)
$0807(2055) $41 A (kód znaku A)
$0808(2056) $48 H (kód znaku H)
$0809(2057) $4F O (kód znaku O)
$080A(2058) $4A J (kód znaku J)
$080B(2059) $22 " (kód uvozovek)
$080C(2060) $3A : (kód dvojtečky)
$080D(2061) $97 POKE (kód příkazu)
$080E(2062) $31 1 (kód znaku 1)
$080F(2063) $39 9 (kód znaku 9)
$0810(2064) $38 8 (kód znaku 8)
$0811(2065) $2C , (kód čárky)
$0812(2066) $31 1 (kód znaku 1)
$0813(2067) $00 9 (konec programové řádky)
$0814(2068) $24 Lo ukazatel na počátek dalšího prog. řádku
$0815(2069) $08 Hi  -//           -//-           -//-
$0816(2070) $14 Lo číslo řádku 20 (decimálně)
$0817(2071) $00 Hi  -//           -//-
$0818(2072) $A1 GET (kód příkazu)
$0819(2073) $41 (kód znaku A)
$081A(2074) $24 $ (kód znaku $)
$081B(2075) $3A : (kód dvojtečky)
$081C(2076) $99 PRINT (kód příkazu)
$081D(2077) $41 A (kód znaku A)
$081E(2078) $24 $ (kód znaku $)
$081F(2079) $3A : (kód dvojtečky)
$0820(2080) $89 GOTO (kód příkazu)
$0821(2081) $31 1 (kód znaku 1)
$0822(2082) $30 0 (kód znaku 0)
$0823(2083) $00 (konec programového řádku)
$0824(2084) $00 (konec programu v BASICU)
$0825(2085) $00
```

Ještě připomínám, že jsme popsali stav před spuštěním programu. Až se bude ukládat proměnná, některé ukazatele se změní ($2F-$34). Tak teď již víme, jak je náš program vlastně ulozen. Ze způsobu uložení vyplývá, že přebytečné mezery by nám zbytečně zabíraly paměť. A jak je to s proměnnými?

Proměnné se ukládají do paměti určené pro BASIC. Proto jim program v BASICU (který používá nějaké proměnné) musí nechat trochu místa. Při letmém pohledu na ukazatele je zřejmé, kam se proměnné budou ukládat.

Stručně to můžeme říci takto: proměnné REAL, INTEGER a parametry řetězců hned za konec programu. Za ně se pak ukládají pole. Naopak od konce paměti směrem dolů se ukládají jednotlivé znaky řetězců. Při ukládání se příliš místem nešetří, proto se může stát, že se paměť naplní daty a mohlo by dojít ke kolizi. V tom okamžiku se však před uložením další proměnné zavolá rutina, která proměnné trochu srovná, a rázem je místa dost. Nicméně pokud je program dlouhý, může se stát, že nás překvapí hláška "Out of memory". Jedinou možností je pracovat s co nejmenším počtem proměnných. A teď se podívejme, jak jsou jednotlivé proměnné ukládány (stav po NEW:CLR):

INTEGER

Proměnná se ukládá na místo, kam ukazuje ukazatel $2D, $30. Na místo, kam by se ukládala další proměnná ukazuje $2F, $30. Na $0E se uloží typ aktuální proměnné ($00 - real, $80 - integer). Adresy $45, $46 obsahují jméno aktuální proměnné a $47, $48 ukazuje do jejich dat (na Hi hodnotu). Poslední tři $00 (nuly) nemají pro hodnotu proměnné žádný význam (srovnávací rutina je vyřazuje)

REAL

Desetinný tvar čísla real (např. X) dostaneme podle následujícího vzorce (decimálně):

```
    a-129    z
X=(2     )(-1 )(1+c), kde
a...číslo od 0 do 255 (exponent)
z...7-mý bit: znaménko (hodnota 0 nebo 1)
c...mantisa (je menší maximálně rovna 1)
```

Mantisa vznikne z jednotlivých bitů slov c1-c4.
Zkusme si to pro AB-513:
a=513:z=0: Byte z+c1 je nulový, tedy nejen z=0, ale i c1=0. Byte c2 je nenulový. Rozepíšeme si ho v binárním tvaru: C2=01000000. Byty c3 a c4 jsou nulové, takže na hodnotu nemají vliv. A teď počítejme zapnutý bit v c2 má označení 9. Tedy:

138-129 0

C=(2)(-1)(1+0.001953125)=513

Hodnota real zabírá na rozdíl od integer sedm Bytů.

STRING

Při ukládání řetězců musíme od sebe odlišovat dvě věci. Za prvé je to samostatný řetězec skládající se pouze ze znaků a za druhé jsou to data nebo údaje o tomto řetězci. Mezi tyto údaje patří jméno proměnné řetězce, počet znaků, ze kterých se skládá, a nakonec je to adresa místa, kde řetězec začíná. Tyto údaje se ukládají mezi proměnné, ale samotný řetězec se ukládá od konce paměti směrem dolů, a to tak, že poslední znak řetězce je blíže ke konci paměti a první blíže k jejímu počátku.

Většina řetězců se začne ukládat o Byte níže, než kam ukazuje $33, $34. Tento ukazatel se pak přednastaví tak, že ukazuje na první znak posledně vloženého řetězce. Jistě jste nepřehlédli slůvko "většina" a jistě vás zajímá, kam že se ukládá ten zbytek, tak tedy: do této oblasti se neukládají řetězce, které jsou uloženy jinde - zpravidla přímo v programu. Jako příklad uvedu:

```
10 C$="AHOJ"
```

V tomto případě se jako počáteční adresa řetězce nastaví adresa, na které leží znak A, a ta se spolu s délkou a názvem řetězce uloží mezi proměnné. Způsob ukládání těchto údajů do proměnných je naprosto stejný jako způsob ukládání jakékoli jiné proměnné (integer, real) včetně použití ukazatelů.

Ještě se zmíním o ukazateli $35, $36 - tento ukazatel při ukládání řetězců ukazuje na předposlední.

POLE

A zbývá nám zjistit, jak se ukládají do paměti pole.

Vyjdeme, stejně jako v předchozích případech, ze stavu po NEW: CLR. Jistě nás nepřekvapí mnohé podobnosti s ukládáním proměnných. Stejně jako máme tři druhy proměnných, máme i tři druhy polí: real, integer, string. Poznávacím znamením je 7-mý bit názvu proměnné.

Je-li pole vícerozměrné, následují dva Byte (Lo, Hi nadimenzována velikost +1) pro každý další rozměr. Pak následuje prostor pro hodnoty. Pro integer jsou to vždy dva Byty, pro real je to už 5 Bytů a pro řetězce jsou to 3 Byty.

Počátek polí ukazuje ukazatel $2F, $30. Na konec polí (a zároveň na místo pro případné další pole) ukazuje $31, $32.

Znalost způsobu uložení proměnných je užitečná nejen pro výhodnou tvorbu programů v BASICU, ale také nám dává možnost vyměňovat si (i ve větším množství) údaje s programem v Assembleru.

JEMNÁ GRAFIKA, BARVY

C64 má standardní jemnou grafiku 320*200 bodů. Grafická paměť RAM tedy zabírá 800 Bytů. Jeden bod na obrazovce tedy odpovídá jednomu zapnutému bitu. Jednotlivé Byty jsou uspořádány stejně jako ve znakové ROM - tedy po osmi v pixelu a jednotlivé pixely jsou seřazeny po obrazovkových řádkách. Ostatně budete mít možnost ověřit si to na některém z příkladů. Podívejme se, jak lze přepočítat souřadnice bodu X (x, y) na jednotlivé Byty. Mějme tedy bod X popsaný souřadnicemi x (0...319) a y (0...199). Pak platí, že RADEK=INT(y/8) : PIXEL=INT(x/8): LINKA=y AND 7: BIT=7-(X AND 7): POCATEK=(počátek grafické RAM) a pokud chceme bod zobrazit, uděláme to takhle:
BYTE=POCATEK+RADEK*320+PIXEL*8+LINKA:POKE BYTE,PEEK(BYTE)OR 2 na BIT
Předtím, než začneme jednotlivé body vynášet je dobře smazat celou grafickou RAM (zapíšeme do ní nuly) a nastavit color RAM.

Grafický režim zapínáme v registru 17 VIC, tedy na adrese 53265 ($D011). Slouží k tomu 5-tý bit, POKE 53265, PEEK(53265)OR32.

Zpět do textového režimu se dostaneme: POKE 53265, PEEK (53265) OR 223,
a zpět do textové color RAM se dostaneme po přepsání registru na adresách 56576 ($DD00) a 53272 ($D018).

Pokud budeme pracovat ve vícebarevném módu, sníží se nám rozlišovací schopnost na 160*200 bodů. Každý může mít jednu ze 4 možných barev. Bude to záležet na tom, jakou hodnotu má vždy dvojice bitů v Bytu. Barvy jim přísluší takto:

bity	grafický režim	textový režim	sprites
00	barva podkladu 53281 ($D021)	barva podkladu 53281 ($D021)	barva podkladu 53281
01	horní 4 bity v graf. color RAM	barva #1 53282 ($D022)	multicolor #0 53285.
10	dolní 4 bity v graf. color RAM	barva #2 53283 ($D023)	barva sprite (r.39-46)
11	dolní 4 bity z text. color RAM	dolní 4 bity z text. color RAM	multicolor #1 53286

Protože grafické režimy jsou velmi podobné, uvedl jsem potřebné údaje i pro textový režim a pro sprity. Kódy barev jsou dokonce stejné.

0...černá
1...bílá
2...červená
3...zelenomodrá
4...purpurová
5...zelená
6...modrá
7...žlutá
8...oranžová
9...hnědá
10...světle červená
11...šedá 1
12...šedá 2
13...světle zelená
14...světle modrá
15...šedá 3

Odlišnosti jsou v zapínání vícebarevných módů. V jemné grafice se to provádí 4-tým bitem registru na adrese 53270 ($D016). Tedy např. takto:
POKE53270,PEEK(53270)OR 16

V textovém režimu se to provádí stejně. Dokonce můžeme použít i tzv. rozšířený barevný mód, který se zapíná 6-tým bitem registru na adrese 53265 ($D011) a kód barev se ukládá do registru na adrese 53284 ($D024).

Barevný mód lze zapínat pro jednotlivé sprity v

registru 28 ($1C), tedy na adrese 53276 ($D01C) a to tak, že nejvyšší bod odpovídá spritu 7, nejnižší spritu 0. A tím se dostáváme ke spritům.

SPRITE

Další předností naší C64 jsou tzv. sprity. Sprite je opravdu malý skřítek, kterého si můžeme zobrazit na obrazovce v textovém i grafickém módu. Jak bude vypadat, to záleží jen na tom, jakými daty ho naplníme. Tento skřítek se umí zvětšovat (2x), dokáže se schovat za levý okraj nebo za znak a ještě další věci. Spritů máme celkem osm a každý můžeme ovládat samostatně. Každý sprite má šířku 3*8 bitů a délku 21 bitů. Data se ukládají po jednotlivých řádcích a to zleva. Tedy nejprve Byte, který je vlevo nahoře, pak Byte prostřední a nakonec Byte vpravo nahoře. Tím jsme uložili první řádek. Stejně uložíme i ostatní, celkem to tedy bude 3*21=63 Bytů. Každý bit jednotlivých Bytů odpovídá jednomu rozsvícenému bodu na obrazovce.

Máme tedy hotová data pro nějakého skřítka, ale kam s nimi? V registru 56576 ($DD00) nastavujeme v negaci jednu ze čtyř bank, se kterou je VIC schopen komunikovat. Paměť se tedy rozpadne na čtyři 16-ti KBytové části. Ve standardním módu jsou v registru $DD00 na příslušných bitech zapsány hodnoty 1 a 1, což po negaci dává hodnoty 0 a 0, tedy pracujeme v bance 0, která začíná na adrese 0 a končí 16383. Data pro jednotlivé sprity můžeme ukládat kdekoli uvnitř nastavené banky.

Programování ve strojovém kódu

Základem počítače C64 je procesor Motorola 6510. BASICOVY překladač překládá náš program v BASICU do kódů, kterým rozumí procesor. Stejně tak ale můžeme napsat program v těchto kódech i my. Jaké má tento způsob programování přednosti a jaké nedostatky, je zřejmé.

Programujeme-li ve strojovém kódu, tedy v kódu, kterému přímo procesor rozumí, pak vlasně překladač BASICU nepotřebujeme. Náš program bude běhat podstatně rychleji,

než by běhal v BASICU, ale na druhé straně přijdeme o veškerý komfort, který nám BASIC nabízí. A teď si vyberte. Výběr nám naštěstí trochu ulehčí požadavky praxe. Podívejme se třeba na programy z minulé kapitoly. Vždy, když jsme přepínali do jemné grafiky, bylo třeba graf. RAM vynulovat a šlo to setsakramensky pomalu. Pokud bychom udělali program na nulování grafické RAM ve strojovém kódu, je celé nulování záležitostí zlomku vteřiny.

Numerické výpočty (a výpočty hodnot funkcí zvláště) patří k nejzdlouhavějším operacím BASICU vůbec. Ale když uvážím, jak dlouho by mi trvalo, než bych vymyslel program svůj ve strojovém kódu, tak tuto práci rád přenechám BASICU. A i kdybych ten program udělal, záhy bych zjistil, že ten program zase moc rychlejší není. Je však ještě jedno řešení: použití rutiny zabudované v překladači, ale to jsme se dostali moc daleko. Tušíme tedy, kdy budeme používat programy ve strojovém kódu a kdy naopak v BASICU.

Programování ve strojovém kódu je poměrně náročné na čas a na bezchybnost (a tím i na hlavu). Nicméně mám takový pocit, že by si to měl každý několikrát zkusit. Kódy se ukládají za sebou do volné oblasti RAM příkazem POKE, čtou se příkazem PEEK a program se spouští příkazem SYS (počátek). Na konec programu dejte instrukci RTS (návrat). A až vám to celkem půjde, opatřte si nějaký monitor strojového kódu např. SUPERMON 64, MONITOR C64 apod. Tento progam nám podstatně ulehčí programování. Za prvé totiž nemusíte psát program v kódech, ale přímo v jednotlivých instrukcích, a za druhé nám taky něco nabízí několik funkcí, které se mohou hodit. Je to např. přenos úseku paměti jinam, ukládání programu na pásku nebo na disketu, výpis úseku paměti v datech nebo v instrukcích atd.. Monitory obvykle neumějí pracovat v desítkové soustavě. Místo toho používají šestnáctkovou soustavu, která je, jak brzy zjistíte, velmi výhodná. Jde jen o zvyk. Jediné, co se nesmí, je psát program do některých systémových proměnných obvodu vst./výstupu a pak do samotného monitoru. Delší verze vás na to upozorní, kratší se v lepším případě zhroutí a v horším udělají to, co nejvíc nechceme.

Popis registrů

Procesor 6510 má několik registrů. K čemu slouží a jak? Jak je budeme značit? Na tyto a jiné otázky se pokusíme nyní odpovědět.

Registr A

Hlavní registr mikroprocesoru 6510 je akumulátor, v assemblerovém těsnopisu označován A. V tomto registru se provádějí všechny důležité operace jako např. sčítání, odčítání, porovnávání aj. Při práci s tímto registrem musíme dbát na jednu skutečnost. Většina matematických operací vyžaduje, aby právě jeden operand byl v akumulátoru, druhý se většinou určuje adresací v paměti. Po provedení příslušné operace se výsledek uloží taktéž do akumulátoru, čímž přepíše původní číslo.

Registry X a Y

Další registry 6510-ky jsou registry X a Y. Tyto, registry stejně jako akumulátor a další přepisované registry, jsou na rozdíl od jiných paměťových buněk ukryty přímo v CPU 6510. Není k nim přístup přímo z BASICU, ale jsou často adresovány pomocí assembleru. Registry X a Y je možno používat dvěma způsoby. První jednoduché využití je k ukládání hodnot, o kterých víme, že je budeme za krátko potřebovat. Druhým užitím jsou čítače relativního umístění, neboli indexové registry.

Registr PC

Dalším registrem, se kterým se seznámíme, je programový čítač, označovaný jako PC. První věcí, které si na PC všimneme je, že je dvakrát větší než ostatní registry 6510, a to 16-bitový. Důvod je jednoduchý. PC registr vždy obsahuje adresu instrukce, která má být prováděna v dalším cyklu, a tudíž musí umět adresovat 65536 adres. Obsah tohoto registru si hlídá přímo procesor a programátor ho může měnit pouze instrukcemi skoku a návratu.

Registr SP

Další registr, který si přiblížíme, se nazývá ukazatel

zásobníku, označovaný jako SP. V počítači je část paměti vyhrazena pro zásobník. Do tohoto zásobníku můžeme vkládat čísla, která se jakoby skládají na sebe a od posledně vloženého čísla zpět je můžeme také číst. K tomu, abychom se v tomto zásobníku vyznali, nám slouží právě registr SP. Při uložení čísla do zásobníku se jeho hodnota automaticky snižuje a naopak při čtení zvyšuje. Při práci se zásobníkem musíme dát pozor na jednu velice důležitou skutečnost. Tento zásobník využívá také příkaz podobný basicovskému GOSUB, který volá v assembleru podprogramy. Tento příkaz si uloží před skokem do podprogramu na vršek zásobníku zpáteční adresu. Později si tedy vysvětlíme, jak zacházet se zásobníkem tak, aby nedošlo ke kolizi. Pokud byste totiž tuto adresu přepsali, program odskočí za registrem SP právě na ukazovanou adresu, což skončí ve většině programů zhroucením.

Registr F

Nyní si probereme snad nejnáročnější registr mikroprocesoru 6510. Tento registr, označený jako F, se nazývá stavový registr. V tomto registru si musíme povšimnout jednotlivých bitů. Každý z bitů má určitou funkci a tomu příslušné označení. Jak později uvidíte, tyto registry jsou velice důležité pro řízení podmíněných skoků programu. Podle výsledku určité operace se nastaví příslušné bity a na té skutečnosti lze větvit program. V tomto případě si však vždy musíte ujasnit, jaká operace nastavuje jaké bity. Nelze např. po instrukcích DEC, resp. INC, které snižují, resp. zvyšují obsah akumulátoru o 1, testovat bit přetečení, jelikož ho tyto instrukce nenastavují. O tom, jaké instrukce ovlivňují příznaky, se dočteme dále při popisování jednotlivých instrukcí.

Označení příznaku	Funkce	Význam při nastavení
C	přenos	nastal
Z	nulovost	nulový
I	přerušení	nepovoleno
D	dec. aritmetika	povolena
B	příkaz zastavení	
V	přetečení	nastalo
N	zápornost	záporný

Nyní si přiblížíme funkci jednotlivých bitů v F registru:

C - Indikační registr přenosu C nám říka, zda výsledek předchozí operace překročil hodnotu 255D. Uvědomme si, že při sčítání dvou čísel menších než 256 nikdy nemůže nastat situace, že by přenos nastal 2x.
Jednobitový indikační systém přenosu je tedy pro naše potřeby dostačující. Jak dále uvidíme, v asembleru je bit přenosu používán ve všech matematických operacích.

Z - Indikační registr nulovosti nás informuje o tom, zda předchozí informace vedla k výsledku 0. Pokud ano, je bit Z rovný 1, v opačném případě je hodnota bitu Z 0.

I - Indikační registr o přerušení nám slouží k povolení, resp. zakázání přerušení vyvolaného signálem IRQ. Jestliže je bit nastaven na 1, potom není možné maskované přerušení. Pro úplné pochopení tohoto příznaku bychom museli zabloudit do hardwaru počítače.

D - Mikroprocesor 6510 má dva režimy, ve kterých může pracovat a to dvojkový a decimální. Hodnota indikačního registru decimální aritmetiky D, ve stavovém registru F, určuje v jakém režimu je procesor. Jestliže je hodnota rovna 1, budou všechny operace v decimálním režimu a naopak. Všeobecně se v assembleru užívá většina operací dvojkovou matematikou.

B - Indikační registr zastavení, neboli registr B, muže být nastaven a vynulován jen samotnou CPU 6510. Programátor příznak B měnit nemůže. Používá se pro určení toho, zda bylo přerušení způsobeno instrukcí BRK.

V - Jestliže pracujete ve znaménkové matematice, potřebujete nějakým způsobem určit, zda došlo k přečtení do znaménkového bitu. K tomu právě používáme indikační registr přečtení. Jestliže příznak V obsahuje 1, došlo k přečtení do znaménkového bitu a naopak. Je důležité věnovat tomuto bitu při práci se znaménkovou matematikou pozornost a vzít do úvahy druh programu, abychom s takovýmto přečtením správně jednali.

N - Posledním bitem ve stavovém registru F je indikační registr znaménka N. Jestliže je tento bit rovný 1, předchozí operace vedla k zápornému výsledku, a jestliže je N rovno 0, výsledek byl buď kladný nebo rovný 0. Jestli je číslo kládné nebo rovno nule, můžeme určit z příznaku Z.

Instrukce uložení

Do této skupiny zahrnujeme celkem 3 instrukce. Jsou to základní instrukce, které přiřazují registrům určitou hodnotu.

LDA Ulož hodnotu do akumulátoru.
LDX Ulož hodnotu do registru X.
LDY Ulož hodnotu do registru Y.

Pro vysvětlení funkce si zvolíme následující příklad. Jestliže například chceme zjistit obsah adresy 0243 H, což z BASICU uděláme příkazem PEEK, použijeme právě výše uvedené instrukce.

LDA 0243 H

Tento příkaz okopíruje obsah adresy 0243 H do akumulátoru, přičemž obsah adresy 0243 H se nemění. Toto je nejjednodušší způsob, jak načíst obsah libovolné adresy do akumulátoru. Existuje však ještě 6 dalších druhů adresace, ale o nich si povíme až příště. Pokud chceme akumulátor naplnit přímo určitým číslem, použijeme také tuto instrukci s tím rozdílem, že před číslo dáme znak #. Tímto jasně říkáme, že číslo které následuje, není adresa, ale hodnota.

LDA # 25 H

Tento příkaz uloží do akumulátoru hexadecimální číslo 25. Toto číslo může být na rozdíl od adresy však pouze jednobytové. Všechny výše uvedené operace lze samozřejmě provádět i s registry X a Y.

POPIS ZABUDOVANÝCH RUTIN

V Kernalu je uloženo několik programů, které se obvykle používají jako podprogramy. Používá je BASIC, ale můžeme je používat i my. Zrovna tak je můžeme i předělat a na jejich počátek přednastavit příslušný vektor. Tabulka skoků pro

jednotlivé rutiny je uložena na konci Kernalu od $FF81 do $FFF5. Protože jejich použití je celkem výhodné, uvedeme si jejich základní popis.

ACPTR: čtení Bytu ze sériové bus

používá registr : A,X
komunikační reg.: A
Zakazuje přerušení, v A je Byte přečtený ze sběrnice.
Příklad: čtení Byte ze sériové sběrnice
JSR ACPTR
STA DATA

CHKIN: příprava souboru pro vstup

používá registr : A,X
komunikační reg.: X
Nejprve zjišťuje, byl-li již soubor otevřen, jestliže ano, pak přenese aktuální data na $B8, $B9, $BA. Následuje test, může-li být soubor vstupní.
Příklad: připrav pro vstup logický soubor 2
LDX#$02
JSR CHKIN

CHKOUT: soubor pro výstup

používa registr : A,X
komunikační r. : X
Obdoba CHKIN. Používá některé stejné podprogramy jako CHKIN.
Příklad: připrav pro výstup log. soubor 4
LDX#$04
JSR CHKOUT

CHRIN: vstup znaku

používá registr : A,X
komunikační r. : A
Nejprve přečte kód vstupního zařízení. Pokud je periferie určena pro vstup, pak z ní čte data zakončená např. $0D (RETURN).

Příklad: příprava Y pro ukládání dat
JSR CHRIN
STA DATA,Y: uložení dat
INY
CMP#$0D : test na RETURN
BNE(JSR CHRIN): není-li, pak vše opakuj od JSR

CHROUT: vyslání znaku

používa registr : A
komunikační r. : A
Pomocí této rutiny mužeme vyslat na obrazovku nebo na periferii kód uložený v A.
Příklad: LDA#$0D : kód RETURNU
JSR CHROUT : odrolovat řádky na obrazovce

CIOUT: vyslání znaku na sériový BUS

používá registr : A
komunikační r. : A
Tato rutina obstarává vyslání znaku na sériovou sběrnici.
Příklad:
LDA#$41 : kód znaku A
JSR CIOUT : vyslání

CINT: inicializace editoru a VIC

používá registr : A,X,Y
Nastaví standardní vstup a výstup, nastaví registry VIC, nastaví standardní grafický mód, blikání kurzoru, dekódovací rutinu klávesnice na $EB47, velikost klávesnicového bufferu na $0A, rychlost odečítání znaku, barvu znaku $0286 na $0E, pomocné ukazatele řádek, další SP editoru, PAL, NTSC, přerušovací registr na adrese $DC0D a další.
Příklad: JSR CINT : nastavení editoru a VIC
JMP RUN

CLALL: uzavření všech souborů

používá registr : A,X
Uzavře všechny soubory a nastaví standardní vstup a výstup.
A=$00,X=$03

Příklad: JSR CLALL
JMP RUN

CLOSE: uzavření daného souboru

používá registr : A,X,Y
komunikační r. : A
Tato rutina uzavře soubor s daným číslem a přerovná tabulku kanálu.
Příklad: uzavření souboru číslo $15
LDA#$15
JSR CLOSE

CLRCHN: nastavení standard. in/out

používá registr : A,X
Nejprve porovná výstup na $03. Je-li kód periferie menší, volá UNTALK, jinak UNLSN. Pak nastaví st. vstup a výstup.
Příklad:
JSR CLRCHN

GETIN: čtení znaku

používá registr : A
komunikační r : A
Nejprve zjistí, z jaké periferie má znak odečítat. Je-li to modém, odečte znak z jeho bufferu, přičemž C=0 a Y se zachovává. Je-li vstupem klávesnice, pak odečte znak z jejího bufferu. V A a Y je kód tohoto znaku a v X je počet znaků, které v bufferu ještě zbyly. Je-li vstupem něco jiného, pak končí skokem do CHRIN na adresu $F166
Příklad: odečtení znaku z klávesnice
JSR GETIN
CMP#$00 : klávesa není stisknuta?
BEQ(JSR GETIN) ne, opakuj vše od JSR GETIN

IOBASE: zjistí počátek r. CIA#1

používá registr : X,Y
komunikační r : X,Y
Tato rutina přečte do X, Y Lo, Hi adresu počátku CIA#1.

IOINIT: inicializace in/out

používá registr : A,X,Y
Nastaví přerušovací registr CIA#1, dataport A atd., totéž u CIA#2, vynuluje SID registry a pak pokračuje na $FDD.

LISTEN: nastavení periferií na příjem

používá registr : A
komunikační r. : A
Tato rutina vyšle na periferii patřičné kódy, podle kterých se nastaví na příjem.
Příklad: nastav periferii s kódem $08 na příjem
LDA#$08
JSR LISTEN

LOAD: naplnění RAM z periferie

používá registr : A,X,Y
komunikační r. : A,X,Y
Podle kódu v A se provádí buď LOAD nebo VERIFY (A=1). Do X, Y ukládáme Lo, Hi adresu možného počátku nahrávání. Po LOAD tam máme adresu konce nahraných dat.
Příklad: přečti data z mag.
LDA#$01 : 1 - magnetofon
LDX#$00 : log. číslo souboru
LDY#$00 : sec. adresa
JSR SETLFS : nastavovací rutina
LDA#$0F : počet znaků názvu souboru
LDX#$Lo
LDY#$Hi : počátek názvu
JSR SETNAM : nastavení par. názvu
LDA#$00
LDX#$01
LDY#$08 : počátek ukládání programu
JSR LOAD
STX$2D
STY$2E : nastavení konce po LOAD BASICU
JMP RUN

MEMBOT: operace s počátkem BASIC RAM

používá registr : X,Y

komunikační r. : X,Y
Tato rutina je naprosto shodná s MEMBOT, jenom s tím rozdílem, že se jedná o konec paměti RAM pro BASIC.

OPEN: otevření souboru

používá registr : A,X,Y
Tato rutina provede otevření příslušného souboru.
Příklad:

```
LDA#$00      : žádný název
JSR SETNAM   : nastavení jména
LDA#$15      : číslo souboru
LDX#$08      : kód periferie
JSR SETLFS   : nastavení parametrů souboru
JSR OPEN     : otevření souboru
```

PLOT: poloha kurzoru

používá registr : A,X,Y
komunikační r. : X,Y
Je-li C=1 pak do X, Y přečte pozici kurzoru, jinak při C=0 na tato místa naopak uloží obsah X a Y. V X je řádek, v Y je sloupec.
Příklad: nastavení kurzoru na pozici $05,$10

```
LDX#$10
LDY#$05
CLC
JSR PLOT
```

RAMTAS: inicializace RAM

používá registr : A,X,Y
Nuluje oblasti $0002 - $0101, $0200 - $02FF, $0300 - $03FF, pak nastavuje $B2, $B3 na $033C, $C1, $C2 na $0400 a inicializuje tuto oblast, nastaví počátek a konec BASIC RAM.

RDTIM: čtení času od zapnutí

používá registr : A,X,Y
komunikační r. : A,X,Y
Zakazuje přerušení. Pak přečte do A $A2 , do X $A1 a do Y $A0. Pak přerušení povoluje.

READST: čtení stavu r. in/out

používá registr : A,X,Y
komunikační r. : A,X,Y
Tato rutina čte stavový registr obvodu vstupu a výstupu, tedy v podstatě obsah adresy $0287 nebo $90.
Příklad: kontrola na konec během čtení
JSR READST
AND$#40
BNE(KONEC) : skok při konci souboru

RESTOR: nastavení vektoru in/out

používá registr : A,X,Y
Tato rutina nastaví $C2,$C3 na $FD30 a dále pak vektory v oblasti $0314 - $0333.

SAVE: uložení dat z RAM

používá registr : A,X,Y
komunikační r. : A,X,Y
Tato rutina uloží do periferie data, která jsou v RAM. Pokud je před zavoláním A nenulové, pak ukazuje na Lo adresu v nulové stránce, která ukazuje na počátek nahrávání dat. Tento mód má tu výhodu, že při LOAD se data nahrají zpět na totéž místo, ze kterého byla zaznamenána (auto-save).

SCNKEY: příprava pro čtení znaku

používá registr : A, X, Y
Tuto rutinu přednastaví SP pro čtení znaku z klávesnice do výchozího stavu, pak čte kód stisknuté klávesy a porovnává ho na STOP, nastavuje správnou dekódovací tabulku, dekóduje některé znaky a podle toho o které jde, buď změní příslušné SP nebo znak zapíše do bufferu. Poměrně často se vyskytuje před rutinou GETIN pro vyvolání a inicializaci SP.

SCREEN: formát textové obrazovky

používá registr : X, Y
komunikační r. : X, Y

Do X načte hodnotu #$28 a do Y #$19. Pak se vrací. Tyto hodnoty odpovídají formátu 40*25.

SECOND: vysílání sekundární adresy

používá registr : A
komunikační r. : A
Tato rutina vyšle sekundární adresu obsaženou v A.

SELFS: uložení kódu int/out

používá registr : A, X, Y
komunikační r. : A, X, Y
Tato rutina zapíše číslo souboru na $B8, kód periferie na $BA a sekundární adresu na $B9.

SETMSG: obsluha hlášek

používá registr : A
komunikační r. : A
Tato rutina zapíše A na $90. Je-li A=$40, pak se tisknou jen řídicí hlášky, A=$80 chybové hlášky, A=$00 nic, hlášky jsou odpojené. BASIC je však vždy zapíná!

SETNAM: obsluha hlášek

používá registr : A, X, Y
komunikační r. : A, X, Y
Po zavolání přepíše A na $87, X na $88 a Y na $BC. Používá se před OPEN, LOAD, SAVE apod.

SETTIM: nastavení systémových hodin

používá registr: A, X, Y
komunikační r. : A, X, Y
Nejprve zakáže přerušení. Pak přepíše jednotlivé registry na adresy: A na $A2, X na $A1 a Y na $A0. Pak přerušení opět povolí a vrací se. Nejvyšší hodnotu na A, nižší X a nejnižší Y. Jednu minutu zapíšeme takto: 1 min. = 60 sec. = 60 * 60 vteřin. Tuto hodnotu přepíšeme do hexa. tvaru a rozdělíme ji do jednotlivých registrů.

SETTMO: ukazatel pro IEEE-Bus

používá registr: A
komunikační r. : A
Hodnotu z A přepíše na $0285.

TALK: periferie pro vyslání

používá registr: A
komunikační r. : A
Nastavení periferií s kódem, který je v A na vysílání.
Přenos je po sériové sběrnici.

UNLSN: povel pro sériovou Bus

používá registr: A
Vyšle povel před novým LISTEN.

Všechny tyto rutiny můžeme libovolně používat, ale je třeba vědět co udělají, aby nedošlo ke zhroucení nebo zakousnutí systému.

Úvod do užívání Simons Basicu

Po nahrání SIMONS BASICU a jeho odstartování se systém ohlasí:

xxx **EXPANDED** CBM V2 BASIC xxx
30719 BASIC BYTES FREE

Nyní má uživatel k dispozici přes 100 nových příkazů BASIC. V tomto návodu je každý příkaz vysvětlen.

Příkazy, které usnadňují programování

SIMONS BASIC nabízí různé příkazy, které usnadňují sestavování a odlaďování programů. Přitom je lhostejno, zdali vytvářený program dále pracuje s příkazy nebo ne.
"KEY" umožňuje obsadit funkční tlačítka C64

"DISPLAY" zpětně listuje příkazy na funkčních tlačítkách

"AUTO" a "RENUMBER" přejímají automatické číslování řádku

"MERGE" připojí k programu v paměti program uložený na vnějším médiu

Příkazy pro užívání funkčních tlačítek

KEY, číslo (1-16), "příkaz" (max. 15 znaků)
Cíl: obsadit funkční tlačítka příkazem a toto obsazení libovolně měnit. Číslo ve formátu je identické s číslem funkčního tlačítka. Příkaz, kterým chceme tlačítko obsadit, musí být v uvozovkách a nesmí být delší než 15 znaků. Kódování jednotlivých funkčních tlačítek:

```
F1,F3,F5,F7.....stiskem F1,F3,F5,F7
F2,F4,F6,F8.....stiskem SHIFT F1,F3,F5,F7
F9,F11,F13,F15....stiskem C= F1,F3,F5,F7
F10,F12,F14,F16...stiskem SHIFT C= F1,F3,F5,F7
```

Automatické provedení příkazu

Mnohdy potřebujete některé příkazy ihned uzavřít RETURNEM. Při tom je příkaz současně vykonán. K provedení je třeba připojit +CHR$(13) a RETURN.

Po stisknutí funkčního tlačítka je příkaz zobrazen a vzápětí vykonán.
Příklad: KEY 7, "LIST" +CHR$(13) /RETURN/
Výsledek: Po stisknutí tlačítka je příkaz proveden, tj. program se vypíše na obrazovku.

DISPLAY

Cíl: Zobrazit osazení funkčních tlačítek.
Příklad: DISPLAY /RETURN/

```
KEY 1,"......."
KEY 2,"......."
 :
 :
 :
KEY 16,"......"
```

AUTO

Automatické číslování řádků s předvolbou šířky kroku.

Po zadání příkazu AUTO se na obrazovce objeví číslo prvního řádku. Nyní mohou být zadány libovolné řádky v BASIC a na obrazovce se objeví vždy nové číslo řádku v požadovaném rozestupu.

```
Příklad: AUTO 10,5 /RETURN/
         10 PRINT"TEST",
         15 GOTO 10
         20
```

Po zadání RETURN zrušení AUTO.

RENUMBER

Cíl: přečíslování programových řádek.

RENUMBER přebírá automatické přečíslování programových řádek. Program začíná po provedení příkazu prvním číslem, které je zadáno. Další čísla řádků jsou zvýšena o příslušnou šíři kroku. Tento příkaz je zvlášť vhodný, potřebujeme-li vložit do programu další řádky.
Nevýhodou je, že příkaz nemění skokové adresy příkazu GOTO a GOSUB.

```
Příklad:1 PRINT "     "
        2 FOR X=0 TO 15
        3 POKE 53280,X
        4 NEXT X

RENUMBER 100,10 /RETURN/
        100 PRINT "     "
        110 FOR X=0 TO 15
        120 POKE 53280,X
        130 PAUSE 1
        140 NEXT X
```

PAUSE

Cíl: Zpoždění prováděného programu.

PAUSE umožňuje vložit do určitého místa programu určité zpoždění. Tato pauza může být stiskem RETURN přerušena. Současně s probíháním příkazu je možné zobrazit komentář.

```
Příklad:
105 PAUSE 10
135 PAUSE"stisknete RETURN, aby program pokračoval ",10
```

LIN

Cíl: určení řádku obrazovky, ve kterém se nachazí kurzor.
Priklad:10 PRINT " "
        20 FOR I=1 to 5
        30 : A=LIN
        40 : ?LIN,A
        50 NEXT I

C GOTO

Cíl: skok na zavislý řádek.
     Příkaz umožňuje skok na řádek, který se určí až při běhu programu.
Příklad:

```
10 FOR I=1 TO 5
20 : C GOTO Ix10+130
30 END
40 ?"I=1" : NEXT I
50 ?"I=2" : NEXT I
60 ?"I=3" : NEXT I
70 ?"I=4" : NEXT I
80 ?"I=5" : NEXT I
```

RESET

Cíl: nastavení ukazatele dat na první hodnotu určitého řádku dat.
     V standardním BASICU jsou data čtena jedno po druhém. V podstatě tam existuje jen možnost příkazem RESTORE ukazatel postavit na první hodnotu prvního návěstí dat. RESET umožňuje nastavit ukazatel na první hodnotu určitého řádku dat. Při dalším příkazu RESET je tato hodnota přečtena a ukazatel dat posunut o jeden řádek vzad.

MERGE

Cíl: Nahrání programu, který byl předtím uložen na pásku či disketu, do pracovní paměti počítače. Program v pracovní paměti zůstane přitom zachován. Užívají se stejné názvy programu a stejná čísla jako ve spojení s LOAD.
Pozor! Při použití příkazu MERGE nesmějí být v programu, který je natahován, a v programu v pracovní paměti stejná

čísla řádků. Proto se doporučuje program v paměti přečíslovat pomoci RENUMBER.

Pomoc při listování v programu

PAGE

Cíl: dělení programu do odstavců po n+1 řádcích. PAGE umožňuje předvolit počet řádků na obrazovce. Poté, co byl proveden tento příkaz, zobrazuje LIST přesně n+1 řádků na obrazovce. Každý další odstavec je nalistován po stisku RETURN. Při n=0 nepůsobí příkaz PAGE na příkaz LIST. Listování muže být přerušeno RUN/STOP.
Pozor! Parametr n se vztahuje k řádkům obrazovky, nikoli k programovým řádkům.

OPTION

Cíl: zdůraznění všech nových příkazů SIMONS BASICU.
OPTION s parametrem 10 způsobí inverzní zobrazení příkazu při listování programem. Libovolná jiná hodnota zase zvýraznění zruší.

DELAY

Cíl: mění rychlost, se kterou se provádí listing.

hodnota	rychlost znak/sec
1	290
5	100
10	50

Příklad: 1 PAGE 10 : DELAY 5
K tomu se zadá před příkazem LIST příkaz DELAY n /RETURN/. Parametr n může nabývat hodnot od 1 do 255. n=1 je nejvyšší rychlost, n=255 je nejnižší rychlost. Po LIST /RETURN/ je program listován zadanou rychlostí. Při stisku SHIFT. Stiskem CTRL se dá postup listování ještě zpomalit nebo přerušit stiskem C=.

FIND

Cíl: vyhledávání BASIC kódu, případně řetězce a zobrazení čísel řádků ve kterých se nachází. Příkaz FIND prohledává program podle zadaných BASIC kódů či řetězců a ukazuje na obrazovce ta čísla řádků, ve kterých se nachází.

Pomoc při odstraňování chyb.

TRACE

Cíl: zobrazení čísla řádku, který se zpracovává.

Příkaz TRACE se zadává před startem programu. Při běhu programu se objeví v pravém horním rohu okénko, ve kterém se zobrazuje posledních šest prováděných řádků. Čísla řádků se přepisují automaticky, takže nejspodnější odpovídá vždy naposled provedenému BASIC řádku. Čísla řádků nejsou viditelná v režimech HIRES a MULTICOLOR. Okénko překrývá normální obsah obrazovky. Příkaz TRACE je zrušen zadáním TRACE 0.

RETRACE

Cíl: obnovení okénka s TRACE.

Je-li při práci s příkazem TRACE program přerušen a obrazovka vymazána, potom může být pomocí příkazu RETRACE obnovené okénko s posledními řádky opět zobrazeno.

DUMP

Cíl: vyhledání proměnných v programu a jejich okamžitých hodnot.

Po příkazu DUMP jsou vyhledány všechny proměnné se svými okamžitými hodnotami v tom pořadí, jak po řadě přicházejí. Vyhledání je možno zpomalit stiskem CTRL.

COLD

Cíl: převedení C64 do výchozího stavu

Po COLD je počítač přenesen do výchozího stavu a hlásí se jako po natažení SIMONS BASIC.
Pozor! Program v pracovní části paměti je ztracen. Může se zachránit jen tehdy, jestliže se předtím zadá nový BASIC řádek, provádějící příkaz OLD.

Ochrana programů

SIMONS BASIC dává 2 příkazy, které chrání určité řádky programu. Tím zabraňuje, aby nepovolané osoby přepsaly program.

Pozor! Opravu potom není možné provést jinak, než novým zadáním chráněných řádků. Proto radím pro vlastní použití uchovat jednu kopii nechráněnou.

DISAPA

Cíl: ukazuje, že programový řádek má být chráněn.

DISAPA je používán jako příkaz v prog. řádku, aby ukazoval, že tento má být chráněn. Automaticky nastavuje čtyři dvojtečky před následujícím příkazem.

SECURE

Cíl: ochrana všech programových řádků v nichž je uveden DISAPA.

Příkaz SECURE způsobí, že kód v programových řadcích za příkazem DISAPA je chráněn. Programový řádek je normálně proveden.

Příklad:

```
100 REM DISAPA A SECURE
135
140 INPUT "CODE ":A
145
150 PRINT "CODE OK"
```

OLD

Cíl: zrušení příkazu NEW.

OLD umožňuje, aby program, který byl příkazem NEW vymazán, byl vrácen zpět a proveden / ukazatel START a END OF BASIC jsou nastaveny zpět/.

Pozor! Příkaz je účinný jen do té doby, dokud není zadán další programový řádek.

Operace s řetězci:

SIMONS BASIC umožňuje rozsáhlé operace s řetězci, formátování výrazu na obrazovce a příkazu INPUT. Ve spojení se standardními řetězci je tím umožněno mnohonásobně kontrolovat řetezce. Všechny příkazy mohou být použity při editaci programu.

INSERT

Umožňuje zavedení jednoho řetězce do jiného, přičemž vznikne řetězec nový.

```
A$=INSERT("IN","COMMODORE",9)
PRINT A$
```

Zobrazí: COMMODORE IN

PRINT A

Je-li podmínka splněna, je výsledek 1 (pravda), když ne, je výsledek 0 (nepravda).

INST

Přepíše řetězec od určité pozice jiným řetězcem. Parametr udává, od jaké pozice má být řetězec přepsán.

PLACE

Vyhledává v řetězci určitou skupinu znaků. Výsledkem je celočíselná hodnota, která udává pozici prvního hledaného znaku. Není-li znak nalezen, je výsledek 0.

CENTRE

CENTRE přenese řetězec přesně doprostřed určitého řádku obrazovky. Retězec přitom nesmí být delší než 39 znaků. Mají-li po sobě následovat dva příkazy CENTRE, musí být mezi ně vložen příkaz PRINT.

USE

USE umožňuje vyjadřování numerických hodnot v předem daném formátu. Každé číslo představuje přitom jeden řád před nebo po desetinné čárce. Též je možné do formátovaného řádku vložit text. Formátované údaje mohou být zobrazeny ve formě řetězcové proměnné. Číslo, které má být vytvořeno, musí být zobrazeno jako řetězcová proměnná, tj. číslo musí být před vydáním příkazu USE převedeno na řetězec pomocí příkazu STR$.

AT

AT umožňuje řetězci určitou pozici na obrazovce. Přitom udává sloupec (0-39) a řádek (0-24). Příkaz AT se může použít vícekrát za sebou.

PETCH

PETCH umožňuje při zadávání z klávesnice jen určité znaky a jejich počet omezit. Řídící klávesa za uvozovkami přitom určuje, který znak může být zadán, a udává počet znaků, který může být nejvýš zadán. Zadaný řetězec nebo numerická hodnota jsou uloženy do proměnné, jejíž název se musí zadat.

INKEY

Dotaz, které funkční tlačítko bylo stisknuto. Výsledek je číslo funkční klávesy nebo 0 v případě, že nebyla stisknuta žádná funkční klávesa.
Příklad:

```
100 A-INKEY
110 ON A GOSUB 1000,2000
120 GOTO 100
1000 PRINT "F1 bylo stisknuto":RETURN
2000 PRINT "F2 bylo stisknuto":RETURN
```

Podle stisknuté klávesy se objeví odpovídající výsledek.

ON KEY

Příkaz ON KEY přiměje C64 k tomu, aby ohledal klávesnici a zjistil, zda byla stisknuta klávesa uvedená v řetězci znaků. Stisk každé jiné klávesy je ignorován. Je-li stisknuta klávesa uvedená v řetezci, pokračuje program v místě podmíněného skoku (GOTO). Obsazené proměnné ST uchovávají hodnotu CHR$ klávesy, která byla stisknuta. Příkaz je zvlášť vhodný pro programy uváděné výběrem menu.
Pozor! Je-li použit příkaz ON KEY, ohledává C64 klávesnici tak dlouho, dokud není stisknuta klávesa z uvedené množiny znaků. Pro zrušení se používá příkaz DISABLE.

DISABLE

Zrušení příkazu ON KEY. Ohmatání klávesnice, které je zadáno pomocí příkazu ON KEY, je pomocí DISABLE zrušeno.

RESUME

Aktivování předchozího příkazu ON KEY. Příkazem RESUME se vyvolá naposled aktivovaný příkaz ON KEY. Potom čeká program znovu na stisk klávesy znaku z daného řetězce.

Pojednání o číslech

Tento odstavec obsahuje různé příkazy, které usnadňují práci s čísly. K tomu patří příkazy pro výpočty, určování desetinných míst, převádění čísel mezi číselnými soustavami, jakož i Booleovou algebrou.

Přídavné aritmetické operace

MOD

Cíl: určení zbytku po číselném dělení.
Příkaz MOD převádí parametry X a Y nejprve na dvě čísla typu INTEGER (bez zaokrouhlení). Poté je provedeno dělení. Výsledkem je číslo, které představuje zbytek po dělení. Příkaz může být použit přímo nebo v programu.
Příklad:
PRINT MOD(11,4) /RETURN/
3
Výsledek: 11:4=2 zbytek 3

DIV (X, Y)

Cíl: určení výsledku celočíselného dělení.
Příkaz DIV převádí parametry X a Y na dvě čísla typu INTEGER a provádí dělení. Výsledkem je číslo před desetinnou čárkou. Příkaz může být použit přímo nebo v programu.
Příklad:

```
PRINT DIV(11,4) /RETURN/
2
```
Výsledek: 11:4=2 zbytek 3

FRAC (n)

Cíl: určení čísla za desetinnou tečkou.
Příkaz FRAC určuje místa za desetinnou čárkou u čísla s pohyblivou desetinnou čárkou. Přitom je zobrazeno max. 9 míst za desetinnou čárkou.
Příklad:
```
PRINT FRAC(11/3) /RETURN/
```
Výsledek: 0.666 666 667

Převádění čísel

% - binární na decimální

Cíl: převedení binárního čísla na decimální.
Příkaz převádí zadané binární číslo na dekadické. Binární číslo musí mít 8 cifer a musí obsahovat pouze 1 a 0. Je-li jiný formát čísla, objeví se chybové hlášení ?NOT BINARI CHAR.
Příklad:
```
PRINT%01011010 /RETURN/
90
```

$ - hexadecimální na decimální

Cíl: převedení hexadecimálního čísla na dekadické.
Příkaz $ převádí žádané haxadecimální číslo na dekadické. Hexadecimální je čtyřmístné, složené z číslic 0-F. Nejsou-li při zadávání splněny všechny podmínky, následuje chybové hlášení.
Příklad:
```
PRINT $ABCD /RETURN/
43981
```

EXOR (n, n1)

Cíl: provedení operace OR EXCLUSIVE.
Příkaz provádí operaci EXOR mezi dvěma čísly. Zadaná

čísla jsou převedena na binární a potom bit po bitu porovnávána. Výsledek je převeden na dekadické číslo. Oba parametry n mohou nabývat hodnot 0-65535. Srovnávat se mohou i čísla jiných číselných soustav.

Operace s disketami

SIMONS BASIC má dva příkazy pro operace s disketami, které vyrazně usnadní práci.

DISK

Cíl: otevření kanálu, provedení návěstí, uzavření kanálu.
Příkaz DISK nahrazuje následující příkazy jazyka BASIC: OPEN, číslo file, číslo přístroje, sekundární adresa: PRINT číslo file. CLOSE.
Příklad: nové formátování disket.
DISK"NO : SIMONS BASIC, 01" /RETURN/
Výsledek: za několik minut je disketa naformátována.

DIR"$:řetězec

Cíl: přečtení obsahu direktory nebo její části.
Příkaz DIR nahrazuje příkaz LOAD "$", 8. Přitom je možno direktory prohlédnout, popř. prohlédnout jen názvy některých file.
Příklad:
DIR"$ /RETURN/
Výsledek: je prohlédnutá celá direktory.

Grafika SIMONS BASICU

Tento odstavec popisuje grafické příkazy SIMONS BASICU: První část odstavce popisuje stavbu obrazu na stínítku obrazovky a vysvětluje rozdíl mezi HIRES a MULTICOLORU. Dále jsou ukázány možnosti C64 v barvě a je popsáno; jak je určena barva grafiky. Poslední část se zabývá grafickými příkazy a možnostmi grafiky.

Stavba obrazovky

Při využívání televizní obrazovky pro grafická znázornění je obrazovka rozdělena do matice 320x200 bodů. Každý bod má X a Y souřadnici. Levý horní roh obrazovky má souřadnice 0,0. HIRES grafika shodně dělí obrazovku na 320x200 bodů. MULTICOLOUR ji naproti tomu dělí na 160x200 bodů. Vzdálenost dvou bodů v X směru je oproti HIRES grafice dvojnásobná.

Barvy

Commodore C64 umožňuje vybrat si mezi 16 barvami. V matici 8x8 bodů mohou být zvoleny až tři různé barvy. Každé barvě je přiřazeno určité číslo:

0-černá	8-oranžová
1-bílá	9-hnědá
2-červená	10-světle červená
3-tyrkysová	11-šedá 1
4-fialová	12-šedá 2
5-zelená	13-světle zelená
6-modrá	14-světle modrá
7-žlutá	15-šedá 3

Způsob vyznačení

Vsechny příkazy SIMONS BASICU týkající se grafiky mají společnou jednu vlastnost: znak, udávající způsob značení. Znak určuje, jak má být bod na obrazovce znázorněn.

a) při HIRES

znak	funkce
0	vymaže bod
1	vyznačí bod
2	invertuje (vyznačený bod vymaže a obráceně)

b) při MULTICOLOUR

znak	funkce
0	vymaže bod
1	vyznačí bod v barvě 1
2	vyznačí bod v barvě 2
3	vyznačí bod v barvě 3
4	inverze bodu

Pozn. k inverzi bodu:

Bod pozadí bude zobrazen v barvě 3.
Bod barvy 1 bude zobrazen v barvě 2.
Bod barvy 2 bude zobrazen v barve 3.
Bod barvy 3 bude zobrazen v barvě pozadí.

Grafické příkazy

HIRES zf, hg

Cíl: určení barvy znaku a pozadí.

Příkaz HIRES přepíná obrazovku do vysoce rozlišitelné grafiky. Každý bod obrazovky může být nyní jednotlivě osazen (v matici 320x200 bodů). Parametr zf určuje barvu bodu. Parametr hg je vyhrazen barvě pozadí, do kterého je bod zasazen.

Příklad:

```
100 HIRES 0,1
1000 GOTO 1000
```

Výsledek: barva bodu je černá a pozadí bílé.

REC x, y, s1, y1, bod

Cíl: nakreslit obdélník.

Příkaz REC kreslí obdelník na obrazovce. Oba první parametry X a Y určují souřadnice levého horního rohu obdélníku. Parametr x1 a y1 udávají délku stran.

Příklad:

```
10 REM OBDELNIK
20 HIRES 2,1,
30 FOR X=5 TO 125 STEP 5
40 : REC x,1.3x x,320-5/2x x ,120-9x x,1
50 NEXT X
60 PAUSE 5
```

MULTI c1, c2, c3

Cíl: nastartovat MULTICOLOUR mód a určit tři barvy značek.

Přijde-li po příkazu HIRES příkaz MULTI, nastartuje se grafika v režimu MULTICOLOUR: kreslený bod má proti bodu v módu HIRES dvojnásobnou šíři (proti bodu v módu HIRES).

Parametry c1, c2, c3 pevně stanovuji tři barevné možnosti

LOW COL c1,c2,c3

Cíl: změna barev.

Příkaz LOW COL umožňuje přiřadit jednomu příkazu pro kreslení další tři možné barvy. Parametry c1-c3 tyto barvy určují.

Pozor! Všechny tři parametry musí být zadány také v grafice HIRES.

HI COL

Cíl: vrácení barev daných v MULTI.

Po příkazu HI COL mohou být barvy, které jsou prvně dány příkazem MULTI, opět použity (ve spojení s typem znak - bod).

PLOT x, y, bod

Cíl: nakreslit bod.

Příkazem PLOT se na obrazovce vyznačí bod. Parametry x a y udávají souřadnice bodu.

Příklad:

```
10 REM PLOT
20 HIRES 0,1
30 FOR X=0 TO 320 STEP 0.5
40 : Y = 100+SIN(X/34)*50
50 : PLOT X,Y,1
60 : PLOT X,100,1
70 NEXT X
1000 goto 1000
```

TEST (x, y)

Cíl: určit polohu bodu.

Příkaz TEST určuje polohu bodu na obrazovce. Pomocí parametrů x a y jsou dány souřadnice testovaného bodu. V- Výsledek je 1, je-li na testovaném mís bod vyznačen. V opačném případě je výsledek 0.

LINE x, y, x1, y1, bod

Cíl: nakreslit úsečku.

Příkaz LINE kreslí úsečku mezi počátečními body x a y a koncovými body x1 a y1.
Příklad:

```
10 REM LINE
20 HIRES 1,6
30 FOR X=0 TO 320 STEP 8
40 : M=100/320*X
50 LINE 320-X,M,X,100+M,1
60 NEXT X
70 GOTO 70
```

CIRCLE x, y, x1, y1, bod

Cíl: narkreslit elipsu.
Pomocí příkazu CIRCLE je možno nakreslit elipsu. Parametry x a y určují střed elipsy, x1 poloměr ve směru x a y1 poloměr ve směru y.
Pozor! Kružnice je zvláštní formou elipsy. Aby kružnice vznikla, musí být splněny následující podmínky.

HIRES	x-rádius=1.15 x,y-rádius
MULTI	x-rádius=0.575x,y-rádius
Tiskárna	x-rádius=y-rádius

ARC x, y, sa, se, i, x1, y1, bod

Cíl: nakreslit oblouk.
Příkaz ARC umožňuje nakreslit oblouk. Parametry mají následující význam:
x, y - střed elipsy (kružnice), z níž má být oblouk nakreslen
sa - počáteční úhel
se - koncovy úhel
i - vzdálenost úhlu, která se vypočítává
x1 - rádius
y1 - rádius obrazu, jehož oblouk má být nakreslen

ANGL x, y, úhel, x1, y1, bod

Cíl: nakreslit rádius.
Příkaz ANGL umožňuje nakreslit rádius elipsy. Elipsa přitom nemusí být na obrazovce. Parametry mají následující význam:

x, y - střed elipsy
úhel - úhel pod kterým paprsky stojí
x1, y1 - poloosy elipsy, jejíž paprsky se mají kreslit

Příklad:

```
10 REM ANGL
20 HIRES 2,1
30 FOR X=5 TO 358 STEP 3
40 ANGL 220,125,x,x/1,4,x/3.8,1
50 NEXT X
60 GOTO 60
```

PAINT x, y, barva

Cíl: vyplnit barvou uzavřenou plochu.

Příkaz PAINT vyplní uzařenou plochu barvou. Barva je určena příslušným kódem. Parametry x a y udávají souřadnice bodu uvnitř barevné plochy. Plocha může být barvou vyplněna vícekrát.

BLOCK x,y,x1,y1,bod

Cíl: vyplněný obdélník.

Příkaz BLOCK je výhodnější možnost pro nakreslení plného obdélníku. Rozhodující je rychlost, se kterou se obdélník kreslí. Parametry x a y určují souřadnice levého horního rohu, x1 ohraničuje blok ve směru x a y1 ve směru y.

Příklad:

```
10 REM BLOCK
20 HIRES 2,2:MULTI 2,1,5
30 X=10
40 NEPAT
50 FOR F=1 TO 3
60 DY=RND(1)*140
70 BLOCK*10+dy,x+10,160,f
80 X=X+8
90 NEXT F
100 UNTIL*120
110 GOTO 110
```

DRAW řetězcová proměnná, x, y, bod

Cíl: nakreslit obrazec.

Příkaz DRAW umožňuje načrtnot libovolný obrazec a ten potom nakreslit na obrazovku. Obrazec vznikne podobně jako při kreslení na papír. Číframi (n) se zadá, jakým směrem se má tužka pohybovat a má-li být stopa viditelná či nikoli.

Význam cifer:

0 - pohyb vpravo - neviditelný
1 - pohyb nahoru - neviditelný
2 - pohyb dolů - neviditelný
3 - pohyb vlevo - neviditelný
4 = 2
5 - pohyb vpravo - viditelný
6 - pohyb nahoru - viditelný
7 - pohyb dolů - viditelný
8 - pohyb vlevo - viditelný
9 - konec obrazce

Cifry mohou být zadány přímo v uvozovkách nebo jako řetězce. Parametry x a y udávají počáteční souřadnice.

NOT r, s

Cíl: pootočit obrazec.

Příkaz NOT umožňuje pootočit o určitý úhel obrazec, který byl načrtnut příkazem DRAW. Tento úhel je určen parametrem r.

r.....úhel rotace	
0.....0 stupňů	1.....45 stupňů
2.....90 stupňů	3.....135 stupňů
4.....180 stupňů	5.....225 stupňů
5.....270 stupňů	6.....315 stupňů

Parametr n musí ležet v rozsahu 0 až 7. Druhý parametr udává velikost obrazce a může nabývat hodnot mezi 1-255.

CSET n

Cíl: výběr režimu znaky/grafika nebo navrácení poslední grafiky.

Příkaz CSET umožňuje v programu volit různé soubory znaků.

CSET 0 - velká písmena/grafika

CSET 1 - velká/malá písmena
Dále může být způsobem obnovena naposled použitá grafika.
CSET 2 - HIRES
CSET 2:MULTI c1, c2, c3 - MULTI - parmetry c1-c3 určují nové barvy

Příklad:
```
10 REM CSET 0/1
20 PRINT "    "
30 PRINT A (12,14)"SIMONS BASIC"
40 CSET 1:PAUSE 1:CSET 0:PAUSE 1
50 GOTO 20
```

COLOUR br, hg

Cíl: určení barev pozadí a rámu obrazovky.

Příkaz COLOUR stanoví barvu pozadí a rámu, a to jak v normálním režimu, tak v režimu vysoce rozlišitelné grafiky. Barvy jsou dány odpovídajícími ciframi. Barva rámu může být změněna dalším příkazem COLOUR. Barva pozadí při vysoce rozlišitelné grafice nebo v MULTI grafice příkazem HIRES.

NRM

Cíl: vrací obrazovku z grafiky do normalního rozlišení

Příkaz NRM vymaže vysoce rozlišitelnou grafiku nebo MULTI grafiku z obrazovky a vrací obrazovku do normálního stavu.

Příklad: po 5 s se program vrátí k normálnímu rozlišení
```
10 HIRES 0,1:MULTI 0,2,6
20 REC 0,0,40,20,1
30 REC 20,20,40,20,2
40 REC 40,40,40,20,3
50 PAUSE 5
60 NRM
```

Text v grafice

CHAR x, y, kód znaku, velikost

Cíl: zobrazit jednotlivé znaky na obrazovce, která je v grafickém režimu.

Příkazem CHAR se dají vložit do grafiky znaky. Parametry mají následující význam:
x, y - souřadnice levého horního rohu matice znaku
kód znaku - POKE kód
velikost - faktor, kterým se znaky ve vertikálním směru roztáhnou

TEXT x, y, "(CTRL a) řetězec, bod, velikost, odstup

Cíl: převest na obrazovku v grafickém režimu řetězec znaků.
Příkaz TEXT umožňuje uvést na obrazovku v grafickém režimu řetězec znaků. Parametry mají následující funkci:
x, y - souřadnice levého horního rohu řetězce
Dalším parametrem je sám řetězec nebo proměnná. První znak určuje, zda bude řetězec zobrazen v režimu malá či velká písmena.
(CTRL a) - velká písmena invers a
(CTRL b) - malá písmena invers b
velikost - faktor, jímž se ve vertikálním směru řetězec roztáhne
odstup - určuje odstup mezi jednotlivými znaky

Řízení obrazovky

Tento odstavec pojednává o řízení obrazovky. SIMONS BASIC umožňuje programově řídit barvu obrazovky nebo její části, rolování obrazovky, naplnění určitými znaky, uložení obsahu do paměti ev. vytištění na tiskárně.

FLASH f1, rychlost

Cíl: blikání určité barvy.
Příkaz FLASH způsobí dlouhodobé střídání barev mezi normálním a inverzním módem. Parametr f1 určuje barvu. Rychlost může nabývat od 0 do 255. Jednotka je 1/16s. V režimech HIRES nebo MULTI je příkaz FLASH bez vlivu.

OFF

Cíl: ukončuje příkaz FLASH.

BFLASH

Cíl: rychlé střídání barev rámu obrazovky.

Příkaz BFLASH způsobí dlouhodobě střídání barev rámu obrazovky. Parametry f1 a f2 určují barvu. Rychlost udává rychlost změn (1-255).

BFLASH 0

Příkaz BFLASH 0 zruší příkaz BFLASH.

FCHR r, c, w, d, kód

Cíl: vyplnit určitou část obrazovky určitými znaky.

Příkaz FCHR umožňuje vyplnit určitou část obrazovky určitými znaky. Parametry určují:

r (0:24) c (0:39) levý horní roh pole, který se má vyplnit
w - počet sloupců
d - počet řádků
kód - POKE kód znaku, kterým se má pole vyplnit

FCOL r, c, w, d, barva

cíl: určení barvy znaku v určitém poli.

Příkaz FCOL přiřazuje znakům ve vybraném poli obrazovky určitou barvu. Parametry r, c, w, d odpovídají výše uvedeným. Poslední parametr je vyhrazen barvě.

Příklad:

```
10 REM FCHR-FCOL
20 POKE 53280,8:POKE 53281,8
30 PRINT"  "
40 FCHR 10,10,10,10,10
50 FOR x=10 TO 15 STEP5
60 FOR y=10 TO 15 STEP5
70 FCOL x,y,5,5,f
80 f=f+1
90 NEXT y
100 NEXT x
```

FILL r, c, w, d, kód, barva

Cíl: vyplnit určitá pole obrazovky znaky známé barvy.

Příkaz FILL naplní část obrazovky znaky předem zvolené barvy. Parametry r, c, w, d jsou shodné s parametry udanými u předchozích příkazů.

Příklad:
```
10 REM FILL
20 POKE 53280,1:POKE 53281,1
30 PRINT"     "
40 FOR x=0 TO 15
50 FILL x,2xx,5,5,x,x
60 NEXT x
70 GOTO 70
```

MOVE r, c, w, d, dr, dc

Cíl: dublování části obrazovky.

Příkaz MOVE dubluje určitou část obrazovky. Parametry r, c, w, d udávají rozsah, který má být opakován. Parametry dr, dc udávají absolutní polohu místa, od kterého začíná kopírovaná část.

Příklad:
```
10 REM MOVE
20 POKE 53280,1:POKE 53281,1
30 PRINT"  "
40 FILL 0,0,5,5,1,10
50 MOVE 0,0,5,5,0,35
60 MOVE 0,0,5,5,19,0
70 MOVE 0,0,5,5,19,35
80 GOTO 80
```

INV r, c, w, d

Cíl: invertovat část obrazovky.

Příkaz INV invertuje všechny znaky v určité obrazové části obrazovky. Parametry r, c, w, d odpovídají předchozím parametrům.

Rolování obrazovky

SIMONS BASIC umožňuje rolovat určitou část obrazovky na čtyři různé směry. První parametr určuje směr rolování - LEFT, RIGHT, DOWN 0. Druhý parametr může být buď w nebo b.

W - cyklické rolování, tj. bez ztráty řádků.
B - rolování bez oběhu obrazu, tj. nebudou nasunuty prázdné řádky.

Nahrávání a uchovávání dat z obrazovky.:

SCRSV 2, 8, 2 "nazev, s, w"

Cíl: přenesení dat obrazovky v režimu LOW RESOLUTION na disk nebo kazetu.

Příkaz SCRSV umožňuje uložit data z obrazovky přímo na disketu nebo na kazetu. Druhá cifra za příkazem udává číslo přístroje. Názvem rozumíme file, pod příslušným názvem je obsah obrazovky uložen a může být znovu pomocí příkazu SCRSV vyvolán. Tímto příkazem nemůže být ukládána grafika v režimu HIRES nebo MULTI.

SURLD 2, 8, 2 "název"

Cíl: natažení uložených dat z disku nebo kazety.

Tímto příkazem se mohou z externí paměti natáhnout do počítače data, která byla předtím pomocí příkazu SCRSV uložena. Parametry odpovídají příkazu SCRSV.

Vytištění dat

COPY

Příkaz COPY vytiskne HIRES nebo MULTI grafiku. Aby tiskárna vytiskla přesný kruh, musí být v případě CIRCLE zadané parametry x a y shodné.

HRDCOPY

Tento příkaz vytiskne data z obrazovky LOWRES.

BCKGNDS bf, h1, h2, h3

Cíl: určení barvy, pozadí, znaku.

Každý znak na obrazovce je v matici 8x8 bodů. Základní barva tohoto čtverečku odpovídá barvě obrazovky (kromě inverzního zobrazení). Příkaz BCKGNDS určuje barvu pozadí znaku v normálním a inverzním písmu. Nezapomeňte, že příkaz

platí jen pro ty znaky, které se nacházejí nahoře na klávesách. Nikoli pro grafické znaky. Parametr bf příkazu určuje barvu obrazovky. Další 3 parametry udávají barvu pozadí znaku.

Sprite a grafika

SIMONS BASIC má dvě grafické zvláštnosti. Jednou z nich je možnost vytvořit si svůj vlastní soubor znaků a jimi obsadit klávesy Commodore C64. Každý znak je přitom osazen opět do matice 8x8 dodů. Druhou zvláštností jsou sprite a jejch uvedení do pohybu. Ve standardní verzi CBM BASIC je pro vygenerování SPRITE zapotřebí mnoha příkazů POKE. SIMONS BASIC nahrazuje zmíněné POKE jednoduchými, pro zacházení snadnými příkazy.

V dalším textu budou SPRITE nazývány MOB (moveable object block). Současně je možno zobrazit až 8 nezávislých MOB. MOB je možno vytvořit v zásadě dvěma způsoby:

HIRES - 24 * 21 bodu, jednobarevný
MULTICOLOUR - 12 * 12 bodu, tříbarevný

MOB mohou být zobrazeny buď na normální obrazovce nebo v grafickém módu.

Tvorba MOB

DESIGN c, sa+gc

Cíl: vyčlenění místa v paměti pro MOB.

Příkaz DESIGN rezervuje potřebné místo v paměti C64 pro MOB, který má být vykonstruován. První parametr c udává, zda má být MOB vyobrazen v režimu HIRES (0) nebo MULTI (1), druhý parametr definuje startovací adresu v paměti počítače. Adresa se vypočítává z čísla bloků vynásobením 64. Každý MOB spotřebuje z paměti 64 Byte.

číslo bloku	paměťové místo
13 - 15	832 - 1023
32 - 63	2048 - 4095
128 - 255	8192 - 16383

Má-li být MOB zobrazen na obrazovku v režimu HIRES, musí být k počáteční adrese přičtena hodnota 49152 ($C000). Je-li v programu použit příkaz MEM, potom je pro MOB možné

využít jen 192 - 255.
Pozor! Můžete sestrojit jen tolik MOB, na kolik je místo v paměti počítače. Na obrazovce se jich může nacházet toliko 8. V programu je možno 1 MOB nahradit jiným, přičemž pod stejnou startovací adresou pracuje nový.
Příklad: připravit místo v paměti pro HIRES MOB na normální obrazovce.
100 DESIGN 0,832

Výsledek: MOB bude uložen v bloku 1313*64=832.

CMOB c1, c2

Cíl: přiřazení barvy pro MULTICOLOUR MOB.
Parametry mohou nabývat hodnot mezi 0 a 15 a odpovídají barvám C64. Všechny body zadané b mají barvu c1 a body zadané D mají barvu c2.

MOB SET mb, blk, col, pr, res

Příkaz pevně stanovuje určité vlastnosti MOB.

mb - číslo MOB
blk - stanovuje, ze kterého bloku jsou brána data pro MOB
col - stanovuje barvu příslušného MOB

MMOB SET mn, x1, y1, x2, y2, exp, rychlost

Příkaz MMOB způsobí, že se MOB zobrazí a umožní jeho pohyb.
mn - číslo MOB
x1, y1 - počáteční adresa MOB
x2, y2 - cílová souřadnice MOB
exp - velikost MOB
rychlost - rychlost pohybu (1-255) 1 = nejvyšší

RLOCMOB mn, x, y, exp, rychlost

RLOCMOC pohybuje objektem zobrazeným v určitém místě obrazovky do jiné pozice obrazovky. Souřadnice x a y patří cíli pohybu. Ostatní parametry odpovídají předchozím. Vymazání MOB z obrazovky provedeme příkazem MOB OFF.

DETECT n

Příprava dotazu na kolizi MOB. Příkaz DETECK se musí zadat před příkazem CHECK. Je-li parametr n=0, je připravena kontrola srážky dvou MOB. Je-li n=1; je počítač připraven kontrolovat kolizi mezi znakem na obrazovce a MOB. Příkaz DETECK musí být zadán 2*. Při prvním provedení příkazu je příslušný registr vymazán a podruhé je aktivován pro kontrolu kolize.

CHECK

Příkazem CHECK se testuje, zda se uskutečnila srážka dvou MOB. Parametry m1 a m2 udávají parametry MOB, mezi nimiž je srážka očekávána. V kladném případě (uskutečnila-li se srážka) je provedeno návěstí za THEN.

Generování nové sady znaků

MEM

Aby se mohla sada nadefinovat nebo změnit, musí se přeložit z ROM do RAM. K tomu nám slouží příkaz MEM. Příkaz provádí funkce:
- posunutí obsahu paměti znaků (ROM) do rozsahu RAM za Kernal
- posunutí obrazovky RAM do $CC00
- ohraničení rozsahu MOB na bloky 192+255

Strukturované programování

Velkým problémem při programování ve standardním BASICU je nemožnost strukturovaného programování. Příkazy GOTO a GOSUB způsobují nepřehlednost a nesrozumitelnost kontroly vytvořeného programu. Pomoc přinášejí pouze řádky s komentáři REM, které jednotlivé rutiny vysvětlují. SIMONS BASIC je oproti tomu vybaven příkazy strukturovaného programování.

Programové smyčky, podmínky

IF...THEN...ELSE

Kontrola podmínky a podle výsledku provedení návěští. Rozdíl proti standardnímu příkazu je v tom, že zde lze zadat návěstí, které je vykonáno v případě nesplnění podmínky.

Příklad:
```
10 PRINT"KONEC (a/n)"
20 FETSCH"an",1,antw$
30 IF ANT w$="a" THEN END: ELZE: GOTO 10
40 END
```

REPEAT...UNTIL

Opakování návěstí tak dlouho, dokud není splněna podmínka. Příkaz má podobnou funkci jako FOR...TO...NEXT ve standardním BASICU. REPEAT startuje funkci a UNTIL kontroluje podmínky. Je-li podmínka splněna, program končí.

RCOMP

Umožňuje přezkoušet podmínky naposled použitého příkazu IF..THEN..ELSE. Příkaz přináší ulehčení tehdy, když je svázáno více podmínek dohromady.

LOOP...EXITIF...EDN LOOP

Oběh po smyčce po dobu umístění podmínek. Tento příkaz umožňuje sestavit takovou programovou smyčku, v níž je konec pevně stanoven. Smyčka může obsahovat více podmínek. Je-li splněna příslušná podmínka, oběh po smyčce končí a program pokračuje dále za smyčkou. Není-li splněna zadaná podmínka, probíha program po smycce znova.

Technika programování

Čtyřmi příkazy umozňuje SIMONS BASIC psaní strukturovaného programu. Zmíněné příkazy dovolují opatřit programové rutiny názvy (LABEL) a pod stejnými názvy jejich opětovná vyvolání). Potřeba GOTO a GOSUB se tím nadále sníží.

PROC

PROC umožňuje přiřadit programové rutině symbolickou adresu. Všechny znaky za příkazem PROC tvoří Label, pod nímž může být potom rutina vyvolána. Ve stejném programovém řádku nesmí potom žádná další návěstí vystupovat.

END PROC

END PROC určuje konec uzavření rutiny. Příkaz je ekvivalentní s RETURN ve standardním BASICU. Způsobí zpětný skok na příkaz bezprostředně následující po EXEC.

CALL

Skok na symbolickou adresu (Label). Příkaz CALL odpovídá příkazu GOTO BASIC standard. Zde ovšem neskáče na první číslo řádku, ale na symbolickou adresu.

EXEC

Příkaz EXEC odpovídá příkazu GOSUB. Program je zde volán pod symbolickým názvem (Label). Po návratu z podprogramu je prováděn příkaz bezprostředně následující po EXEC.

Proměnné

V BASIC musí mít všechny proměnné určitou hodnotu. V SIMONS BASIC existuje možnost přiřadit proměnné globální hodnotu pro celý program a na určitých místech další hodnotu. Globální hodnota se přitom neztrácí.

LOCAL

Stanovení proměnných v určitém místě programu. Příkaz LOCAL přiřazuje proměnné nové hodnoty. Původní hodnota této proměnné zůstane zachována.

GLOBAL

Obnovení původních hodnotových přiřazení v programu.

Pojednání o chybách

SIMONS BASIC umožňuje zachytit chyby v programu. Vyskytne-li se chyba, potom dojde v určitém místě programu k jeho rozvětvení. S proměnnými ERRN a ERRLN může být přidavně určeno, o jakou chybu se jedná a ve kterém řádku se nachází. S těmito informacemi může být potom chybové hlášení opatřeno textem.

ON ERROR

Vyskytne-li se při zpracování programu chyba, odskočí program na uvedený řádek. V proměnných ERRN a ERRLN jsou uložena čísla chyb a příslušného řádku.

číslo chyby	chyba
1	TOO MANNY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT PRESENT
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEFINED STATEMENT
18	BAD SUBSCRIPT
19	RE-DIMENSIONED ARRAY
20	DIVIZION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG

NO ERROR

Příkazem NO ERROR se potlačí hlášení chyby. Program běží dál. Při výskytu další chyby se objeví hlášení známé ze standard BASICU.

OUT

Zrušení kontroly chyb SIMONS BASIC. Po zadání OUT jsou chyby kontrolovány obvyklým způsobem.

Hudba se SIMONS BASICEM

Jednou ze zajímavých zvláštností SIMONS BASICU je možnost programovat hudbu. Cvičením a zkušeností se dá napodobit zvuk mnoha různých hudebních nástrojů. Ve standardním BASICU je nutno používat mnoho příkazů POKE. SIMONS BASIC má příkazy, které tuto práci podstatně usnadní. Aby se z C64 vytvořil hudební nástroj, je nutno zadat následující parametry:

- síla zvuku
- tvar vlny
- obalová křivka
- hlas - oktáva
- nota

Hudební příkazy

VOL n

Příkaz VOL n nastavuje sílu zvuku tří tónových generátorů. Parametr n může nabývat hodnot od 0 do 15.

n=0 generátor vypnut
n=15 max. síla zvuku

WAVE st, bin. číslo

Nastavení tvaru průběhu. Tvar průběhu určuje charakter zvuku tří tónových generátorů C64.

st = 1 hlas 1
st = 2 hlas 2
st = 3 hlas 3

Druhým parametrem WAVE je binární číslo. Nastavením jednotlivých bitů je nastaven. Přesné vysvětlení jednotlivých bitu:

Bit 0 - gate

Tento bit trigruje generátor obalové křivky. Je-li tento nastaven, jsou inicializovány cykly **ATTACK** DECCAY-SUSTAIN. Je-li tento bit 0, začíná cyklus RELEASE. Při provádění příkazu PLAY je tento bit řízen automaticky.

Bit 1 - synchronizace

Pomocí tohoto bitu je možné synchronizovat funkci určitého hlasu s frekvencí jiného hlasu. Synchronizuje-li se hlas s proměnnou frekvencí a další, který má konstantní frekvenci, může se vyrobit velmi komplexní zvuk. Nejlepších výsledků se docílí, když se první frekvence zvolí níže než proměnná.

Bit 2 - kruhová modulace

Při kruhové modulaci je trojúhelník nahrazen průběhem, který vznikne kombinací zvoleného hlasu s jiným hlasem. Stejně zde se doporučuje frekvenci jednoho hlasu nechat konstantní a druhou měnit.

Bit 3 - testovací bit

Tento bit není používán. Nemusí být nastaven.

Bit 4 - trojúhelník

Zapíná oscilátor generující trojúhelník. Trojúhelníkové kmity obsahují málo vyšších harmonických.

Bit 5 - pila

Zapíná oscilátor generující pilové kmity. Pilové kmity obsahují mnoho sudých a lichých harmonických.

Bit 6 - obdélník

Zapíná generátor obdélníku, podíl vyšších harmonických kmitů je určen šířkou pulsu.

Bit 7 - šum

Zapíná generátor šumu. Sum závisí na frekvenci oscilátoru

ENVELOPE hlas, Attack, Decay, Sustain, Releace

Nastavení obalové křivky. Obalová křivka každého hlasu může být nastavena odděleně. Nastavení obsahuje čtyři parametry: Attack, Decay, Sustain, Release.

Příklad: obalová křivka pro hlas 1

```
30 ENVELORE 1,8,8,8,0
```

MUSIC n, "tón"

Určení tónu, který má počítač zahrát. Příkazem MUSIC jsou

určeny další parametry, které mají vliv na vznikající tón. Parametr n určuje výšku tónu a musí ležet mezi 1 a 255.

Další parametr určuje výšku tónu a délku jednoho nebo více tónů. Nejdříve je v řetězci určen hlas (SHIFT HOME (1+3)). Potom následuje řídící tlačítko F1-F8 pro takt a notu (C0-A7)

Funkce řídicích tlačítek

tlačítko	funkce
SHIFT CLR/HOME + cifry určují	hlas
F 1 následující nota je jako	1/16
F 2	1/8
F 3	1/4
F 4	1/2
F 5	1/1
F 6	2/1
F 7	4/1
F 8	8/1

Každá nota je sestavena z písmene a oktávy. Dostane se jako 1/2 tónu zvýšená nota po současném stisku názvu noty a HOME a G. Tím se startuje RELEASE poslední noty.

OXFORD PASCAL

Pascal je vykonný programovací jazyk vyšší úrovně, jehož autorem je Niklaus Wirth z Curychu ze Svicarska. Může být efektivně implementován na malých počítačích stejně jako na velkých. Poskytuje četné výhody proti jiným mikropočítačovým jazykům jako je BASIC. Některé z těchto výhod jsou:

- bloková struktura jako je třeba v ALGOLU
- jména proměnných mohou odrážet význam proměnných
- výkonné technické prostředky pro strukturalizaci dat
- uivatelem definované soubory a konstanty
- nadné spojování funkcí a podprogramů
- rekuzivní volání
- průběžné samočištění
- kontrolu chyb během provádění programu
- dynamické přidělování proměnných
- vyšší standardizace
- vysoká prováděcí rychlost
- větší programová logibylita (možnost rozšiřování)

Oxford pascal je implementací standardního PASCALU navržený pro C64. Poskytuje všechny rysy tohoto mocného jazyka společně s doplňky pro majitele domácích počítačů.

Oxford Pascal umožňuje dva režimy práce. V jednodušším koreziduje překladač Pascalu v RAM společně s uživatelským programem. To je ideální při učení se jazyku nebo při vytváření menších programů, které nepotřebují disk. Většina příkazů je v tomto režimu použitelná kromě těch, které se týkají disketových souborů. Pro složitější programy může být použit disketový překladač, který vám poskytne výhody tohoto jazyka v plné míře.

Úvod do Pascalu pro začátečníky

Každý počítač pracuje v jazyce čísel nazvaném strojový kód, ale všeobecně je pro lidi těžké programovat ve strojovém kódu a navíc jeho verze jsou rozdílné. Je mnohem jednodušší programovat, komunikovat s počítačem ve vyšším jazyce, jako je třeba Pascal nebo jazyk C. Tyto jazyky jsou vytvořené na základě jakési angličtiny, která má však svá velice striktní pravidla gramatiky, aby se vyloučila možnost nedorozumění.

Pascal byl vyvinut v roce 1968. Je to ideální jazyk pro výuku programování. Program je z pascalu automaticky překládán počítačem do strojového kódu, ve kterém je pak přímo prováděn. Začneme hned s jednoduchým programem.

Příklad 1

Především musíte zadat váš program do paměti počítače, a to se děje prostřednictvím editoru. Zadejte první řádek programu, který je uveden níže, a odešlete klávesou /RETURN/.

```
10 BEGIN
20 WRITE ("AHOJ")
30 END.
```

Uvozovky mají být jednoduché ale wizavrite, na kterém je manuál psán, je nemá.

Jakmile zadáte první řádek, editor by měl automaticky vypsat číslo následujícího řádku, které pak už nemusíte zadávat. Tato čísla nemají v jazyce žádný význam, jsou zde čistě z důvodu snadného editování, čož si ukážeme nyní. Zapamatujte si, že pokud uděláte při zadávání programu

chybu, můžete program opakovat pomocí editačních kláves INST DEL, CRSR nahoru, dolů, doleva, doprava zrovna tak, jak jste zvyklí z BASICU. Až skončíte, zadejte prázdny řádek a tím odstavíte automatické řádkování.
Nyní zadejte r a "return".

Pokud vše proběhne hladce, mělo by se vypsat:

```
Compiling
Program 0 0909
0 error (s)
Compilation complete
```

Nedělejte si nyní starosti s detaily, ale vše, co se nyní odehrává je, že počítač prochází váš program a překládá ho do strojového kódu. Pokud neobdržíte hlašení 0 error(s), pak jste se pravděpodobně dopustili chyby při zadávání. Při bezchybné kompilaci by se měl program automaticky spustit a vypsat hlášení:

```
AHOJ
```

Jednou zkompilovaný program může být spuštěn kolikrát chcete opětovným zadáním:

```
r "return"
```

Potom se překlad již neprovádí a počítač jen vypíše:

```
AHOJ
```

Nyní se podívejme na program detailně. Hlavní tělo Pascalového programu je vždy uzavřeno mezi slovy "begin" a "end". Poslední END musí být vždy zakončeno tečkou. Pascalové programy se skládají z posloupností, které se vykonávají vždy v tom pořadí, v jakém jsou zapsány. Náš program 1 má příklad WRITE, který říká počítači, aby něco vypsal na obrazovku, v našem případě text AHOJ.

Příklad 2

Kromě řetězců mohou být na obrazovku vypisovány i jiné věci. Vyzkoušejte si program. Budeme používat téhož postupu

jako v případě 1, ale nesmíme zapomenou nejprve vymazat program z paměti. Proto zadejte:
NEW "return"

```
begin
write(3+4):
WRITE(6-2-1)
end.
```

Písmena malé a velké abecedy jsou ekvivalentní.

Zadání ukončete příkazem pro kompilaci a spuštění. Počítač by měl vypsat:

```
7   3
```

Příklad dvě obsahuje dva příkazy, které musí být odděleny středníkem. Je to příklad na celočíselnou aritmetiku (INTEGER).

Příklad 3

```
begin
write(6*7,18div4,18mod4,-(4+2)*3)
end.
```

Počítač by měl vypsat:
42 4 2 18

V Pascalu "*" znamená násobení
"div" celočíselné dělení
"mod" zbytek po celočíselném dělení

Všimněte si, jak se používá závorek ke změně pořadí provádění operací. To proto, že počítač provádí násobení a dělení předtím, než provede sčítání a odčítání. Jediným příkazem WRITE může být vypisováno jakékoli množství položek, které musí být odděleny čárkami.

Příklad 4 - funkce předefinovaná v Pascalu

```
begin
write(sqr(4+5),abs(-44),abs(44)odd(3))
end.
```

Počítač by měl vypsat:
81 44 44 TRUE

SQR ABS a ODD se nazývají funkce. V Pascalu existuje značné množství nadefinovaných funkcí.

SQR - následovné číslo dává druhou mocninu tohoto čísla
ABS - dává absolutní hodnotu čísla
ODD - (3) je TRUE (pravdivý výrok), neboť 3 je číslo liché. Tato dává logický nebo říkáme Booleovský výsledek. Může nabývat dvou hodnot TRUE a FALSE. Booleovské hodnoty se v Pascalu užívají ve velké míře, a tak se s nimi seznamte blíže.

Příklad 5 - Booleovské výrazy

```
begin
writeln(true,false,3=3,3=4);
write( 3větší menší4,5menší6,9větší=10);
end.
```

Počítač by měl vypsat:
TRUE FALSE TRUE FALSE
TRUE TRUE FALSE

protože 3 se rovná sama sobě
3 se nerovná 4
atd
WRITERN má týž význam jako WRITE, ale generuje zároveň přechod na nový řádek po vypsání všech položek v závorkách.

Příklad 6 - Booleovské výrazy

Mohou se vám zdát zpočátku komplikované, ale počítač je vyhodnocuje na základě striktní logiky.

```
begin
write((3=3)and(3menší5),(3=4)or(3větší11))
write(not true,not false,not(1=2))
end.
```

Měli bychom obdržet výsledek:

TRUE FALSE FALSE TRUE TRUE

protože (3=3) a (3menší5)jsou pravdivé a (1=2) není pravda, takže not(1=2) je pravda.
x AND y je pravda, když současně y a x jsou pravdivé.
x or y je pravda, když x nebo y nebo oba jsou pravdivé.
not je pravda, když x není pravda a naopak.

Příkazy PASCALU

Nejprve několik slov o symbolech. Jsou to základní kameny pascalovských programů. Rozeznáváme tři druhy:
1. Pascalová klíčová slova. Například BEGIN END. Jsou v Pascalu rezervována stále a nemohou být měněna. Kompletní seznam naleznete dále.
2. Specifické symboly jako . : = menší větší
3. Identifikátory, což jsou jména, která volí uživatel. Mohou to být sekvence písmen nebo číslic, ale musí vždy začínat písmenem.
Upozornění: Identifikátory jsou chápány Pascalem jako různé, liší-li se v prvních 8 znacích, jsou pro Pascal týmiž identifikátory. Velká písmena jsou odpovídající písmenům malé abecedy a tak PI a pi jsou synonyma.

Některé standardní identifikátory jako např. WRITE a WRITELN jsou předeklarovány ve všech verzích PASCALU. Mohou být uživatelem nadefinovány, avšak musí se lišit od všech Pascalových klíčových slov.
Důležité! Pascalové symboly mohou obsahovat mezery "Henry then 8th". Obzvláště si všimněte, že ": =" nemůže být použito jako ":=". Jinak mohou být tabulátory a přechod na nový řádek kdekoliv. V Pascalovém programu jsou ignorovány.

Příklad 7 - proměnné a přiřazení

```
var x,y : integer:
begin
x:=3:y:=27:
writeln(x,y):
x:=4
y:=x+2:
write(x,y,x+y)
end.
```

```
Mělo by se vypsat:
3  27
4   6   10
```

Deklarace var předchází begin a informuje kompilátor, že identifikátory x, y jsou proměnné, které mají uchovat celočíselnou proměnnou. Jak již vyplývá z názvu, proměnná může měnit svoji hodnotu v důsledku vykonávání programu. Na řádku 3 je proměnné x přiřazena hodnota 3 a proměnné y hodnota 27. Později je x přiřazena 24 a y hodnota x+2. Proměnné též mohou být deklarovány jako booleovské nebo jakéhokoli typu.

Příklad 8 - opakování pomocí smyček FOR

```
var integer:
begin
writeln("směr nahoru"):
for i:=1 to 5 do writeln(i):
writeln("směr dolů")
for i:=5 downto -1 do writeln(i):

Mělo by se vypsat:
směr nahoru
  1
  2
  3
  4
  5
směr dolů
  5
  4
  3
  2
  1
  0
 -1
```

Příkaz následující po for..do je opakován tak dlouho, dokud i nenabude všech požadovaných hodnot.

Příklad 9 - příkaz if

```
var i:integer
begin
 for i:=1 to 11
  begin
   writen (i):
    if odd(i) then writeln ("je liche")
   else writeln("je sudé"):
  end:
 end.
```

Výsledek by měl být:

```
1 je liché
2 je sudé
3 je liché
4 je sudé
5 je liché
6 je sudé
7 je liché
8 je sudé
9 je liché
10 je sudé
11 je liché
```

IF příkaz umožňuje počítači výběr, který ze dvou příkazů má hodnotě booleovského výrazu. Část else podmínkového příkazu je nepovinná, ale co je důležité, že před else nesmí být nikdy středník.

Příklad 10 - nalezení aritmetického průměru

tento příklad seznamuje s realizací vstupu dat z klávesnice a s obecnějším druhem smyček.

```
ver:=0:count:=0
begin
  total:=0,count:=0
  write("zadejte libovolná čísla"):
    read(x):
    total:=total+x:
    if xje větší 0 then count:=count +1:
  until x=0:
 writeln("aritmetický průměr je ",total/cont):
end.
```

Po spuštění by nás měl počítač vyzvat k zadání čísel z klávesnice. Sérii ukončíte zadáním čísla 0.

Příklad 11 - příkaz CASE

Tento vzorový příklad uvádí mnohem rafinovanější způsob výběru mezi několika rozdílnými příkazy.

```
var verse,i:integer
begin
  for verse:=1 to 4 do:
   begin
    writeln:
    for:=verse downto 0 do
     case i of
      3:writeln("tri muzi")
      2:writeln("dva muzi")
      1:writeln("jeden muz")
      0:writeln("a jeho pes")
    end:
  end:
end.
```

Program by měl vypisovat:

```
jeden muž
a jeho pes

dva muži
jeden muž
a jeho pes

tři muži
dva muži
jeden muž
a jeho pes
```

Chybové hlášení bylo způsobeno tím, že v poslední verzi nenabylo i hodnoty ani jednoho návěstí z příkazu "case", pro hodnotu i=4 nemůže příkaz CASE pracovat. Návěstí v příkazu CASE mohou být též skládána.

Opravování chyb

Opravování chyb můžete provádět pomocí editoru, aniž byste museli přepisovat celý program. Abyste opravili výše uvedený program, zadejte nejprve:
LIST - program bude vypsán na obrazovce

```
10 var x:boolean:
20     x:integer
30 begin
40     read(x)
50     write(x)
60 end
```

K vymazání druhého řádku stačí zadat 20 následované RETURN. Opětný výpis by měl vypadat:

```
10 var x:boolean
20 begin
30 read(x):
40 write(x):
50 end
```

Rádek 10 je však stále ještě špatný. Chceme číst a vypisovat číslo Integer a ne boolean. Proto zadejte:

```
Change /bolean /integer/
```

Příkaz LIST by měl nyní vypsat správnou verzi programu.

```
10 var x:integer
30 begin
40 read (x):
50 write (x):
60 end.
```

Program nedělá mnoho, pouze přečte číslo z klávesnice a vypíše je na obrazovce. Ucelenější přehled o tom, jak editor pracuje, najdete v souhrnu editovacích příkazů.

Příkaz WHILE

Kromě smyček repeat a for existuje v Pascalu ještě další druh. Příkaz while pracuje podobně jako repeat až na to, že test konce smyčky je proveden již na začátku, takže smyčka nemusí být vůbec provedena. Zrovna tak jako u smyčky for může být opakován pouze jediný příkaz nebo sekvence příkazů uzavřená begin...end.

Např.

```
 i:=1:
while i je menší = do
   begin
   writeln (i):
   i:=I+1:
end.
```

má stejný význam jako:

```
fori:=1 to 5 do writeln(i):
```

Více o datových typech v PASCALU

Příklad 12 - čísla v plovoucí čárce

```
begin
  writeln(3.3,33.0,330.0,0.33):
  writeln(-3.3E3,3.3E-1,4.5+2.1)
end.
```

Počítač by měl vypsat:

```
 3.30000E+00     3.30000E+01     3.30000E+02     3.30000E-01
-3.30000E        3.30000E-01     6.60000E+00
```

Přítomnost buď desetinné tečky nebo exponentu (e) v čísle říká Pascalu, aby s ním zacházel jako s číslem v plovoucí čárce nebo číslem real. Čísla v plovoucí čárce mají v Pascalu přesnost 9-ti číslic a jejich velikost může být v rozmezí 1E-38 až 1E+38. V porovnání s čísly integer byste neměli očekávat, že Pascalová reálná aritmetika bude tak přesná. To znamená, že např. 4,0 může být ve skutečnosti vypsáno jako 3.9999999. Také se nemůžete spolehnout při testování reálných čísel na rovnost.

Příklad 13 - aritmetika v plovoucí čárce

```
var x,y:real
begin
  x:=9.1:
```

```
  y:=8,7:
  writeln(x+y:7:2,x-y:7:2,x/y:7:2,x*y:7:2):
  writeln(sqr(x):7:2,sqrt(x):7:2,abs(x):7:2):
  writeln(trunc(x),trunc(y),round(x),round(y))
end.
```

Mělo by se vypsat:

```
17,80     0.40     79.17     1.05
82.81     3.02      9.10
 9        8         9         9
```

Již dříve jsme se setkali s +,-,*,/. Užívají se hlavně pro sčítání, odčítání, násobení a dělení čísel REAL.

SQRT, SIN, ARTAN, LN, EXP, ROUND, TRUNC

SQR znamená x umocnit na druhou
SQRT znamená druhou mocninu z x
ABS dává absolutní hodnotu z x
TRUNC dává celočíselnou část x
ROUND zaokrouhlí x k nejbližšímu celému číslu
SIN dává sinus x (x je v radiánech)
COS dává cosinus x -//-
ARTAN dává úhel v radiánech, jehož tangens je x
LN dává přirozený logaritmus
EXP dává číslo E povýšené na x-tou

1 radian = 57.29578 stupňů
e = 2.718281

Příklad 14 - výstup s formátováním konstant

```
Program vlny:
const f1=0.5:f2=0.05:amplituda=19:
var x1,x2,y-real:
begin
  x1:=0:
  x2:=0:
    x1:=x1+f1:
    x2:=x2+f2:
```

```
    y:=sin(x1)*sin(x2)*amplituda:
    writeln("X":round(y)+amplituda)
  until false
end.
```

Program by měl zobrazovat amplitudu modulované sinusové vlny. Protože smyčka repeat..until..false v příkladu 14 se bude provádět téměř donekonečna, jediným způsobem, jak ji zastavit, je klávesa STOP.
Počítač by měl vypsat:

```
BREAK AT LINE *****
```

kde ***** je řádek, který byl, když se zmáčkla klávesa STOP. Hlavička programu je u PASCALU nepovinná a v tomto případě slouží pouze k pojmenování programu: vlny. Pro počítač nemá význam a je pouze pomůckou pro dokumentaci.

Jakýkoli text uzavřený mezi parametry symbolu (* text *) je Kompilátorem také ignorován. Tato možnost může být využita pro zápis komentářů, které čtenáři pomohou k porozumění programu. Konstanty uvedené klíčovým slovem const jsou hodnoty, které nemohou uvnitř programu svoji hodnotu měnit.

Deklarace konstant jsou užitečné, chceme-li dát jméno určitým konstantám, a ty pak umožňují jednodušší změnu v programu. Zkuste pomocí editoru změnit frekvence f1, f2 a amplitudu v příkladě 14 a pozorujte, jak se vlna mění.

Znaky

Všimněte si, jak program specifikuje šířku výpisového pole, aby sdělil počítači, kolik znaků přiřadit x, když se vypisuje pokud je zadáno příliš mnoho znaků, je provedeno doplnění potřebným počtem mezer zleva. Pokud není zadán patřičný počet znaků, potřebný pro výpis řetězce, je řetězec zkrácen zprava.
Např.

```
write("ahoj":2)
```

by vypsal: ah

Numerické hodnoty jsou vypsány celé, dokonce i když je specifikováno příliš málo znaků.

Příklad 15 - grafika C64

```
const psloup=40
var rad:=1,i:integer:
begin
  page:
  for rad:=1 to 24 do
  if odd(rad) and add(i) or
  not odd(rad) and not odd(i) then write (chr(177))
  else write(chr(178))
end.
```

Tento program by měl vytvořit na obrazovce pravidelný vzor. Znaky jsou v Pascalu chápány jako řetezce délky 1.

Patří k datovému typu CHAR, CHAR, který má v Oxford Pascalu 256 hodnot, odpovídající souboru ASCII, rozšířenému na C64. Funkce ord(z) dává znak odpovídající ASCII celočíselnému kódu x(0-255).
Protože deklarce array of array se používá velice často, má pro ni Pascal zkratku:

```
šachovnice! 3,5 ! a podobně se to odrazí i v deklaraci:

var šachovnice: array! 1..8,1..8! of figura
```

Definice vašich vlastních datových typů

Žádný z datových typů o kterých jsme se již zmínili, by nebyl skutečně vhodný pro popis figur na šachovnici. Proto vám Pascal umožňuje vytváření vašich vlastních datových typů. To může být provedeno deklarací typu

Např.

```
type figura=(pesec,sterlec,vez,ku,kralovna,kral):

Promena typu figura pak může nabývat jednu ze šesti hodnot.
např.

var moje figura,tvoje figura,figura
begin
```

```
moje figura:=kun;
tvoje figura:=kralovna
```

Deklarace typu se musí provést za deklarací konstant a před deklarací proměnných. Identifikátory užité pro nečíselná data, jako je např. figura musí být jedinečné tj. nesmějí se objevit v jiných nečíselných typech nebo být deklarovány jako konstanty. I nečíselné typy mohou být ordinální (seraditelné), a proto např. naše figura - prvky lůžeme porovnat pomocí relačních operátorů =, menší, větší atd.

Např.

kral (větší) kralovna
kralovna (menší) kun

Jsou pro ně definovatelné také tři funkce PRED (menší) SUCC a ORD

pred(x) dává hodnotu předcházejícího x
succ(x) dává hodnotu následujícího x
ord(x) udává pozici x uvnitř datového typu (počáteční hodnota pesec=0)

a tak:

pred (vez) = strelec
succ (kun) = kralovna

ale pred (pesec) a succ (kral) nejsou definovány.
ord (strelec) = 1
ord (kun) = 3

Příklad 17 - Erastotenovo síto

Tento program vypíše všechna prvočísla od 2 do 127.

```
program Erastoteles;
const n=127;
var sito:set of 2..n;
    number,i:integer;
```

```
begin                    Místo "!" mají být hranaté závorky.
 sito:=!2..n!:
  for number:=2 to n do if number in sito then
   begin
   writeln(number):
   for i:=2 to n div number do
   sito:=sito-!i*number!
 end:
end.
```

Prvočísla jsou celá čísla dělitelná jen sama sebou a 1. Naše síto užité pro vyhledání je novým typem proměnné, která se nazývá množina.

Množiny jsou v Pascalu soubory předmětů, které uzavřeme do hranatých závorek. Předmět pak do množiny patří nebo nepatří. Proti:

```
!1,2,3!
!2,1,3!
!1,1,3,3,2! jsou navzájem ekvivalentní množiny
```

Zkrátka x..y v definici množiny či přiřazení znamená, že do množiny patří všechny položky mezi x a y včetně nich.

```
!1..4,10!=!1,2,3,4,10!
```

Pomocí operátoru IN můžeme testovat, zda předmět do množiny patří, či nikoli. Proto "4 in !1..5!" dá booleovský výsledek TRUE

Typ prvku množiny může být jakéhokoli standardního typu, kromě typu REAL. Hodnoty jsou omezeny rozsahem 0..127.

Nyní se vraťme k našemu programu. Začíná ve výpočtu od čísla 2 a pokračuje směrem vzhůru. Pokud je číslo ještě v množině sít, pak je prvočíslem. Jednoduše eliminujeme všechny součiny naposledy zjištěného prvočísla s čísly, která ještě zbývají v sítu, neboť ty nejsou prvočísla.

Operace povolené mezi dvěmi množinami:

```
x+y x sjednoceno s y
x-y rozdíl mezi x a y
x*y položky přítomné současně v x i v y
```

COMMPAS
vydavatelství, služby a distribuce
Dr. Ivan Pavlíček, Pavel Škvrně
p. o. box 80
140 00 Praha 4