PŘÍLOHA ZPRAVODAJE

ATARI®

KOUZLÍ

OLOMOUC

P ř e d m l u v a

Jistě pro vás není žádnou novinkou, že váš ATARI je mimořádně výkonný a pružný počítač. Nové je to, že pomocí ATARI KOUZLÍ můžete vytěžit ze svého počítače ještě více. Předpokladem k tomu je pouze něco znalostí jazyka BASIC a trochu dobré vůle zabývat se také programy ve strojovém kódu. V případě ATARI KOUZLÍ se nejedná o nějakou novou učebnici assembleru 6502, o tom bylo dost napsáno v PRŮVODCI ASSEMBLERU 6502, který vydal ATARI klub Olomouc. Jde spíše o sbírku zajímavých a užitečných tipů, triků, programů v BASICU a ve strojovém kódu. Protože složité záležitosti se nejlépe osvětlí příklady, má kniha i učební efekt.

Všechny programy jsou rozsáhle komentovány a vysvětleny, abyste jim rychle porozuměli. Pro ty z vás, kteří se zajímají o herní programy, obsahuje kniha dvě lahůdky, vestavitelný hudební podprogram a kompletní zvukový vývojový systém, který můžete přímo použít ve svých programech.

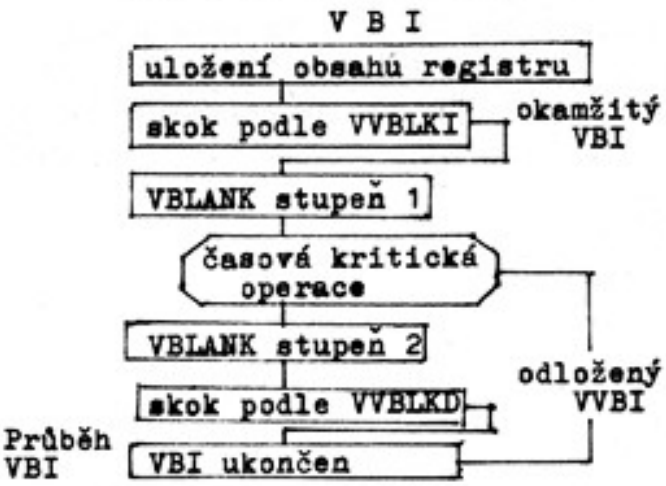
Druhá polovina knihy se více věnuje systémovým programům. Mezi jiným se dozvíte, jak programovat disketové operace ve strojovém kódu a jak můžete efektivně pracovat s výkonným DOS XL.

ATARI KOUZLÍ, by ale nebyla úplná, kdyby neobsahovala několik zvláštních bombónků. Věděli jste, že váš počítač kromě dosavadních 17 grafických modů ovládá ještě další tři? Dočtete se, jak vytvořit textové okénko v grafických modech 9 až 11 programem v BASICu, jedním z mnoha, ze kterého uvidíte, jak lze BASIC podporovat pomocí podprogramů ve strojovém kódu. Programátoři v BASICu si tedy také přijdou na své. Publikace ATARI KOUZLÍ je doplněna i kapitolami doplňujícími a rozšiřujícími informace o programech.

Programování s využitím V B I

Chcete-li programovat grafické a zvukové efekty, měli byste znát mimořádně užitečnou a výkonnou část vybavení počítače. Je to VBI (Vertical Blank Interrupt) t.j. přerušení programu během vertikálního zatemnění. Jak tomuto termínu rozumět? "Interrupt" znamená požadavek zvenčí na přerušení činnosti programu momentálně zpracovaného procesorem (nazývaného též programem v popředí) a vyvolání podprogramu přerušení. Speciálně u VBI vzniká přerušení programu a spuštění podprogramu VBI padesátkrát za sekundu. Pro vysvětlení, proč je tato vlastnost tak zajímavá pro grafiku, si musíme stručně připomenout, jak vzniká obraz monitoru, resp. na televizní obrazovce. Obraz je vykreslován řádek po řádku, počínaje od levého horního rohu. Když paprsek dojde do pravého dolního rohu stínitka, musí se zase vrátit do rohu nahoru vlevo. Během návratu se samozřejmě nezobrazuje. A právě na začátku tohoto vertikálního zatemnění se spustí VBI. Nyní již chápete, proč je VBI pro grafické použití tak důležitý: změny grafiky VBI nejsou na stínítku viditelné jako poruchy, protože v této době žádný obraz nevzniká. Tím jsou pohyby hráčů plynulé, je vůbec umožněn pohyb textu po obrazovce ap. VBI však není vyhrazen jen pro účely grafiky, nýbrž může sloužit pro všechny druhy úloh, které vznikají pereodicky, jako je měření času, vyhodnocování joysticku nebo generování zvuků a hudby. Více se o tom dočtete v následujících odstavcích.

VBI a operační systém



Po spuštění VBI je průběh programu přerušen přechodem do operačního systému. Nejprve se uloží obsah registrů do zásobníkové paměti (stack). Pak následuje skok nepřímo adresovaný vektorem VVBLKI (na adr. 222/223 L/H). Tento vektor ukazuje, pokud ovšem nebyl změněn, na rutinu VBI operačního systému, rozdělenou do dvou stupňu.:

Stupeň 1:
Zde se zvýší hodnota v hodinách reálného času na adresách x12-14, přezkouší se přepínání barev (Attractmodus) a zpracuje se systémový časovač č.1.

Stupeň 2:
Návrat obsahu registru ze zásobníku na příslušné adresy, čtení vstupů z joysticku atd.

V návaznosti na stupeň 2 následuje skok prostřednictvím vektoru RAM VVBLKD (na 224/225), který (opět jen v normálním případě) ukazuje na malý program XITVBV pro konec přerušení. Stupeň 2 a skok podle VVBLKD bude však proveden jen pokud se nejedná o časově kritickou podmínku (např. vstup/výstup z diskety, viz níže).

Proč hned dva VBI ?

Vaše šance je ve změně vektoru RAM tak, aby ukazovaly na vaše vlastní programy. Jak jste viděli existují dva takové vektory, kterými můžete zavést program do VBI: tzv. "okamžitý VBI" a "odložený VBI". Proč zrovna dvě možnosti? To lze rychle vysvětlit: VBI vzniká také během operací vstupu,např. během zavádění programu z diskety. Protože však v takové situaci nezbývá mnoho přebytečného času, VBI se zkracuje, uskuteční se jen skok dle okamžitého VBI a provede se pouze stupeň 1 rutiny ROM-VBI. Tím se dostane program ihned na XITVBV, takže stupeň 2, jenž obsahuje přece jen poněkud časově náročné kopírování stínového registru, odpadá.

Konkrétně to znamená:

1. Má-li váš program zůstat aktivní také během vstupní/výstupní operace, musíte jej zavést do "okamžitého VBI".Zajímavé je to např. v případě úvodního programu, který během zavádění vlastního programu bliká na obrazovce titulkem, nebo se stará o zcela jednoduchou animaci.
2. Ve všech ostatních případech je třeba použít "odložený VBI", který je během I/O operace vypnut, a teprve po jejím skončení se automaticky zapíná.

Kolik času máte pro výpočet?

V "okamžitém VBI" by váš program neměl trvat déle než 400µs, protože vertikální zatemňování odpovídá asi 650-ti strojovým cyklům. V odloženém VBI můžete zavádět delší programy. V žádném případě však nesmějí překročit 20 000 mikrosekund, protože po této době se již spouští další VBI. Kromě toho měli byste mít na zřeteli,že:

a) V těchto 20 000 µs je také obsažen čas, ve kterém vzniká následující obraz. Vlastně vertikální zatemnění trvá u evropských PAL ATARI jen 4500 µs. V této době lze využit asi 7400 strojových cyklů a s těmi se již dá něco podniknout. Přitom nezapomínejte, že rutina ROM-VBI z toho spotřebuje asi 900-1000 cyklů.

b) Čas pro výpočet, spotřebovaný ve VBI, jde na úkor programu v popředí. Proto máteli ve VBI časově náročný program a přitom ještě běží BASIC, zjistíte, že program v Basicu se začíná "plížit".

Ještě slovo k délce vertikálního zatemnění. Mezi americkým NTSC-ATARI a evropským PAL-ATARI je podstatný rozdíl. Zatímco obrazová frekvence je pro PAL 50 Hz, americké ATARI pracují se 60 Hz. Tím je použitelný časový úsek ve VBI podstatně zkrácen - místo 4500 µs trvá VBI pouhých 1400 µs (cca 2300 strojových cyklů). Majíli vaše programy bez závad běžet na amerických systémech, neměli byste tento limit překročit. V normálním případě můžete přesto využit o něco delší čas, protože každý DISPLAY-LIST (obrazový podprogram DL) má na počátku obsahovat třikrát osm prázdných řádků, ve kterých rovněž nevzniká žádná obrazová informace. V tomto čase asi 1500 mikrosekund (2500 cyklech),můžete také dělat změnu grafiky, které nebudou viditelné jako poruchy:

Z toho plyne:

u PAL ATARI máte 7400 - 1000+2500 = 8900 cyklů

u NTSC-ATARI 2300 - 1000+2500 = 3800 cyklů

O způsobu výpočtu cyklů programů ve strojovém kódu se dočtete v odstavci "Jak rychlý je váš ATARI".

Podíváme se tedy na problém blíže a zůstaneme nejprve u "odloženého VBI". Jak už víte, jsou v RAM dvě paměťová místa VVBLKD: $224/225 (L/H), která po zpracování ROM VBI určují skokovou adresu. Tento vektor ukazuje na malou rutinu s názvem XITVBV ($E462), v níž VBI končí. Zařídíte-li, aby tento vektor ukazoval na váš vlastní program a po jeho skočení skočíte na XITVBV, stane se vaš program součástí VBI.

Není vhodné měnit VVBLKD vlastními příkazy STA (nebo dokonce POKE v BASICu). Představte si, že nastane situace, kdy ihned po změně nižšího byte $224 vznikne přerušení. Poněvadž program je okamžitě přerušen, najde VBI vektor složený ze starého vyššího byte a nového nižšího byte, což s největší pravděpodobností povede ke zhroucení činnosti počítače.

Naštěstí nabízí operační systém počítače jednotné řešení tohoto problému: do X-registru je třeba uložit vyšší byte počáteční adresy rutiny VBI, do Y-registru nižší byte. Ke vstupu do "odloženého VBI" je nutno dát do akumulátoru číslo 7. Nakonec je nutno vyvolat ROM rutinu pro zavedení nových vektorů, podprogram SETVBV ($E45C).

```
LDX #06          ; vyší byte, např. pro stranu 6
LDY #00          ; nižší byte
LDA #7           ; pro VBI
JSR SETVBV       ; uložení nového vektoru
```

Tato rutina připojí na VBI program, který je uložen v paměti od adresy $0600 (známá stránka 6...). Váš vlastní VBI program musí obsahovat ukončení přerušení - poslední provedený příkaz musí být

```
JMP XITVBV       ; XITVBV:= $E462
```

Tím se program přerušení ukončí a bude následovat hlavní program.

Shrnutí:

K zavedení programu ve strojovém kódu do "odloženého VBI" musíte učinit následující kroky.

1. Program uložit do paměti, jeho poslední provedený příkaz musí být JMP XITVBV ($E462).
2. Vyšší byte počáteční adresy uložit do registru X, nižší do Y.

3. Do akumulátoru vložit 7 (pro odložený VBI).
4. Skočit do podprogramu JSR SETVBV (ØE45C).

Okamžitý VBI

V podstatě pro něj platí to, co bylo řečeno výše. Rozdíly jsou tyto:

1. Vektor RAM je na VVBLKI (Ø222,223 L,H)
2. Před voláním SETVBV musí být do akumulátoru uloženo číslo 6 (na rozdíl od 7 u odloženého VBI).
3. Poslední příkaz vašeho programu musí obsahovat a ukazovat na začátek rutiny ROM-VBI prostřednictvím JMP SYSVBV (ØE 45F), pokud ji ovšem nechcete přeskočit.

Vypnutí programu VBI

Funguje v zásadě stejně jako zapnutí. Použijete opět rutinu SETVBV a do příslušného vektoru tím uložíte jeho původní hodnotu.

```
Při odloženém VBI:     LDX #XITVBV/256      ;XITVBV = ØE462
                       LDY #XITVBV  255     ;LSB
                       LDA #7               ;pro odložený VBI
                       JSR SETVBV

Při okamžitém VBI:     LDX #SYSVBV/256      ;SYSVBV = ØEØ5F
                       LDY #SYSVBV  255     ;LDB
                       LDA #6               ;pro okamžitý VBI
                       JSR SETVBV
```

Stiskem SYSTEM RESET se rutiny VBI vypínají také.

Pozor:

Aby nedošlo nepříjemným překvapením, měly by rutiny VBI vždy začínat příkazem CLD, aby se procesor přepnul do normálního binárního módu. Zvláště když VBI běží paralelně s BASICem, je 6502 v okamžiku vyvolání VBI v módu BCD a tento pracovní stav rutina ROM-VBI nemění. Následkem toho příkazy sčítání a odčítání dávají jiné výsledky !

Vícehlasá hudba, ale s obalovou křivkou :

V případě, že jste někdy zkoušeli napsat program pro vícehlasou hudbu v jazyku BASIC, tak zajisté víte, že přitom narazíte na několik obtíží. Časování je obtížné, zvláště u krátkých not nesouhlasí délka a jednotlivé tóny při použití příkazu SOUND jsou slyšitelné "natlačeny" na sebe. Skutečně nemožné je v BASICu opatřit tóny obalovou křivkou nebo současně zobrazovat na obrazovce grafické znázornění. Že je něco takového, je vidět u řady programů, kde hudba zní v průběhu vykonávání vlastního programu. Strojový program uvedený v této kapitole vám ukáže, jak se to dělá a rovněž vám předvede elegantní způsob, jak na to.

Co umí tento hudební program:

Možností, které program dává, je řada
-vícehlasá hudba (až 4 tóny současně)
-každý hlas má vlastní programovatelnou obalovou křivku
-noty se zadávají pouze textem(nikoliv čísly)
-v každém kanálu je možné zahrát až 128 not
-zatím co zní hudba, je počítač zcela k dispozici uživateli jak pro program v BASICu tak ve strojovém kódu
-v případě, že jste obvzlášť muzikální, můžete pomocí tohoto programu skládat vlastní hudbu

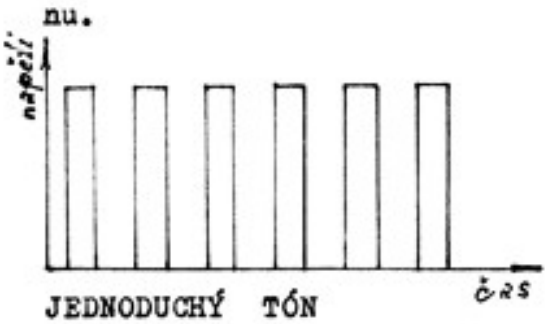
Je to umožněno použitím VBI (viz kapitola VBI), přičemž zabijeme dvě mouchy jednou ranou. Na jedné straně je možno lechce programovat délku not pomocí periodické struktury VBI, na druhé straně běží celý program kvaziparalelně, takže počítač může zpracovat jiné úkoly.

Obalová křivka

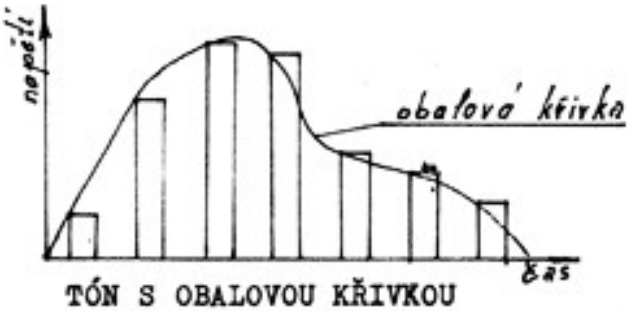
V předchozím už byla řeč o obalových křivkách - ale co to vlastně je?
K tomu několik vysvětlení:
Jednoduchý tón se skládá s periodické řady impulsů, z nichž všechny mají stejnou amplitudu. Příklad příkazu v jazyku BASIC pro takový tón je

```
SOUND Ø,5Ø,1Ø,8
```

Přičemž poslední parametr je úroveň hlasitosti (a tím amplitudy) tónu. Zní to ale jako z flašinetu.
Mnohem zajímavěji to zní, když se tónu přiřadí určitá charakteristika, tzn. náběh a dozvuk ve tvaru obalové křivky. Tím se rozumí rychlá změna hlasitosti během generování tónu.



JEDNODUCHÝ TÓN



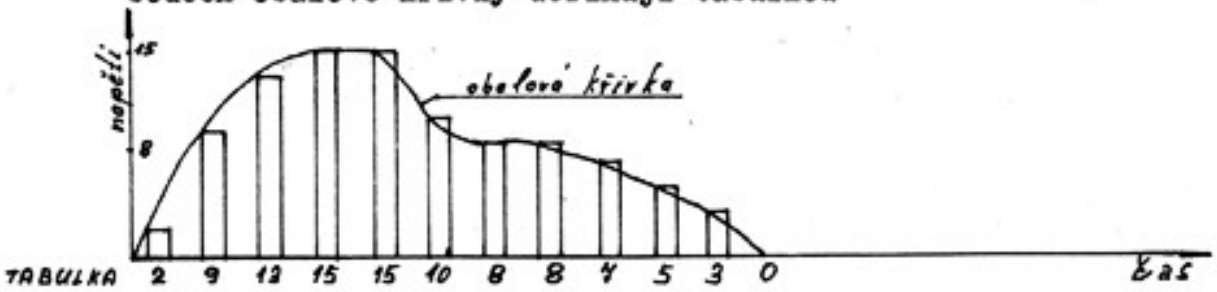
TÓN S OBALOVOU KŘIVKOU

Z obrázku je patrno, odkud se bere termín obalová křivka. Vrcholky impulsů jsou ohrazené touto křivkou. Příkladem programování obalové křivky v jazyku BASIC je

```
FOR L = 3Ø TO Ø STEP -1:SOUND Ø,5Ø,1Ø,L/2: NEXT L
```

Jde to ještě lépe

Nebudeme se samozřejmě omezovat pouze na jednoduché obalové křivky jako v jazyku BASIC. K napodobení složitějších obalových křivek se hodnoty hlasitosti v jednotlivých bodech obalové křivky definují tabulkou



Rozložení obalové křivky do tabulky

Při zaznění tónu se nastaví ukazatel na začátek této tabulky a v každém VBI se posouvá o jedno dál. Momentální hodnota hlasitosti se z tabulky vybere a zapíše se na místo bitu hlasitosti registru AUDC. Tím je možno napodobit přirozený zvuk hudebních nástrojů. Prudký náběh klavíru se vyjádří strmým nárustem křivky, zatím co zvuk flétny se dosáhne po-

malým nárustem hlasitosti. V tomto směru nestojí nic v cestě vlastní tvořivosti. Je možno experimentovat s různými obalovými křivkami, vyplatí se to.

Vkládání not

Vkládání not se dá elegantně provést v assembleru tak, že se využije jeho inteligence k překladu označení not do příslušné hodnoty frekvence. Všechny noty jsou předem definovány jako tzv. "EQUATES". Tak se nazývají konstanty, kterými se příkazem "=" uděluje hodnota (název pochází z toho, že v mnoha asemblerech se příkazu "EQU" používá místo "="). Noty se pak zapisují svým (písmenovým) označením do řádků, které začínají příkazem ".BYTE". Poněkud odlišné to je u EQUATES půltónů:

nota		způsob zápisu
C#	=	CIS
D#	=	DIS
F#	=	FIS
G#	=	GIS
A#	=	AIS
PAUSA	=	P

Za označením noty se ještě uvádí oktáva. Přípustné jsou hodnoty 3 a 4 (plus 5 pro C), přičemž 3 je nejnižší oktáva. Hodnoty kmitočtů byly převzaty z původní tabulky ATARI. Označení noty se oddělí čárkou a následuje údaj délky tónu, pro něž jsou rovněž definovány "EQUATES". K označení se ovšem používají písmena anglické abecedy:

symbol	délka noty
H :	půlová nota
Q :	čtvrťová nota
E :	osminová nota
S :	šestnáctinová nota

Prodloužené noty (s tečkou) je možno jednoduše vytvořit dělením dvou po sobě následujících délek not - např. H+Q je tečkovaná půlová nota. Prakticky vypadá kompletní zadání tónu, např. C#,3. oktáva, čtvrťová následovně:

```
.BYTE CIS 3,Q
```

Samozřejmě je možno do jednoho řádku zapsat více tónu. Kvůli přehlednosti se doporučuje psát pro každý takt zvlášť řá-

dek, např.:
.BYTE C3,H,E3,Q,P,Q (P je na místě pausy)
Uvedený příklad je pro 4/4 takt.

Přiřazení tónu jednoho kanálu se provádí tak, že před prvním příkazem .BYTE příslušného kanálu zadá STIMM 1 pro kanál 1, STIMM 2 pro kanál 2 atd. Můžete se přitom držet vzorového programu. Řada tónu jednoho kanálu se ukončí tak, že místo hodnoty tónu se zapíše ENDE. Jestliže se místo ENDE zadá START, pak se bude melodie opakovat od začátku. V případě, že se nevyužije všech čtyř kanálů, pak se označení příslušného kanálu zapíše .BYTE ENDE, např.:

STIMM 4 .BYTE ENDE — V tomto případě zůstane kanál č.4 v klidu. Přišlo-li by v tomto případě slůvko START,pak by se počítač "zasekl".

Zadání obalové křivky

Ke každému kanálu musí být zadaná vlastní obalová křivka. To se provádí zadáním číselné řady, která udává hodnoty hlasitosti po sobě následujících hodnot (jak bylo uvedeno u obr. 3 na začátku). Hodnoty se opět umístí do příkazů .BYTE a pomocí tabulek HUELL1 až HUELL4 se přiřadí jednotlivým kanálům. Každá obalová křivka se ukončí příkazem ENDE, podobně jako u hodnot tónů.

A teď k věci.
Šťastný majitel diskety, která se prodává k originálu této knihy HEXENKUCHE, si může ihned poslechnout jeden demonstrační příklad. Stačí k tomu (při zasunutém modulu BASIC) vyvolat program příkazem

RUN "D:MUSIK.BAS"

a zazní hudba. Tento tzv. BASIC-LOADER najdete za strojovým programem v přiloženém vzorovém programu. V této formě je možno tento program převzít do vlastnictví programů.Přitom nelze zamlčet dvě důležité nevýhody této metody: BASIC-loader potřebuje dlouhou dobu k tomu, aby strojový program spolu s hudebními daty uložil do paměti. Horší ale je, že v programu BASIC nemůže být zadána další melodie. BASIC-loader je vhodný pouze tehdy, když se potřebuje hotová hudební

rutina,nebo se chce poslouchat, co hudební program umí.

V případě, že chcete zadat další melodie, pak je nutné se zabývat asemblerovým zápisem. Je to možno zapsat prakticky do každého assembleru pro ATARI, pokud používá standartní kód assembleru pro 6502. V každém případě fungují: EDITOR/ASSEMBLER Cartridge, EASMD, MAC/65. Na výše uvedené disketě naleznete zdrojový program pod názvem VBIMUSIK.SRC.

HUDEBNÍ PROGRAM VE STROJOVÉM KÓDU :

Vyvolání tohoto programu se provádí skokem na začátek programu ($98ØØ,38912 dek.) tak, že příkaz k vyvolání hudby v BASICu má tvar:

```
X = USR(38912)
```

Hudba se vypíná rovněž skokem na příslušnou rutinu, která je vzdálená 4 byte od začátku programu. Odpovídající volání v jazyku BASIC je:

```
X = USR(38916)
```

Přirozeně je možné vyvolat hudební program i ve strojovém kódu. V tom případě je ale nutné přeskočit příkazy PLA,které jsou potřebné pouze pro příkaz USR v BASICu. Příslušný strojový program může vypadat následovně.:

```
MEIN    =   $98Ø1      hudba zapnuta
MAUS    =   $98Ø5      hudba vypnuta
......
JSR MEIN               hudba zní
......
JMP MAUS               hudba opět vypnuta
......
```

Běh hudebního strojového programu

První příkaz skoku JMP, který, jak bylo právě uvedeno, slouží k uvedení do chodu, skáče na podprogram MUSEIN (hudba zapnuta). Zde se vymaže obsah tónových registrů a vnitřní proměnné programu, Pak se do VBI zavede programová část MUSVBI. Vše další již probíhá "mimochodem" ve VBI. Podprogram MUSEIN končí příkazem RTS a tím se vrací řízení počí-

tače na původní program, který hudební program vyvolal. Rutina VBI podprogramu MUSVBI je od chvíle spuštění aktivována každou 1/50 sekundy. Obsahuje v podstatě smyčku, která pro každý zvukový kanál proběhne jedenkrát, celkem tedy čtyřikrát. V této smyčce jsou vždy vyvolány dva podprogramy MUXPLY a MUXHUL a aktuální číslo kanálu je uloženo v registru X.

MUXPLY vybírá správné pořadí not. Při každém vyvolání zpracovává zvukový kanál daný registrem X. Průběh je následující.:

1. Kontrola, zda není na tomto kanálu generován tón. Pokud ne, přejde na bod 3.
2. Je-li generován tón, pak se trvání tónu (MODUR1-4) sníží o jedničku. Je-li trvání tónu větší než 0, pak zde MUXPLY končí.
3. Dojde-li program na toto místo (MUXNEU), pak dosavadní tón skončil a z tabulky tónů se vybere další. K tomu se zjistí adresa kanálu X ze STMBL/H a pomocí ukazatele tónu MUPNT1-4 se tento tón vybere z tabulky not. Dále se na začátku nového tónu nastaví ukazatel obalové křivky na nulu, tzn. na začátek tabulky obalových křivek.
4. Nyní se testuje, zda se skutečně jedná o hodnotu tónu, nebo snad o kód konce nebo startu. Pokud je to START,pak se ukazatel tónu nastaví opět na nulu a příslušný tón začíná znovu. Při instrukci ENDE se ukazatel tónu neposouvá dále,takže při dalším běhu smyčky se opět rozpozná konec (ENDE), kromě toho se vynuluje příkaz PAUSE, čímž se tento kanál uzavře.
5. Pokud se jedná o normální hodnotu tónu (včetně PAUSE),pak se jeho délka trvání přečte z tabulky tónů,ukazatel tónu se posune dál a výška tónu se předá POKEY. Pokud se rozpoznal příkaz PAUSE, pak se pro generátor obalové křivky nastaví nula, takže tento generátor zůstane nevyužit.

Průběh programu obalové křivky je poněkud jednodušší:

1. Je-li příznak nastaven na PAUSE, nastaví se hlasitost příslušného kanálu na 0.
2. V opačném případě se načte příští hodnota hlasitosti z obalové křivky podle základní adresy (z MHLTBL/H) a uka-

zatele obalové křivky (MUHPT1-4) doplněné bity udávajícími zkreslení čistého tónu a předá se dále na POKEY. Pokud ale byla indikována hodnota "ENDE" - konec (z tabulky obalové křivky), pak se POKEY nezmění a zpracování zde končí. Toho se dá využít tak, že se krátká obalová křivka nechá ukončit hodnotou větší než 0, a tím se vytvoří úhoz, který přechází do konstantního tónu. Tabulka obalové křivky se tak udrží relativně malá.

3. Ukazatel obalové křivky se posune na další hodnotu v tabulce MUXAUS. Zde se hudební VBI opět vypne a event. se vypnou i zbývající znějící tóny.

```
          Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          Ø11Ø ;x          HUDBA VE VBI                  X
          Ø13Ø ;x Hudební program s generátorem          x
          Ø14Ø ;x        obalové křivky                  x
          Ø15Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          Ø16Ø ;
          Ø17Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          Ø18Ø ;x  Definice not/kmitočtu a délky not     x
          Ø19Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
=ØØF3     Ø2ØØ C3     =   243
=ØØE6     Ø21Ø CIS3   =   23Ø
=ØØD9     Ø22Ø D3     =   217
=ØØCC     Ø23Ø DIS3   =   2Ø4
=ØØC1     Ø24Ø E3     =   193
=ØØB6     Ø25Ø F3     =   182
=ØØAD     Ø26Ø FIS3   =   173
=ØØA2     Ø27Ø G3     =   162
=ØØ99     Ø28Ø GIS3   =   153
=ØØ9Ø     Ø29Ø A3     =   144
=ØØ88     Ø3ØØ AIS3   =   136
=ØØ8Ø     Ø31Ø B3     =   128
=ØØ79     Ø32Ø C4     =   121
=ØØ72     Ø33Ø CIS4   =   114
=ØØ6C     Ø34Ø D4     =   1Ø8
=ØØ66     Ø35Ø DIS4   =   1Ø2
=ØØ6Ø     Ø36Ø E4     =   96
=ØØ5B     Ø37Ø F4     =   91
=ØØ55     Ø38Ø FIS4   =   85
=ØØ51     Ø39Ø G4     =   81
=ØØ4C     Ø4ØØ GIS4   =   76
=ØØ48     Ø41Ø A4     =   72
=ØØ44     Ø42Ø AIS4   =   68
=ØØ4Ø     Ø43Ø B4     =   64
=ØØ3C     Ø44Ø C5     =   60
          Ø45Ø ;
=ØØFF     Ø46Ø PAUSE  =   255       kód pro pausu
=ØØFE     Ø48Ø START  =   254       kód pro spuštění hlasu
=ØØFD     Ø49Ø ENDE   =   253       kod pro ukončení hlasu
          Ø5ØØ ;
=ØØØ6     Ø51Ø DAUER  =   6         trvání šestnáctinové noty
=ØØFF     Ø47Ø P      =   PAUSE     zkratka pro pausy
```

```
=ØØ3Ø        Ø53Ø H        =   DAUER*8      půlová nota
=ØØ18        Ø54Ø Q        =   DAUER*4      čtvrtová nota
=ØØØC        Ø55Ø E        =   DAUER*2      osminová nota
=ØØØ6        Ø56Ø S        =   DAUER
             Ø57Ø ;
             Ø58Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø59Ø ;x  Systémové konstanty/adresy O.S.   x
             Ø6ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø61Ø ;
=ØØ1Ø        Ø62Ø HI       =   $Ø1ØØ        HI-byte
=ØØFF        Ø63Ø LO       =   $FF          LO-byte
             Ø64Ø ;
=E45C        Ø65Ø SETVBV   =   $E45C
=E462        Ø66Ø XITVBV   =   $E462
=D2ØØ        Ø68Ø AUDF1    =   $D2ØØ
=D2Ø1        Ø69Ø AUDC1    =   $D2Ø1
=D2Ø8        Ø7ØØ AUDCTL   =   $D2Ø8
=D2ØF        Ø71Ø SKCTL    =   $D2ØF
             Ø72Ø ;
             Ø73Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø74Ø ;x  Vnitřní proměnné programu         x
             Ø75Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø76Ø ;
=9BEØ        Ø77Ø MUDATA   =   $9BEØ        rozsah proměných
=9BEØ        Ø79Ø MUPNT1   =   MUDATA       ukazatel v tab.not
=9BE4        Ø8ØØ MUDUR1   =   MUDATA+4     čítač délky not
=9BE8        Ø81Ø MUPAU1   =   MUDATA+8     flag pro pausu
=9BEC        Ø82Ø MUHPT1   =   MUDATA+12    ukazatel v obal.křivce
             Ø83Ø ;
=ØØCE        Ø84Ø CSREGL   =   $CE          registr na nulté stránce
=ØØCF        Ø85Ø CSREGH   =   $CF
             Ø86Ø ;
             Ø87Ø ;
             Ø88Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø89Ø ;Tabulka skoků,bude zavolána z BASICu
             Ø9ØØ ; Zapnutí: A = USR(38912)
             Ø91Ø ; Vypnutí: A = USR(38916)
             Ø92Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø93Ø ;
ØØØØ         Ø94Ø        *=  98ØØ
98ØØ 68      Ø95Ø        PLA                pro BASIC,volání přes
                                            USR
98Ø1 4C1998  Ø96Ø            JMP MUSEIN     zapnutí hudby
98Ø4 68      Ø97Ø            PLA            pro BASIC,volání
                                            přes USR
98Ø5 4C4398  Ø98Ø            JMP MUSAUS     vypnutí hudby
             Ø99Ø ;
             1ØØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             1Ø1Ø ; VBI rutina,bude volána každou 1/5Ø s
             1Ø2Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             1Ø3Ø ;
98Ø8 D8      1Ø4Ø MUSVBI CLD                dekadický mod
98Ø9 A2ØØ    1Ø5Ø        LDX #Ø             kanál Ø hraje nejdřív
98ØB 2Ø5A98  1Ø6Ø MUNXKN JSR MUXPLY         hrát jeden hlas>>
98ØE 2ØA998  1Ø7Ø        JSR MUXHUL         obalová křivka pro
                                            jeden hlas>>
9811 E8      1Ø8Ø            INX            následující hlas
9812 EØØ4    1Ø9Ø            CPX #4         už všechny 4 Hlasy?
9814 DØF5    11ØØ            BNE MUNXKN     ne --->
9816 4C62E4  112Ø        JMP XITVBV         pak VBI ukončit ===>
```

```
                 1140 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 1150 ;x Inicializační rutina              x
                 1160 ;x Inic.POKEY,vymazání prom,nastavení VBI
                 1170 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 1180 ;
9819 A900        1190 MUSEIN LDA #0          vymazat POKEY a prom.
981B A203        1200        LDX #3
981D 9D00D2      1220 MUSEN1 STA AUDF1,X     AUDF/C 1/2
9820 9D04D2      1230        STA AUDF1+4,X   AUDF/C 3/4
9823 9DE49B      1240        STA MUDUR1,X    délka tonu :=0
9826 9DE09B      1250        STA MUPNT1,X    ukaz.not :=počátek tab.
9829 9DEC9B      1260        STA MUHPT1,X    ukaz.obal.křivky :
                                             poč.tabulky
982C CA          1270        DEX             už všechny 4?
982D 10EE        1280        BPL MUSEN1      ne,znovu --->
                 1300 ;
982F A903        1310        LDA #3          POKEY nastavit zpět
9831 8D0FD2      1320        STA SKCTL
9834 A900        1330        LDA #0          zvukový řídící reg.zpět
9836 BD08D2      1340        STA AUDCTL
                 1350 ;
9839 A008        1360        LDY #MUSVBI LO nový vektor VBI("SB)
983B A298        1370        LDX #MUSVBI/HI MSB
983D A907        1380        LDA #7          pro odložení VBI
983F 205CE4      1390        JSR SETVBV      rutina O.S.,nastav VBI
9842 60          1400        RTS
                 1410 ;
                 1420 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 1430 ;x Ukončení VBI                      x
                 1440 ;x Vektor bude vrácen na původní hodnotu
                 1450 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 1460 ;
9843 A062        1470 MUSAUS LDY #XITVBV&LO původ.vektor VBI(LSB)
9845 A2E4        1480        LDX #XITVBV/HI MSB
9847 A907        1490        LDA #7
9849 205CE4      1500        JSR SETVBV      >>>
984C A900        1520        LDA #0          hlas vypnout
984E A203        1530        LDX #3          všechny 4 kanály
9850 9D00D2      1540 MUSAU1 STA AUDF1,X     AUDF/C 1/2
9853 9D04D2      1550        STA AUDF1+4,X   AUDF/C 3/4
9856 CA          1560        DEX             už všechny 4?
9857 10F7        1570        BPL MUSAU1      ne,znovu --->
9859 60          1580        RTS
                 1590 ;
                 1600 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 1610 ; Hraní jednoho kanálu
                 1620 ; Zkoušet délku tomu,event.zajistit no-
                        vou notu
                 1630 ; Test na KONEC, START, PAUSA
                 1640 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 1650 ;
985A BDE49B      1660 MUXPLY LDA MUDUR1,X    je kanál ještě aktivní?
985D F005        1670        BEQ MUXNEU      né,pak nová nota
985F DEE49B      1690        DEC MUDUR1,X    jinak jen délku tonu-1
9862 D03E        1700        BNE MUXPEN      ton ještě trvá --->
                 1710 ;
9864 A900        1720 MUXNEU LDA #0          flag pro pausu nejprve
9866 9DE89B      1730        STA MUPAU1,X    vynulovat,nastavit
9869 9DEC9B      1740        STA MUHPT1,X    ukazatel obálky
986C BDD898      1750        LDA STMTBL,X    výchozí adresa tab.
```

```
986F B5CE      1760          STA CSREGL     not na nulté stránce
9871 DDDC98    1770          LDA STMTBH,X
9874 85CF      1780          STA CSREGH
               1790 ;
9876 BCE09B    1800          LDY MUPNT1,X   ukazatel v tab.not
9879 B1CE      1810          LDA (CSREGL),Y výška tonu
987B C9FE      1820          CMP #START     příkaz START?
987D D008      1830          BNE MUXN1      ne --->
987F A900      1850          LDA #0         ukazatel not nastavit
9881 9DE09B    1860          STA MUPNT1,X   na počátek tabulky
9884 4C6498    1870          JMP MUXNEU     a znovu ===>
               1880 ;
9887 C9FD      1890 MUXN1    CMP #ENDE      příkaz KONEC ?
9889 F018      1900          BEQ MUXPAU     ano,pak imitace pausy
               1910 ;
988B 48        1920          PHA            poznam.výšky tónu
988C C8        1930          INY            délku z tab.not
988D B1CE      1940          LDA (CSREGL),Y číst
988F 9DE49B    1950          STA MUDUR1,X   v čítačí pro délku not
9892 C8        1960          INY            ukazatel not směrpvat
9893 98        1970          TYA            na další notu
9894 9DE09B    1980          STA MUPNT1,X   a uložit do paměti
               1990 ;
9897 8A        2000          TXA            adresy pro POKEY
9898 0A        2010          ASL A          index 2 krát
9899 A8        2020          TAY
989A 68        2030          PLA            znovu vyzvednout výšku
                                            tonu
989B C9FF      2040          CMP #PAUSE     příksz PAUSE?
989D F004      2050          BEQ MUQPAU     ano,pak vytvoření pausy
               2060 ;
989F 9900D2    2070          STA AUDF1,Y    jinak noty v POKEY
98A2 60        2080 MUXPEN   RTS
               2090 ;
98A3 A9FF      2100 MUXPAU   LDA #PAUSE     vyp.obal. křivky
98A5 9DE89B    2110          STA MUPAU1,X
98A8 60        2120          RTS
               2130 ;
               2140 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               2150 ;Generátor obalové křivky
               2160 ;Hlasitost pro kanál nastavit úměrně
               2170 ;obalové křivce
               2180 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               2190 ;
98A9 BDE89B    2200 MUXHUL   LDA MUPAU1,X   flag pro pausu
98AC C9FF      2210          CMP #PAUSE     je pausa?
98AE D005      2220          BNE MUXH1      né,norm.obal.křivka
98B0 A900      2240          LDA #0         pausa:nulová hlasit.
98B2 48        2250          PHA            hlasitost poznamenat
98B3 F019      2260          BEQ MUXH2      (vždy!) v POKEY --->
               2270 ;
98B5 BDE098    2280 MUXH1    LDA MHLTBL,X   výchozí adr.obal.křivky
98B8 85CE      2290          STA CSREGL     v regist.na nulové stránce
98BA BDE498    2300          LDA MHLTBH,X   MSB
98BD 85CF      2310          STA CSREGH
98BF BCEC9B    2320          LDY MUHPT1,X   aktual v obal.křivce
98C2 B1CE      2330          LDA (CSREGL),Y nový oporný bod
```

```
98C4 C9FD      2340          CMP #ENDE      je obal.křivka ukončena?
98C6 F00F      2350          BEQ MUXHEN     ano,pak zakončit!
               2360 ;
98C8 48        2370          PHA            oporný bod poznamenat
98C9 C8        2380          INY            ukazatel obal.křivky
98CA 98        2390          TYA            uložit do paměti
98CB 9DEC9B    2400          STA MUHPT1,X
               2410 ;
98CE 8A        2420 MUXH2    TXA            odchyl.pro reg. pokey
98CF 0A        2430          ASL A          index 2 krát
98D0 A8        2440          TAY
98D1 68        2450          PLA            opor.bod obal.křivky
98D2 09A0      2460          ORA #10*16     POKEY mod:čistý ton
98D4 99D1D2    2470          STA AUDC1,Y    uložit do POKEY
98D7 60        2490 MUXHEN   RTS
               2500 ;
               2510 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               2520 ;Tabulky výchozích adres tabulek not
               2530 ;a obalových křivek
               2540 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               2550 ;
98D8 E85FD64F  2560 STMTBL .BYTE  STIMM1&LO,STIMM2&LO,STIMM3&LO,
                                  STIMM4&LO
98E4 9899999A  2570 STMTBH .BYTE  STIMM1/HI,STIMM2/HI,STIMM3/HI,
                                  STIMM4/HI
               2580 ;
98E0 5068880AB 2600 MHLTBL .BYTE HUELL1&LO,HUELL2&LO,HUELL3&LO,
                                 HUELL4&LO
98E4 9A9A9A9A  2610 MHLTBH .BYTE HUELL1/HI,HUELL2/HI,HUELL3/HI,
                                 HUELL4/HI
               2620 ;
               2630 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               2640 ;Tabulky not:
               2650 ;každá tabulka začíná Label STIMM <n>
               2660 ;kde <n> je číslo hlasu (1-4),potom následují
               2670 ;noty (nota.délka) v příkazu .BYTE.
               2690 ;Jako poslední nota musí být "ENDE" nebo
                     "START"
               2700 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               2720 ;
98E8           2730 STIMM1
               2740 ;
98E8 60186018  2750          .BYTE E4,Q,E4,Q,D4,Q,D4,E,E4,E
98EC 6C186C0C
98F0 600C
98F2 5B185B18  2760          .BYTE F4,Q,F4,Q,E4,Q,E4,Q
98F6 60186018
98FA 600C6C0C  2770          .BYTE E4,E,D4,E,E4,E,F4,E,G4,Q,F4,
                                   E,E4,E
98FE 600C5B0C
9902 51185B0C
9906 600C
9908 6C2479 0C 2780          .BYTE D4,Q+E,C4,E,C4,H
990C 7930
990E 60186018  2790          .BYTE E4,Q,E4,Q,F4,Q,F4,E,E4,E
9912 5B185B0C
9916 600C
9918 6C186C18  2800          .BYTE D4,Q,D4,Q,E4,Q,E4,E,D4,E
```

```
991C 6Ø186ØØC
992Ø 6CØC
9922 79ØC8ØØC 281Ø          .BYTE C4,E,B3,E,C4,E,D4,E,E4,Q,
                                  D4,E,C4,E
9926 79ØC6CØC
992A 6Ø186CØC
992E 79ØC
993Ø 8Ø249ØØC 282Ø          .BYTE B3,Q+E,A3,E,A3,H
9934 9Ø3Ø
9936 51185118 283Ø          .BYTE G4,Q,G4,Q,A4,Q,F4,E,E4,E
993A 48185BØC
993E 6ØØC
994Ø 6C186C18 284Ø          .BYTE D4,Q,D4,Q,G4,Q,F4,E,D4,E
9944 51185BØC
9948 6CØC
994A 79ØC6CØC 285Ø          .BYTE C4,E,D4,E,E4,E,F4,E,G4,Q,
                                  F4,E,E4,E
994E 6ØØC5BØC
9952 51185BØC
9956 6ØØC
9958 6C2479ØC 286Ø          .BYTE D4,Q+E,C4,E,C4,H
995C 793Ø
995E FD       287Ø          .BYTE ENDE
              288Ø ;
              289Ø STIMM2
              29ØØ ;
995F 79187918 291Ø          .BYTE C4,Q,C4,Q,B3,Q,B3,E,C4,E
9963 8Ø188ØØC
9967 79ØC
9969 6C186C18 292Ø          .BYTE D4,Q,D4,Q,C4,Q,C4,Q
996D 79187918
9971 79ØC8ØØC 293Ø          .BYTE C4,E,B3,E,C4,E,D4,E,E4,Q,
                                  D4,E,C4,E
9975 79ØC6CØC
9979 6Ø186CØC
997D 79ØC
997F 79188Ø18 294Ø          .BYTE C4,Q,B3,Q,C4,H
99B3 793Ø
9985 79187918 295Ø          .BYTE C4,Q,C4,Q,D4,Q,D4,E,C4,E
9989 6C186CØC
998D 79ØC
998F 8Ø188Ø18 296Ø          .BYTE B3,Q,B3,Q,C4,Q,C4,E,B3,E
9993 791879ØC
9997 8ØØC
9999 9ØØC99ØC 297Ø          .BYTE A3,E,GIS3,E,A3,E,B3,E,C4,Q,
                                  B3,E,A3,E
999D 9ØØC8ØØC
99A1 79188ØØC
99A5 9ØØC
99A7 9Ø189918 298Ø          .BYTE A3,E,GIS3,Q,A3,H
99AB 9Ø3Ø
99AD 6Ø186Ø18 299Ø          .BYTE E4,Q,E4,Q,F4,Q,D4,E,C4,E
99B1 5B186CØC
99B5 79ØC
99B7 8Ø188Ø18 3ØØØ          .BYTE B3,Q,B3,Q,E4,Q,C4,E,B3,E
99BB 6Ø1879ØC
99BF 8ØØC
99C1 9ØØC8ØØC 3Ø1Ø          .BYTE A3,E,B3,E,C4,E,D4,E,E4,Q,
                                  D4,E,C4,E
```

```
99C5 79ØC6CØC
99C9 6Ø186CØC
99CD 79ØC
99CF 79188Ø18 3Ø2Ø          .BYTE C4,Q,B3,Q,C4,H
99D3 793Ø
99D5 FD       3Ø3Ø          .BYTE ENDE
              3Ø4Ø ;
99D6          3Ø5Ø STIM3
              3Ø6Ø ;
99D6 F3ØCD9ØC 3Ø7Ø          .BYTE C3,E,D3,E,E3,E,F3,E,G3,Q,
                                  G3,Q
99DA C1ØCB6ØC
99DE A218A218
99E2 D9ØCC1ØC 3Ø8Ø          .BYTE D3,E,E3,E,F3,E,G3,E,A3,Q,
                                  A3,Q
99E6 B6ØCA2ØC
99EA 9Ø189Ø18
99EE A218B618 3Ø9Ø          .BYTE G3,E,F3,Q,E3,Q,F3,Q
99F2 C118B618
99F6 A218A218 31ØØ          .BYTE G3,Q,G3,Q,C3,H
99FA F33Ø
99FC 9ØØCA2ØC 311Ø          .BYTE A3,E,G3,E,F3,E,E3,E,D3,Q,D3,Q
9AØØ B6ØCC1ØC
9AØ4 D918D918
9AØ8 A2ØCB6ØC 312Ø          .BYTE G3,E,F3,E,E3,E,D3,E,C3,Q,C3,Q
9AØC C1ØCD9ØC
9A1Ø F318F318
9A14 B618B818 313Ø          .BYTE F3,Q,F3,Q,E3,Q,A3,Q
9A18 C1189Ø18
9A1C D918C118 314Ø          .BYTE D3,Q,E3,Q,A3,H
9A2Ø 9Ø3Ø
9A22 79ØC8ØØC 315Ø          .BYTE C4,E,B3,E,A3,E,G3,E,F3,Q,F3,Q
9A26 9ØØCA2ØC
9A2A B618B618
9A2E A2ØCB6ØC 316Ø          .BYTE G3,E,F3,E,E3,E,D3,E,C3,Q,C3,Q
9A32 C1ØCD9ØC
9A36 F318F318
9A3A 9Ø18A2ØC 317Ø          .BYTE A3,Q,G3,E,F3,E,E3,E,C3,E,
                                  D3,E,E3,E
9A3E B9ØCC1ØC
9A42 F3ØCD9ØC
9A46 C1ØC
9A48 B618A218 318Ø          .BYTE F3,Q,G3,Q,C3,H
9A4C F33Ø
9A4E FD       319Ø          .BYTE ENDE
              32ØØ ;
              321Ø ;
9A4F          323Ø STIMM4
              324Ø ;
9A4F FD       325Ø          .BYTE ENDE
              326Ø ; Není použito v příkladu
              327Ø ;
              328Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              329Ø ; Tabulky obalových křivek
              33ØØ ; Zde budou operné body obal.křivek uloženy
              331Ø ; jako hodnoty hlasit.Každá tab. začíná
              332Ø ; "HUELL <n>" a končí "ENDE"
              333Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              335Ø ;
```

```
9A5Ø ØAØAØ8Ø7 336Ø HUELL1 .BYTE 1Ø,1Ø,8,7,6,9,8,8,8,7,7,7
9A54 Ø6Ø9Ø8Ø8
9A58 Ø8Ø7Ø7Ø7
9A5C Ø6Ø6Ø6Ø5 337Ø        .BYTE 6,6,6,5,5,4,3,2,1,1,Ø,ENDE
9A6Ø Ø5Ø4Ø3Ø2
9A64 Ø1Ø1ØØFD
9A68 Ø2Ø6Ø8Ø8 338Ø HUELL2 .BYTE 2,6,8,8,8,7,7,7,6,6,6,6
9A6C Ø8Ø7Ø7Ø7
9A7Ø Ø6Ø6Ø6Ø6
9A74 Ø5Ø5Ø4Ø4 339Ø        .BYTE 5,5,4,4,4,3,2,2,2,1,Ø,ENDE
9A78 Ø4Ø3Ø2Ø2
9A7C Ø2Ø1ØØFD
9A8Ø Ø2Ø4Ø6Ø8 341Ø HUELL3 .BYTE 2,4,6,8,9,1Ø,9,8,8,7,7,6
9A84 Ø9ØAØ9Ø8
9A88 Ø8Ø7Ø7Ø6
9A8C Ø6Ø6Ø6Ø6 342Ø        .BYTE 6,6,6,6,5,5,5,5,5,5,5,5,5,5
9A9Ø Ø5Ø5Ø5Ø5
9A94 Ø5Ø5Ø5Ø5
9A98 Ø5Ø5
9A9A Ø4Ø4Ø4Ø4 343Ø       .BYTE 4,4,4,4,4,4,4,4,4,4,2,1,Ø,ENDE
9A9E Ø4Ø4Ø4Ø4
9AA2 Ø4Ø4Ø2Ø1
9AA6 ØØFD
9AA8 FD        344Ø HUELL4 .BYTE ENDE
               345Ø ;
```

Použití ve vlastních programech :

Je více možností, jak dostat hudbu do vlastních programů:

1. Assemblerový. OBJ. soubor se přemění na BASIC-loader,který je pak možno převzít do vlastního programu pomocí příkazu ENTER. Program DATGEN na konci této knihy to umožní. Nevýhody této metody již byly popsány. Přednost této metody je v tom, že dostaneme jediný uzavřený program.
2. Načtení strojového programu z diskety. K tomu se použije pomocný program k zavádění binárních programů - viz. odstavec BLS ... Nevýhoda: Program se skládá z více částí.
3. Použití souboru EXECUTE pod DOS XL. Strojový program je pak načten příkazem DOS-LOAD. Tato metoda se výborně hodí pro vývojové fáze programu. Bližší vysvětlení je možno najít v kapitole Dávkové zpracovaní a BCOM. STARTUP.EXC na přiložené diskětě k originálu také pracuje na tomto principu.

Poznámka: U všech tří metod se musí dbát na to, aby paměť od adresy Ø98ØØ byla chráněna před přístupem z jiných programů. Nejjednodušeji toho lze dosáhnout příkazy.

```
POKE 1Ø6,144:GR.Ø:REM rezervovat 4 kB
```

Několik rad

Pomocí tohoto programu je možno z ATARI vyloudit docela pěknou hudbu. Každý program při použití dobrých akustických efektů získá na úrovni.

Je ještě řada možností, jak program vylepšit nebo zkrátit.

- Zkrácení tabulky obalové křivky tzv. průběhem ADSR (Attack-Decay-Sustain-Release). Obalová křivka se tím dá vložit pomocí max. 4 byte.
- Zabudování tzv. SLIDE efektu. Kmitočet při přechodu mezi dvěma tóny se nemění skokem, ale plynule přechází
- Zabudováním efektu "vibráto"
- Rozšíření o basové tóny, které je možno vytvořit pomocízvukového módu 12.

Jak je vidět, možností je řada. Pokud uvedené příklady přispěly k inspiraci, pak zbývá jen popřát hodně úspěchů při programování vlastního hudebního generátoru.

```
31ØØØ REM x BASIC-Loader pro zvukový program VBI
31Ø2Ø POKE 1Ø6,144:GRAPHICS Ø:POSITION 5,7:?"Inicializace"
31Ø3Ø GOSUB 32ØØØ
31Ø4Ø GRAPHICS Ø
31Ø5Ø A=USR(MUEIN)
31Ø9Ø END
32ØØØ REM x Načítání binárního programu
32Ø1Ø S=Ø:RESTORE 321ØØ
32Ø2Ø FOR A=38912 TO 39592:READ D:POKE A,D: S=S+D: NEXT A
32Ø3Ø IF S<> 63952 THEN ? "Chyba v datech!": STOP
32Ø35 REM Při kontrole vyšlo S=63799,program fungoval!
          (Poznámka překl.)
32Ø4Ø MUEIN=38912:MUAUS=MUEIN+4
32Ø9Ø RETURN
321ØØ DATA 1Ø4,76,25,152,1Ø4,76,67,152,216,162,Ø,32,9Ø,152,
       32,169
3211Ø DATA 152,232,224,4,2Ø8,245,76,98,228,169,Ø,162,3,157,
       Ø,21Ø,157
3212Ø DATA 4,21Ø,157,228,155,157,224,155,157,236,155,2Ø2,16,
       238,169
3213Ø DATA 3,141,15,21Ø,169,Ø,141,8,21Ø,16Ø,8,162,152,169,7,
       32,92
3214Ø DATA 228,96,16Ø,98,162,228,169,7,32,92,228,169,Ø,162,
       3,157,Ø
3215Ø DATA 21Ø,157,4,21Ø,2Ø2,16,247,96,189,228,155,24Ø,5,222,
       228,155
3216Ø DATA 2Ø8,62,169,Ø,157,232,155,157,236,155,189,216,152,
       133,2Ø6
3217Ø DATA 189,22Ø,152,133,2Ø7,188,224,155,177,2Ø6,201,254,
       2Ø8,8,16
3218Ø DATA Ø,157,224,155,76,1ØØ,152,2Ø1,253,24Ø,24,72,2ØØ,
       177,2Ø6
```

```
32190 DATA 157,228,155,200,152,157,224,155,138,10,168,104,
      201,255
32200 DATA 240,4,153,0,210,96,169,255,157,232,155,96,189,
      232,155,201
32210 DATA 255,208,5,169,0,72,240,25,189,224,152,133,206,
      189,228,152
32220 DATA 133,207,188,236,155,177,206,201,253,240,15,72,
      200,152,157
32230 DATA 236,155,138,10,168,104,9,160,153,1,210,96,232,95,
      214,79
32240 DATA 152,153,153,154,80,104,128,168,154,154,154,154,
      96,24,96
32250 DATA 24,108,24,108,12.96,12,91,24,91,24,96,24,96,24,96,
      12,108
32260 DATA 12,96,12,91,12,81,24,91,12,96,12,108,36,121,12,
      121,48,96
32270 DATA 24,96,24,91,24,91,12,96,12,108,24,108,24,96,24,
      96,12,108
32280 DATA 12,121,12,128,12,121,12,108,12,96,24,108,12,121,
      12,128
32290 DATA 36,144,12,144,48,81,24,81,24,72,24,91,12,96,12,
      108,24,108
32300 DATA 24,81,24,91,12,108,12,121,12,108,12,96,12,91,12,
      81,24,91
32310 DATA 12,96,12,108,36,121,12,121,48,253,121,24,121,24,
      128,24
32320 DATA 128,12,121,12,108,24,108,24,121,24,121,24,121,12,
      128,12
32330 DATA 121,12,108,12,96,24,108,12,121,12,121,24,128,24,
      121,48
32340 DATA 121,24,121,24,108,24,108,12,121,12,128,24,128,24,
      121,24
32350 DATA121,12,128,12,144,12,153,12,144,12,128,12,121,24,
      128,12
32360 DATA 144,12,144,24,153,24,144,48,96,24,96,24,91,24,108,
      12,121
32370 DATA 12,128,24,128,24,96,24,121,12,128,12,144,12,128,
      12,121
32380 DATA 12,108,12,96,24,108,12,121,12,121,24,128,24,121,
      48,253
32390 DATA 243,12,217,12,193,12,182,12,162,24,162,24,217,12,
      193,12
32400 DATA 182,12,162,12,144,24,144,24,162,24,182,24,193,24,
      182,24
32410 DATA 162,24,162,24,243?48,144,12,162,12,182,12,193,12,
      217,24
32420 DATA 217,24,162,12,182,12,193,12,217,12,243,24,243,24,
      182,24
32430 DATA 182,24,193,24,144,24,217,24,193,24,144,48,121,12,
      128,12
32440 DATA 144,12,162,12,182,24,182,24,162,12,182,12,193,12,
      217,12
32450 DATA 243,24,243,24,144,24,162,12,182,12,193,12,243,12,
      217,12
32460 DATA 193,12,182,24,162,24,243,48,253,253,10,10,8,7,6,
      9,8,8,8
32470 DATA7,7,7,6,6,6,5,5,4,3,2,1,1,0,253,2,6,8,8,8,7,7,7,6,
      6,6,6
32480 DATA 5,5,4,4,4,3,2,2,2,1,0,253,2,4,6,8,9,10,9,8,8,7,
```

```
      7,6,6,6
32490 DATA 6,6,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,2,1,
      0,253,253
```

Kreslící tabulka ATARI

Nejuniverzálnější a nejjednoduší zařízení pro zadávání dat do počítače ATARI.

Programování tabulky

Jsou dvě možnosti zadávání :

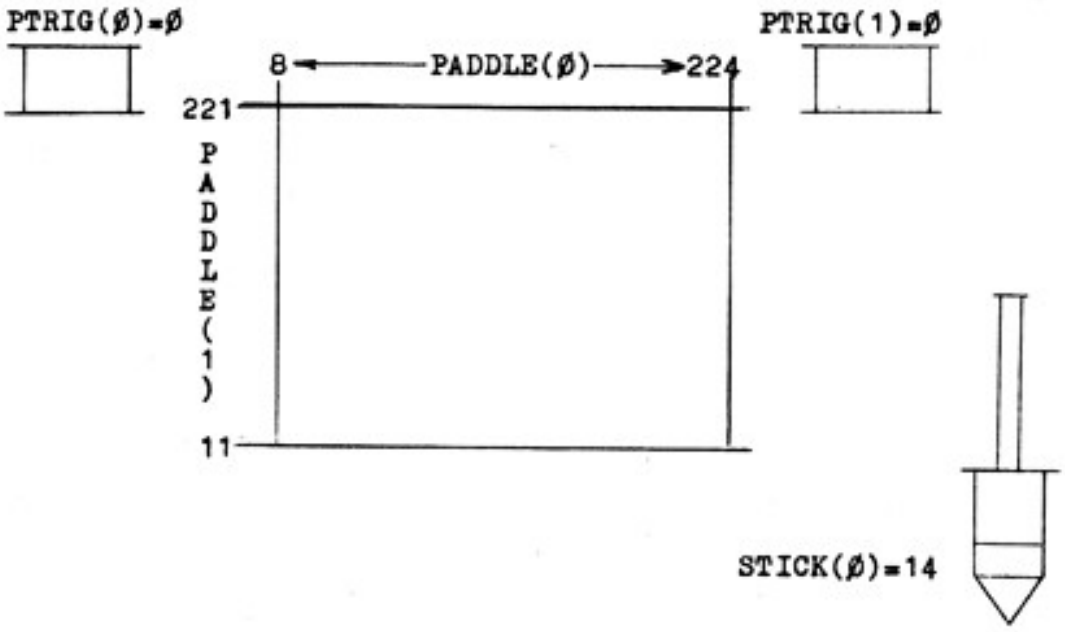
a) Čtení pomocí PADDLE z plochy citlivé na dotyk

X = PADLE (0) čte vzdálenost od levého okraje
Y = PADDLE(1) čte vzdálenost od spodního okraje

Když není dotyk na tabulce, mají X a Y hodnotu 228.

b) Dále jsou k dispozici tři tlačítka, na která se můžeme dotazovat jako na Joystick (nebo Paddle triger)

STICK(0) = 15 nestlačeno žádné tlačítko
= 14 stlačeno tlačítko na psacím hretu
= 11 stlačeno levé tlačítko nebo PTRIG(0)=0
= 7 stlačeno pravé tlačítko nebo PTRIG(1)=0



F u n k c e tabulky

Dva malé příklady programování pomocí tabulky

Zvuková tabulka"

```
1Ø P=PADDLE(Ø): IF P=228 THEN SOUND Ø,Ø,Ø,Ø: GOTO 1Ø
2Ø SOUND Ø,P,1Ø,8: GOTO 1Ø
```

Nastavení barev pomocí tabulky

```
1Ø PØ=PADDLE(Ø): P1=PADDLE(1): IF PØ=228 OR P1=228 THEN 1Ø
2Ø SETCOLOR 2,PØ/15,P1/15: GOTO 1Ø
```

Umístění kurzoru na obrazovce

První a jednoduchou možností je absolutní polohování objektu na obrazovce. K tomu je potřebné určit souřadnice X a Y kurzoru přímo z hodnot Paddle tabulky.

Příklad v BASICu:

```
1Ø     REM xxAbsolutní polohování kurzoru pomocí tab.
3Ø     OPEN #3,4,Ø,"K:"
1ØØ    IF PEEK(764)<>255 THEN GET #3,A:? CHRØ(A);:REMStla-
       čené tlačítko?
11Ø    XP=PADDLE(Ø): YP=PADDLE(1): IF XP=228 or YP=228 THEN 1ØØ
12Ø    GOSUB 1Ø ØØØ
19Ø    GOTO 1ØØ
1ØØØØ  REM xVýpočet polphy kurzoru
1ØØ1Ø  XC=(XP-1Ø)/5: YC=23-(YP-2Ø)/8
1ØØ2Ø  IF YC>23 THEN YC=23: REM Kontrola hranic
1ØØ3Ø  IF YC<Ø THEN YC = Ø
1ØØ4Ø  IF XC>39 THEN XC=39
1ØØ5Ø  IF XC<2  THEN XC=2
1ØØ6Ø  POKE 84,YC:POKE 85,XC:? CHRØ(34);CHRØ(3Ø);:REM vedení kurz.
1ØØ9Ø  RETURN
```

Absolutní polohování s grafickou tabulkou:

Druhá metoda, označená jako relativní polohování, je trochu složitější. Souřadnice se nevypočítávají bezprostředně z polohy pera na tabulce jako v předcházejícím příkladě, ale vztahují se vždy na poslední napsaný bod. Vložte

do počítače dále uvedený příklad a vyzkoušejte si ho. Když nasadíte pero (nebo prst!) na libovolné místo tabulky, zůstane kurzor ještě na původním místě. Až když začnete pohybovat perem, kurzor ho sleduje. Když dosáhnete okraje tabulky, zvednete pero, nasadíte ho znovu až na opačném konci, potom můžete kurzorem pohybovat dále.

```
10      REM xxx Relativní polohování kurzoru pomocí tabulky
20      YRANGE=16:XRANGE=16: REM xxx Citlivost
30      OPEN #3,4,0,"K:"
100     IF PEEK(764)<>255 THEN GET #3,A: ? CHR$(A);
110     XP=PADDLE(0):YP=PADDLE(1): IF XP=228 OR YP=228 THEN 100
120     GOSUB 10000
190     GOTO 100
10000 REM xx Řízení kurzoru
10010 XC=PEEK(85):YC=PEEK(84): XAUP=PADDLE(0):YAUP=PADDLE(1)
10020 XP=PADDLE(0):YP=PADDLE(1):REM xxx nová poloha
10030 IF XP=228 OR YP=228 THEN RETURN:REM xxx zvednuto
10035 XP=(XP+XAUP)/2:YP=(YP+YAUP)/2: REM střední hodnota
10040 XC=XC+(XP-XAUP)/XRANGE: YC=YC-(YP-YAUP)/YRANGE:XAUP=
      YP
10050 IF YC>23 THEN YC=23: REM xxx zkouška hranic
10060 IF YC<0 THEN YC=0
10070 IF XC>39 THEN XC=39
10080 IF XC<2 THEN XC=2
10090 POKE 84,YC:POKE 85,XC:? CHR$(31);CHR$(30);:REM vede-
      ní kurzoru
10100 GOTO 10020
```

Relativní polohování s grafickou tabulkou :

Výhody uvedených metod: Lze zabránit slabému "Třepání" kurzoru z prvního příkladu a kromě toho lze touto metodou editovat grafiku, která překročí rozlišení tabulky. V příkladu pro relativní polohování je dodatečně zahrnuto vytvoření střední hodnoty ze dvou po sobě jdoucích hodnot, což přispívá k uklidnění kurzoru.

Zvukový generátor ve VBI s komfortním editorem

V mnoho programech v BASICu můžete narazit na následující problém. Během zvukového efektu se zastaví pohyb na obrazovce. BASIC je totiž příliš pomalý na to, aby při zajímavých zvukových efektech mohl pohyb pokračovat. Ve strojovém jazyce je to přirozeně možné, i když to není zcela jednoduché, poněvadž nelze použít žádného jednoduchého povelu "SOUND". Se zvukovým generátorem můžete tento nedostatek překonat. Můžete jedním povelem vyvolat v BASICu sled příkazů pro zvukové efekty, které potom běží nezávisle na BASICu. I když programujete ve strojovém jazyce je dále popsaný zvukový systém velmi užitečný, poněvadž tóny mohou být sestaveny pomocí komfortního editoru.

Vlastnosti:

- tóny se vyrábí pomocí editoru
- až 32 tónů je možno uchovat v paměti
- je možno spojovat zvuky
- je možno současně vyvolat více zvuků
- možnost použití v BASICu i assembleru
- počítač je plně k dispozici i během zvukovýc efektů.

Jak pracuje ?

S pomocí programu SOUND-EDITOR můžete sestavit až 32 různých zvuků a ihned je s vysokou věrností poslouchat. Jestliže tyto tóny chcete využít ve vlastním programu, můžete je uložit na disketu a ve svém programu vyvolat podprogramem SNDLOADI.BAS. To umožňuje jednoduchou změnu tónů. Jestliže zjistíte, že některý zvuk se k vašemu programu nehodí, můžete tóny změnit editorem a uložit do paměti. Při příštím průběhu bude váš program již obsahovat nové zvuky.

SOUND-EDITOR, zvukový editor

Na příštích stránkách najdete výpis programu zvukového editoru. Tento program můžete napsat klávesnicí. Pokud vlastníte disketu "HEXENKÜCHE", můžete jej nahrát příkazem RUN"D:SNDEDIT.BAS".
Jestliže patříte k těm, kteří musí program napsat klávesnicí, dávejte zejména pozor na správnost dat na konci programu. V každém případě uložte program do paměti dříve než jej poprve použijete.
Jestliže program hlásí "DATEN-FEHLER" chyba dat, přepsali jste se při psaní programu a doporučujeme vám zkontrolovat vaše data s výpisem programu. Jestliže jste vše udělali správně, musí se zvukový editor přihlásit následujícím menu.

```
SOUNDTEDITOR                                    F.FINZEL 1984

Musternummer  (číslo vzorků)                    : Ø
Freg. Anfang  (počáteční frekvence)             : Ø
Freg. Aenderung  (změna frekvence)              : Ø
Laut. Anfang  (počáteční hlasitost)             : Ø
Laut AEnderung   (změna hlasitosti)             : Ø
Modus          (mod)                            : 5   17 Bit
Laenge         (délka)                          : Ø
Kanál Nr.      ( číslo kanálu)                  : Ø
Prioritaet     (priorita)                       : Ø
Naechsr.Muster (příští vzorek                   : NIL

TRIGGER FUER SOUND   (spouštění zvuku)
START : LOAD FILE    OPTION: SAVE FILE
```

Obsluha zvukového editoru je jednoduchá. Zastrčíte joystick do vstupu číslo 1. Pohybem nahoru nebo dolů volíte řádek, pohybem doleva nebo doprava zvětšujete nebo zmenšujete hodnotu řádku. Vyjímku tvoří pouze řádek modus. Zde volíte pohybem joysticku doleva nebo doprava způsob zkreslení zvuku.
Co tedy můžete všechno nastavit?
Číslo vzorku: Ø - 31
Zde si můžete volit jeden z 32 možných, který má být editován.

Počáteční frekvence : (Ø-255)

Výška tónu, která se nastaví v počátečním okamžiku. Ø - nejvyšší tón, 255 - nejnižší

Změny frekvence : (-127 až + 127)

Jako změna frekvence se označuje hodnota představující změnu frekvence 1/50 s.Záporné hodnoty zvyšují a kladné hodnoty snižují tón Vyzkoušejte si relativně vysoké hodnoty:

Např.:5Ø, -6Ø, dostanete zajímavé bublavé zvuky.

Částečná hlasitost : (Ø-255)

Síla zvuku, keterá se nastaví v počátečním okamžiku. Sílu zvuku lze normálně nastavit v 16-ti stupních (Ø-15). Síla zvuku odpovídá síle zvuku 8 v příkazu SOUND.

Změna hlasitosti : (-127 až +127)

Změna hlasitosti odpovídá změně hlasitosti za 1/50 s. Tak jako v předcházejícím odstavici je tato hodnota již vynásobena 16. Tím se docílí jemnějších rozdílů v hlasitosti. Když nastavíte hodnotu 1, tak se síla zvuku každých 16 taktů (tj. 1/5Ø s) zvýší o jeden stupeň. Také zde jsou zajímavé relativně vysoké hodnoty(např.4Ø)

Mód : Tím můžete zařadit způsob vzniku tonú POKEY. Ten odpovídá zhruba parametru "D i s t o r t i o n" v BASICu. Můžete nastavit tyto možnosti

5 & 17 Bit Poly	: nečisté zvuky
5 Bit POLY	: monotonní zkreslení
5 & 4 Bit Poly	: zvuk podobný motoru
Purer Ton	:čistý ton
4 Bit Poly	: Zkreslený tón

D é l k a : (Ø-255)

Délka tonu vyjádřená v 1/5Ø s.

Čísla kanálu : (Ø-3)

k dispozici jsou 4 tónové generátory

Priorita : (Ø-31)

S prioritou to vypadá následovně. Ačkoli ATARI dává programátorovi k dispozici 4 zvukové generátory, může se stát, že právě obsazený kanál dostane výzvu k začátku svého nového zvuku. To připadá v úvahu hlavně u her s mnoha akcemi. Aby se zvládly takové situace, přiřazuje se každému

zvuku priorita, která rozhoduje o důležitosti zvuku. Jestliže sě hraje zvuk nižší priority (např.Ø) a přitom se požaduje zvuk s vyšší prioritou (např.3), zvuk s nižší prioritou se vypne a začne nový tón. Naopak ale nemůže tón s nižší prioritou přerušit tóm

Příští vzorek : (Ø-31)

Pomocí této volby můžete nastavit velmi složitý sled zvuků, tím, že více tónů zapojíte za sebou. Jestliže nechcete pokračovat dalším zvukem nastavíte NIL. V případě, že nastavíte číslo tónu (Ø-31), potom potom bude tento tón vyvolán po skončení probíhajícího tónu. Druhý tón může dostat také pokyn k vyvolání dalšího tónu. Tímto způsobem je dokonce možné programovat krátké melodie. Další zajímavé využití vyvolá odkaz tónu sám na sebe. To se výborně hodí k editování, protože můžete okamžitě slyšet každou změnu. Takový tón se dá vypnout buďto změnou odkazu na NIL v editoru nebo požadavkem na zvuk s vyšší prioritou v témže kanálu.

Příklad ———— Příklad ——— Příklad ——— Příklad ———

Ponechte číslo vzorku na Ø, nastavte počáteční frekvenci na 3Ø, hlasitost na 6Ø. Volte mód "Purer Tón" a délku 2Ø. Po spuštění slyšíte nyní jednoduchý čistý tón. Experimentujte nyní se změnou frekvence a změnou hlasitosti, budete se divit, jak zajímavé tóny můžete vyloudit ze svého počítače.Když jste vyvolali tón, který chcete uchovat, zvolte pro pokus jiné číslo vzorku..

Ukládání zvukových dat do paměti

Na disk můžete napsat větu o 32 tónech. K tomu je zapotřebí stlačit tlačítko."OPTION" a dále zadat název souboru v standartním souboru-formátu pro ATARI.

D: FILENAME.EXT

Doporučujeme užít extedr.SND, který vás informuje, že se jedná o soubor zvukových dat.

Příklad:

Stlačte OPTION a vložte "D:TEST.SND", stlačte RETURN a zvuková data se zapíší na disk. Jak už bylo řečeno, můžete data kdykoliv zase přečíst a měnit. K tomu je třeba stlačit START a vložit název souboru.

Přehled o programu : SOUND-EDITOR :

řádek		
100	-210	Vyvolání inicializačního programu
200	- 390	Hlavní smyčka programu
1000	- 1190	Tvoří programování obrazovky
2000	- 2090	Volba pole, zmáčknutím knoflíku se vyvolá aktualní zvuk.
3000	- 3090	Program pro změnu polí. R určuje relativní pozici v tabulce zvuků. MN a MX určují minimální případně maximální hodnotu pole, H nejmenší délku kroku při změnách
4000	- 4970	Podprogram pro vytištění jakéhokoliv libovolného pole, větvení probíhá na řádku 4Ø2Ø
5000	- 5090	Tiskne hodnotu všech polí,
6000	- 6090	Tiskne hodnotu všech polí
7000	- 7090	LOAD rutina pro čtení
7500	- čtení	názvu souboru.
8000	- 8710	Soubor pomocných rutin pro změnu polí
9000	- 9030	Pomocná rutina pro mazání tabulky zvuku, tisk hlášení VBI
10000	- 10010	Popis pro POKEY mód
32000	- konec	Zvukový program ve strojovém jazyce

```
 1Ø    REM xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 2Ø    REM x            Zvukový editor                  x
 3Ø    REM x              pro VB/ZVUK                   x
 4Ø    REM xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 5Ø    REM
 1ØØ   POKE +Ø6,144: GRAPHICS 2+16: POSITION Ø,5: ?  6;
       "OKAMZIK PROSIM !"
 11Ø   CONSOL = 53279
 12Ø   DIM DØ(4Ø), FØ(2Ø)
 13Ø   GOSUB 32ØØØ: REM x Strojový program
 14Ø   GOSUB 9ØØØ: REM x Vymazání tabulky zvuků
 15Ø   GOSUB 93ØØ: REM x Zapnutí VBI
 2ØØ   GOSUB 1ØØØ: POKE 559,34: GOSUB 91ØØ
 21Ø   MUSTER=Ø: INDEX=SNDTAB: N=1: C=19: GOSUB 5ØØØ
 3ØØ   REM x Hlavní smyčka programu xxxxxxxxxxxxxxx
 31Ø   GOSUB 2ØØØ: REM x Volba pole
 32Ø   GOSUB 3ØØØ: REM x Provedení změn
 33Ø   GOSUB 4ØØØ: REM x Zobrazení pole
 39Ø   GOTO 3ØØ
 4ØØ   REM xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
1000 REM xxxZobrazení masky obrazovky xxx
1005 GRAPHICS 0: POKE 752,1: SETCOLOR 1,0,8: SETCOLOR 2,11,0
1010 ? "  SOUND - EDITOR       P.FINZEL 1984" REM
1020 ?:?:? " Číslo vzorku        :"
1030 ?:? " Počáteční frekvence       : "
1040 ? " Změna frekvence             :"
1050 ?:? "Počáteční hlasitost        :"
1060 ? "Změna hlasitosti             :"
1070 ?:? "Mod                        :"
1080 ? "Délka                        :"
1090 ? "Číslo kanálu                 :"
1100 ? "Priorita                     :"
1110 ?:? "Příští vzorek              :"
1180 POSITION 2,22: ? "START:LOAD FILE - OPTION:SAVE FILE"
1190 RETURN

2000 REM xxx Výběr pole xxx
2010 ST=STICK(0): POKE77,0: IF ST=13 OR ST=14 THEN GOSUB
     9200
2020 N=N+(ST=13 AND N<10)-(ST=14 AND N>1)
2030 IF STRIG(0)=0 THEN POKE SNDNUM,MUSTER: REM x Vypnutí
     tónu
2040 POKE CONSOL,8: IF PEEK(CONSOL)=6 THEN GOSUB 7000
2050 IF PEEK(CONSOL)=3 THEN GOSUB 6000
2090 RETURN

3000 REM xxx Změna pole xxx
3020 ON N GOTO 3100,3200,3300,3400,3500,3600,3700,3800,
     3900,3950
3090 RETURN

3100 REM x Změna čísla vzorku ? x
3110 MN=0: MX=1: A=MUSTER: GOSUB 8000
3120 IF A<>MUSTER THEN MUSTER=A: INDEX=SNDTAB+MUSTER*8:
     GOSUB 5000
3190 RETURN

3200 REM xxx Počáteční frekvence xxx
3210 R=0: MN=0: MX=255: H=1: GOSUB 8100
3220 RETURN

3300 REM xxx Změna frekvence xxx
3310 R=1: MN=-127: MX=127: H=1: GOSUB 8150: RETURN

3400 REM xxx Počáteční hlasitost xxx
3410 R=2: MN=0: MX=248: H=1: GOSUB 8100
3490 RETURN

3500 REM xxx Změna hlasitosti xxx
3510 R=3: MN=-127: MX=127: H=1: GOSUB 8150
3590 RETURN

3600 REM xxx Zkreslení xxx
3610 R=4:MN=0: MX=192: H=32: GOSUB 8100
3690 RETURN

3700 REM xxx Délka tónu xxx
3710 R=5: MN=0: MX=255: H=1: GOSUB 8100
3790 RETURN

3800 REM xxx Volba čísla kanálu xxx
3810 R=6: GOSUB 8200
3820 P=INT(A/4 *4: A=A-P
```

```
3830 MN=0: MX=3: H=1: GOSUB 8000
3840 A=P+A: GOSUB 8300
3890 RETURN

3900 REM xxx Nastavení priority xxx
3910 R=6: GOSUB 8200
3920 P=INT(A/4) * 4: K=A-P: A=P/4
3930 MN=0: MX=31: H=1: GOSUB 8000
3940 A=A * 4 + K: GOSUB 8300: RETURN

3950 REM xxxOdkaz na příští vzorek xxx
3960 R=7: GOSUB 8250
3970 MN=-1: MX=31: H=+: GOSUB 8000
3980 GOSUB 8350
3990 RETURN

4000 REM xxx Výpis pole na obrazovku xxx
4020 ON N GOTO 4100,4200,4300,4400,4500,4600,4700,4800,
     4900,4950
4090 RETURN

4100 REM xxx Výběr čísla vzorku xxx
4110 H=MUSTER: L=3: GOSUB 8700
4190 RETURN

4200 REM xxx Výběr počáteční frekvence xxx
4210 R=0: L=5: GOSUB 8500: RETURN

4300 REM xxx Výběr změna frekvence xxx
4310 R=1: L=6: GOSUB 8600: RETURN

4400 REM xxx Výběr počáteční hlasitosti xxx
4410 R=2: L=8: GOSUB 8500: RETURN

4500 REM xxx Výběr změny hlasitosti xxx
4520 R=3: L=9: GOSUB 8600: RETURN
4600 REM xxx Výber zkreslení xxx
4620 R=4:L=11:GOSUB 8200: H=A/32: RESTORE 10001+H:
     READ D0: POSITION C,L: ? D0;"        "RETURN

4700 REM xxx Výběr délky tónu xxx
4710 R=5: L=12: GOSUB 8500: RETURN
4790 RETURN

4800 REM xxx Výber čísla kanálu xxx
4810 R=6: L=13: GOSUB 8200: H=A-INT(A/4)*4: GOSUB 8700:
     RETURN

4900 REM xxx Výběr priority xxx
4910 R=6: L=14: GOSUB 8200: H=INT(A/4): GOSUB 8700:
     RETURN
4950 REM xxx Výběr odkazu na příští vzorek xxx
4960 R=7: L=16: GOSUB 8250: IF A=-1 THEN POSITION C,L:
     ? "NIL": RETURN
4970 POSITION C,L: ? A; "      ": RETURN

5000 REM xxx Společný výpis všech hodnot vzorku xxx
5010 GOSUB 4100: GOSUB 4200: GOSUB 4300: GOSUB 4400:
     GOSUB 4500
5020 GOSUB 4600: GOSUB 4700: GOSUB 4800: GOSUB 4900:
     GOSUB 4950
5090 L=3: RETURN
```

```
6000 REM xxx SAVE - Uložení v paměti xxx
6010 D$="SAVE-FNAME (D:FN.EXT)": GOSUB 7500: IF F THEN
     RETURN
6020 TRAP 6050: OPEN #1,8,0,F$
6030 FOR I=0 TO 255: A=PEEK(SNDTAB+I): PUT #1,A: NEXT I
6050 CLOSE #1
6060 TRAP 40000: GOSUB 9100: GOSUB 9300
6090 RETURN

7000 REM xxx LOAD - Čtení xxx
7010 D$="LOAD-FNAME (D:FN.EXT)": GOSUB 7500: IF F THEN 7050
7020 TRAP 7050: OPEN# 1,4,0,F$
7030 FOR I=0 TO 255: GET #1,A: POKE SNDTAB+I,A: NEXT I
7050 CLOSE #1
7060 TRAP 40000: GOSUB 9100: GOSUB 9300
7070 GOSUB 5000
7090 RETURN

7500 REM xxx Vkládání souborů xxx
7510 GOSUB 9150: POSITION 2,20: ? D$;: POKE 764,255
7520 TRAP 7590: INPUT F$: IF F$(+,+)="D" THEN F=0:RETURN
7590 F=1: RETURN

8000 REM x Pomocné rutiny, změna polí,ovládání kurzoru xxx
8005 POSITION C-1,L: ? CHR$(190);
8010 IF ST=11 THEN A=A-H: IF A<MN THEN A=MN
8020 IF ST=7 THEN A=A+H: IF A>MX THEN A=MX
8050 IF ST<>11 AND ST<>7 THEN GOSUB 9200
8080 POSITION C-1,L: ? "   " ;
8090 RETURN

8100 REM xxx Čtení, změna, výpis hodnot (0-255)
8110 GOSUB 8200: GOSUB 8000: GOSUB 8300
8120 RETURN

8150 REM xxx Čtení, změna, výpis hodnot (+/-127)
8160 GOSUB 8250: GOSUB 8000: GOSUB 8350: RETURN
8200 A=PEEK(INDEX+R): RETURN : REM xxx Čtení hodnoty (0-255)
8250 A=PEEK(INDEX+R): A=A-256*(A>127): RETURN: REM
     xxx Čtení +/-127
8300 POKE INDEX+R,A: RETURN: REM xxx Psaní hodnoty 0-255
8350  A=A+256*(A<0): POKE INDEX+R,A: RETURN: REM xxx psa-
     ní +/-127 xxx

8500 REM xxx Výpis hodnoty 0-255
8510 POSITION C,L: ? PEEK(SNDTAB+MUSTER*B+R); "    ":RETURN

8600 REM xxx Výpis hodnoty +/-127 xxx
8610 GOSUB 8250: POSITION C,L: ? A; "     ":RETURN

8700 REM Výpis pomocné proměnné H xxx
8710 POSITION C,L: ? H; "    ": RETURN

9000 REM xxx Vymazání tabulky zvuků xxx
9010 FOR I=0 TO 31
9020 FOR J=0 TO 6: POKE SNDTAB+I*8+J,0: NEXT J: POKE SNDTAB
     +I*8+7,255
9030 NEXT I
9090 RETURN

9100 REM xxx Vytištění textového řádku xxx
```

```
9110 POSITION 2,20: ?"        )))TRIGER FUER SOUND(((     "
     : RETURN

9150 REM xxx Smazání textového řádku xxx
9160 POSITION 2,20: ?"                                    "
     :RETURN

9200 REM xxx Doba čekání xxx
9210 TICK=PEEK(20)
9220 IF PEEK (20)=TICK THEN 9220
9230 RETURN

9300 REM xxx VBI zapnutí xxx
9310 A=USR(SNDEIN)
9320 RETURN
10000 REM xxx POPIS ZKRESLENÍ xxx
10001 DATA 5 & 17 Bit Poly
10002 DATA 5 Bit Poly
10003 DATA 5 & 4  Bit POly
10004 DATA 5 Bit Poly
10005 DATA 17 Bit Poly
10006 DATA  Purer Ton
10007 DATA 4 Bit poly
10010 REM xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

32000 REM xxx Strojový program pro zvuk xxx
32010 S=0: RESTORE 32100
32020 FOR A=39936 TO 40212: READ D:POKE A,D: S=S+D: NEXT A
32030 IF S<>36579 THEN ? "Chyba dat !": STOP
32040 SNDEIN=39936: SNDAUS=39940: SNDTAB=40448: SNDFLG=1790:
      SNDNUM=1791
32090 RETURN
32100 DATA 104,76,222,156,104,76,245,156,32,109,156,32,18,
      156
32110 DATA 76,98,228,169,7,141,244,157,14,254,6,144,6,173,
      244,157
32120 DATA 32,55,156,206,244,157,16,240,173,255,6,201,255,
      240,8,32
32130 DATA55,156,169,255,141,255,6,96,10,10,10,170,189,5,
      158,240
32140 DATA 44,189,6,158,41,3,168,189,6,158,74,74,217,240,
      157,144,28
32150 DATA 153,240,157,138,153,228,157,189,0,158,153,232
      157,189,2
32160 DATA 158,153,236,157,189,5,158,153,224,157,32,195,
      156,96,169
32170 DATA 3,141,244,157,32,123,156,206,244,157,16,248,96,
      172, 244
32180 DATA 157,185,224,157,240,63,56,233,1,153,224,157,208,
      29,190
32190 DATA 228,157,189,7,158,201,255,240,4,32,55, 156,96,
      169,0,153
32200 DATA 240,157,153,232,157,153,236,157,32,195,156,96,
      190,228,157
32210 DATA 189,1,158,24,121,232,157,153,232,157,189,3,158,
      24,121,236
32220 DATA 157,153,236,157,32,195,156,96,185,232,157,72,185,
      236,157
```

```
32230 DATA 74,74,74,74,29,4,158,72,152,10,168,104,153,1,
      210,104,153
32240 DATA 0,210,96,169,0,141,254,6,169,255,141,155,6,32,
      2,157,162
32250 DATA 156,160,8,169,7,32,92,228,96,162,228,160,98,
      169,7,32,92
32260 DATA 228,32,2,157,96,169,0,162,7,157,0,210,202,16,
      250,141,8
32270 DATA 210,169,3,141,15,210,96
```

Použití zvuků v BASICu

Nyní přijde to nejdůležitější. Jakým způsobem můžete zvuky připravené editorem zařadit do svého programu? Není nic snadnějšího. Potřebujete ke svému programu připojit program SOUNDLOADER -"SNDLOAD1.BAS", který najdete dále. Tento program vyřídí všechny úkoly, které souvisejí se zvukovými programy, jako rezervace paměti, "Poken" strojového programu, čtení dat vyrobených pomocí zvukového editoru, aktivace rutiny VBI.

Soundloader

Na příštích stránkách můžete najít výpis programu v BASICU sloužící pro zavedení zvukového strojového programu SNDLOAD1.BAS. Pro ty, kteří chtějí tento program napsat přes klávesnici, je důležité vědět,blok dat v SNDLOAD1.BAS je zcela identický s blokem dat v programu Sound-Editor. V obou případech se využívá stejný strojový program. Jestliže jste již editor napsali a to je nutný předpoklad pro použití programu Sound-loader, můžete si práci výrazně zjednodušit. Nahrajete Soundeditor,pomocí příkazu LIST "D:S.LST",32000,32999 uložíte na prázdnou disketu či kazetu příkazem LIST "C:S.LS",32000,32999. Nyní zbývá již dopsat zbývající řádky. Vlastníci diskety HEXENKUCHE jsou ve výhodě, poněvadž BASIC-loader je na disketě obsažen jako SNDLOAD1.BAS.

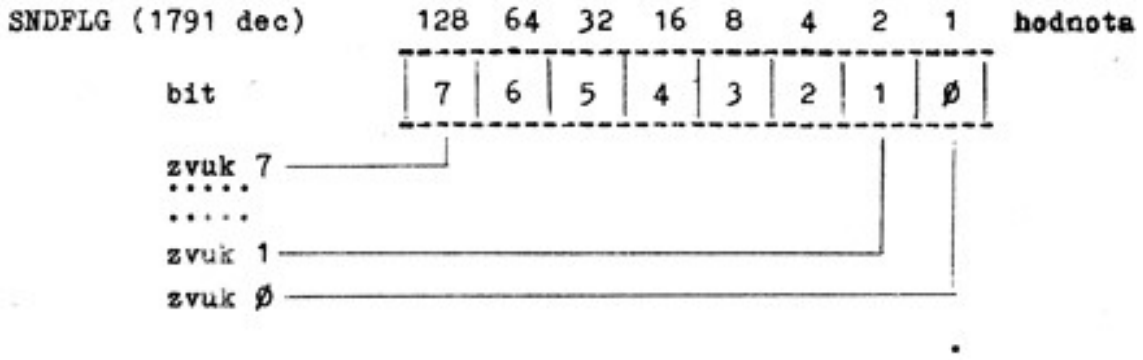
Užití programu Soundloader

Především napište na řádek 32530 název souboru, který jste pomocí zvukového editoru vytvořili. Např. název souboru z předcházející kapitoly, který jsme pro ukázku vytvořili pod názvem souboru TEST.SND. Vlastníci diskety HK nevkládají

nic, na disketě je demonstrační program pod názvem DRUMS. SND. Nyní spusťte program povelem RUN a čekejte na ukončení inicializace. Nyní obdržíte hlášení, jakým způsobem můžete SOUND-VBI opět vypnout (A=USR(SNDAUS)).

A už to běží

Máte dvě možnosti vyvolání zvuku. První a jednoduší možností je napsat číslo zvuku do paměťové buňky SNDNUM(179Ø dec). Můžete použít čísla od 1 do 31. Hned si to zkuste.
Napište POKE SNDNUM,Ø
Již slyšíte zvuk, který byl napsán v Editoru pod číslem Ø. Zvuk slyšíte přirozeně jen v tom případě, jestliže byl v editoru skutečně napsán.
Druhý způsob je poněkud komplikovanější, ale má jednu významnou výhodu. Můžete totiž vyvolat současně více zvuků. Váš počítač ATARI má 4 zvukové kanály a měli byste je plně využívat. Výběr zvuků se provádí pomocí paměťové buňky SNDFLG(1791), jejíž jednotlivé bity odpovídají zvukům Ø až 7.



Výběr zvuků přes SNDFLG

Jestliže chcete vyvolat jen jeden zvuk, např. Ø, zadejte
POKE SNDFLG,1

Jestliže chcete současně vyvolat více zvuků, musíte jejich hodnoty sečíst a hodnotu součtu zapsat do SNDFLG. Pro zvuk Ø hodnota (1) a zvuk 2 hodnota (4), to znamená
POKE SNDFLG,5

Tento postup je možný tehdy, když zvuky, které mají znít současně, mají přiřazeny různé kanály. Možnosti tohoto

způsobu vyvolání zvuku jsou velké, můžete současně vyvolat až 4 melodie. Příklad využití těchto možností najdete v kapitole "ATARI jako bubeník". Nevýhodou tohoto způsobu vyvolání zvuků je, že takto lze vyvolat pouze prvních 8 zvuků. Konkrétně to znamená, že nejčastěji používaným zvukům a zvukům, které mají znít současně, přiřazujeme při editování čísla Ø až 7.

```
1ØØ    REM xxx Zde může být váš program
11Ø    REM xxx
12Ø    REM xxx Do řádku 3253Ø vložte název zvuk.souboru
2ØØ    GOSUB 31ØØØ: REM xxx zapnutí zvuku
21Ø    GRAPHICS Ø: ?: ? "Nyní můžete vyvolat zvuky vytvoře
       né editorem "
22Ø    ? "pomocí POKE SNDFLG,<bit vzorku>"
23Ø    ? "nebo  POKE SNDNUM,<číslo zvuku>"
24Ø    ? "Vypnutí zvuku pomocí rutiny A=USR(SNDAUS)"
29Ø    END
31ØØØ REM xxx Basic-loader pro zvukový program
31Ø1Ø POKE 1Ø6,144: GR.Ø: POSITION 1Ø,9: ? "Inicializace"
31Ø2Ø GOSUB 32ØØØ
31Ø3Ø GOSUB 325ØØ: REM Nahrání tabulky zvuků
31Ø4Ø A=USR(SNDEIN): REM xxx Vypnutí zvuku
31Ø9Ø RETURN
32ØØØ REM xxx Zvukový strojový program xxx
32Ø1Ø S=Ø: RESTORE 32100
32Ø2Ø FOR A=39936 TO 42Ø12: READ D:POKE A,D: S=S+D;NEXT A
32Ø3Ø IF S   36579 THEN ? "Chyba DAT!": STOP
32Ø4Ø SNDEIN=39936: SNDAUS=3994Ø:SNDTAB=4Ø448: SNDFLG=179Ø:
       SNDNUM=1791
32Ø9Ø RETURN
321ØØ DATA 1Ø4,76,222,156,1Ø4,76,245,156,216,32,1Ø9,156,
       32,18,156
3211Ø DATA 76,98,228,169,7,141,244,157,14,254,6,144,6,173,
       244,157
3212Ø DATA 32,55,156,2Ø6,244,157,16,24Ø,173,255,6,2Ø1,255,
       24Ø,8,32
3213Ø DATA 55,156,169,255,141,255,6,96,1Ø,1Ø,1Ø,17Ø,189,
       5,158,24Ø
3214Ø DATA 44,189,6,158,41,3,168,189,6,158,74,74,217,24Ø,
       157,144,28
3215Ø DATA 153,24Ø,157,138,153,228,157,189,Ø,158,153,232,
       157,189,2
3216Ø DATA 158,153,236,157,189,5,158,153,224,157,32,195,
       156,96,169
3217Ø DATA 3,141,244,157,32,123,156,2Ø6,244,157,16,248,96,
       172,244
3218Ø DATA 157,185,224,157,24Ø,63,56,233,1,153,224,157,2Ø8,
       29,19Ø
3219Ø DATA 228,157,189,7,158,2Ø1,255,24Ø,4,32,55,156,96,
       169,Ø,153
322ØØ DATA 24Ø,157,153,232,157,153,236,157,32,195,156,96,
       19Ø,228,157
```

```
32210 DATA 189,1,158,24,121,232,157,153,232,157,
          189,3,158,24,121,236
32220 DATA 157,153,236,157,32,195,156,96,185,232,157,72,
      185,236,157
32230 DATA 74,74,74,74,29,4,158,72,152,10,168,104,153,1,
      210,104,153
32240 DATA 0,210,96,169,0,141,254,6,169,255,141,255,6,32,
      2,157,162
32250 DATA 156,160,8,169,7,32,92,228,96,162,228,160,98,169,
      7,32,92
32260 DATA 228,32,2,157,96,169,0,162,7,157,0,210,202,16,
      250,141,8
32270 DATA 210,169,3,141,15,210,96
32500 REM xxx BLOAD xxx Inicializace stroj.programu
32510 BLOAD=1536: RESTORE 32600
32520 FOR A=BLOAD TO BLOAD+33: READ X: POKE A,X: NEXT A
32530 OPEN#3,4,0 "D:DRUMS.SND": REM  xxx Zde nahrát zvu-
      kové soubory
32540 X=USR(BLOAD,SNDTAB,256):REM xxx Nahrát soubory
32550 CLOSE#3:RETURN
32600 DATA 104,162,48,169,7,157,66,3,104,157,69,3,104,157,
      68,3,104
32610 DATA 157,73,3,104,157,72,3,32,86,228,132,212,169,0,
      133,231,96
```

Program pro zvuk ve strojovém kódu

Pro pochopení strojového programu pro zvuk je důležitá struktura, se kterou jsou jednotlivé zvuky ukládány do pa - měti

1. byte : Počáteční frekvence
2. byte : Změna frekvence za 1/50 s
3. byte : Počáteční hlasitost * 16
4. byte : Změna hlasitosti za 1/50 s * 16
5. byte : Délka zvuku v 1/50 s
6. byte : Zkreslení * 16
7. byte : Bity 0-1: číslo kanálu, bity 2-7:priorita
8. byte : Odkaz na následující vzorek

Dohromady je k dispozici 32 takových 8-bytových bloků, které od SNDTAB($9E00) zabírají za sebou 256 byte.

Zvuková tabulka není pevně spojena s adresami, může být, jako v případě "BASIC-loaderu", uložena najiné adrese. Přirozeně se s tím musí počítat ve strojovém programu. Konstanta SNDTAB musí směřovat na počátek tabulky. V takovém případě nelze používat SNDLOAD1.BAS, ale je třeba pracovat přímo se soubory v assembleru. Zvuková tabulka tvoří datovou základnu strojového programu. Z této tabulky se kopírují požadovaná data do interních paměťových m míst VBI programu a počítají se aktuální hodnoty zvuku.

Vykonávání skoků

Strojový program začíná skoky tvořenými z příkazů PLA a JMP. To má významnou přednost v tom, že se nemusíme starat o interní programové adresy. Vyvolávací adresa je v tomto případě vždy rovna fyzikální počáteční adrese programu. Druhá adresa skoku je také pevně stanovena, je rovna počáteční adrese + 4. Příkazy PLA slouží k opravám po vyvolání strojového programu prostřednictvím BASICu. Jestliže chcete zvukový program vyvolat ze strojového programu musíte přeskočit PLA. V tomto případě bude váš program vypadat takto:

```
SNDADR = $9000                    ; zde je zvukový program
JSR SNDADR+1                      ; zvuk zapnut
JSR SNDADR+5                      ; zvuk vypnout
```

Inicializace

Skoky vedou při zapnutém strojovém programu k podprogramu SNDINI. POKEY je rezervován (v SNDPOK), registry SNDNUM a SNDFLG jsou vymazány a zvukový VBI program se aktivizuje (viz kapitola programování VBI). Povel v BASICU k zapnutí zvukového programu zní : X=USR(39936).

Program přerušení

Začíná u značky SNDVBI a dělí se na dvě významné části. Zvukový interpretr (SNINTP), který zpracovává momentálně znějící zvuk a test nových požadavků (SNDANF), který testuje předávající paměťové buňky SNDNUM a SNDFLG. Zůstaňme u druhé části. Podprogram SNDANF začíná zkouškou všech 8 bitů SNDFLG. Jestliže je objevena jedna jednička, nastane UP - vyvolání rutiny pro přípravu zvuku (SNVORB). V souvislosti s tím se prohlídne, jestli do SNDNUM přišel požadavek. Jestliže ano, je opět použita přípravná rutina. V UP SNVORB se zkouší, zda délka nového tónu je nenulová a zda jeho priorita je větší nebo stejná než priorita probíhajícího zvuku v požadovaném kanále. Jestliže jsou obě podmínky splněny, přenese se počáteční frekvence, počáteční

hlasitost a délka tónu do proměnných zvukového kanálu. Zvukový interpretr (SNINTP) se skládá především z kmenového programu, který obsahuje jenom jednu smyčku pro obsluhu všech kanálů a z interpretačních rutin pro jeden kanál, která se vyvolá 4-krát. Tam se sleduje jestli je kanál momentálně aktivní. Jestliže ano, sníží se hodnota délky zvuku, přepočítá se s pomocí tabulky zvuků frekvence a hlasitost. Jestliže se zjistí, že zvuk končí, prověří se, zda následuje nový zvuk a eventuelně se nový zvuk připraví.

Vypnutí

Postup při vypnutí je podobný postupu při zapnutí. Skoky vedou k UP SNDEXT, tím, že VBI je uveden do původního stavu a POKEY je ještě jednou inicializován, aby byly ukončeny probíhající tóny. Z BASICu se vypnutí řídí příkazem X=USR(SNDAUS), přičemž SNDAUS=3994Ø.

```
                Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø11Ø ;x       ZVUK VE VBI              x
                Ø13Ø ;xZvuk.gener. v VBI - k edit.SNDEDIT
                Ø149 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø15Ø ;
                Ø16Ø ; Vzorová tabulka:
                Ø17Ø ;
=9EØØ           Ø18Ø SNDTAB =    9EØØ   Vzor.tab.je zde odložena
                Ø19Ø ;
                Ø2ØØ ; Struktura vzor.tab.:opakuje se 32-krát
                Ø21Ø ;
                Ø22Ø ; 1.byte: počáteční frekvence
                Ø23Ø ; 2.byte: změna frekvence
                Ø24Ø ; 3.byte: počáteční hlasitost * 16
                Ø25Ø ; 4.byte: změna hlasitosti * 16
                Ø26Ø ; 5.byte: trvání zvuku v 1/5Ø s
                Ø27Ø ; 6.byte: zkreslení * 16
                Ø28Ø ; 7.byte: bity Ø-1:číslo kanálu,bity
                               2-7: priorita
                Ø29Ø ; 8.byte: odkaz na příští vzorek
               Ø3ØØ  ;
                Ø31Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø32Ø ;OS a hardwarové adresy
                Ø33Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø34Ø ;
=E45C           Ø35Ø SETVBV =   $E45C   O.S. vektory
=E462           Ø36Ø XITVBV =   $E462
                Ø37Ø ;
=D2ØØ           Ø38Ø AUDF1 =    $D2ØØ   registr POKEY
```

```
=D2Ø1          Ø39Ø AUDC1  =   $D2Ø1
=D2Ø8          Ø4ØØ AUDCTL =   $D2Ø8
=D2Ø7          Ø41Ø SKCTL  =   $D2ØF
               Ø42Ø ;
               Ø43Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø44Ø ; Adresy řezů
               Ø45Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø46Ø ;
=Ø6FE          Ø47Ø SNDFLG =   $Ø6FE    vyvolání zvuků Ø-7
                                         přes flag
=Ø6FF          Ø48Ø SNDNUM =   $Ø6FF    vyvolání zvuku Ø-31
                                         přes číslo
               Ø49Ø ;
               Ø5ØØ ; Vnitřní adresy pro zvuk
               Ø51Ø ;
=9DEØ          Ø52Ø SNDATA =     9DEØ   počat.adresa proměn.
               Ø53Ø ;
=9DEØ          Ø54Ø SNDCNT =     SNDATA     čítač pro délku zvuku
                                             (4 byte)
=9DE4          Ø55Ø SNINDX =     SNDATA+4   index v tab.vzorků(4)
=9DE8          Ø56Ø SNFREQ =     SNDATA+8   aktuální frek.(4)
=9DEC          Ø57Ø SNDVOL =     SNDATA+12  aktuální hlasit.(4)
=9DFØ          Ø58Ø SNDPRI =     SNDATA+16  priorita zvuku(4)
=9DF4          Ø59Ø SNDCHN =     SNDATA+2Ø  pomoc.registr(+ byte)
               Ø6ØØ ;
               Ø61Ø ;
               Ø63Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø64Ø ;Adresy skoků pro vyvolání z BASICu,
                     assembleru
               Ø65Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ØØØØ           Ø66Ø          *= $9CØØ
               Ø67Ø ;
9CØØ  68       Ø68Ø          PLA          pro BASIC
9CØ1 4CDE9C    Ø69Ø          JMP SNDINI     zapnutí SOUND-VBI
9CØ4 68        Ø7ØØ          PLA            inicializace pro BASIC
9CØ5 4CF59C    Ø71Ø          JMP SNDEXT     vypnutí VBI
               Ø72Ø ;
               Ø73Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø74Ø ;rutina VBI,běží každou 1/5Ø s
               Ø75Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø76Ø ;
9CØ8 D8        Ø77Ø SNDVBI CLD              dekadický mod
9CØ9 2Ø6D9C    Ø78Ø          JSR SNINTP     interpretr pro zně-
                                             jící ton
9CØC 2Ø129C    Ø79Ø          JSR SNDANF     test nových požadavků
9CØF 4C62E4    Ø8ØØ          JMP XITVBV     ukončení VBI
               Ø81Ø ;
               Ø82Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø83Ø ; Test požadavků zvuku
               Ø84Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               Ø85Ø ;
9C12 A9Ø7      Ø86Ø SNDANF LDA #7         test požadavků přes flag
9C14 8DF49D    Ø87Ø        STA SNDCHN     čítač pro 8 bitů
               Ø88Ø ;
9C17 ØEFEØ6    Ø89Ø SNANF1 ASL SNDFLG     je-li flag Cary
9C1A 9ØØ6      Ø9ØØ        BCC SNANF2     roven nule,žádný poža-
               Ø91Ø ; dávek --> jinak     připr.zvuk
9C1C ADF49D    Ø92Ø        LDA SNDCHN     číslo vzorku (0-7)
9C1F 2Ø379C    Ø93Ø        JSR SNVORB     zvuk zmáčknout>>>
               Ø94Ø ;
```

```
9C22 CEF49D Ø95Ø SNANF2 DEC SNDCHN    již všechny flagy(8)
9C25 1ØFØ   Ø96Ø        BPL SNANF1    otestovány? ne,dále -->
            Ø97Ø ;
9C27 ADFFØ6 Ø98Ø        LDA SNDNUM    požad.přes čís.vzorku
9C2A C9FF   Ø99Ø        CMP #255      kod pro žádný požadavek
9C2C FØØ8   1ØØØ        BEQ SNAEND    ano, konec -->
            1Ø1Ø ;
9C2E 2Ø379C 1Ø2Ø        JSR SNVORB    jinak připravit
9C31 A9FF   1Ø3Ø        LDA #255      a nastavit zpět flag
9C33 8DFFØ6 1Ø4Ø        STA SNDNUM
            1Ø5Ø ;
9C36 6Ø     1Ø6Ø SNAEND RTS
            1Ø7Ø ;
            1Ø8Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            1Ø9Ø ;Příprava zvuků, přenést parametry z tab.
            11ØØ ;<A> : číslo vzorku
            111Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            112Ø ;
9C37 ØA     113Ø SNVORB ASL A         index v tab.vzorků
9C38 ØA     114Ø        ASL A         = číslo vzorku * 8
9C39 ØA     115Ø        ASL A
9C3A AA     116Ø        TAX           index vzorku v X
9C3B BDØ59E 117Ø        LDA SNDTAB+5,X délka zvuku
9C3E FØ2C   118Ø        BEQ SNVEND    potom žádný tón
            119Ø ;
9C4Ø BDØ69E 12ØØ        LDA SNDTAB+6,X  obstarat č.kanálu,
9C43 29Ø3   121Ø        AND #3        je obsaženo v bitech Ø-1
9C45 A8     122Ø        TAY           č.kanálu do Y
9C46 BDØ69E 123Ø        LDA SNDTAB+6,X nejdříve zkontr.prioritu
9C49 4A     124Ø        LSR A         priorita je obsaž.v bitech
                                      2-7
9C4A 4A     125Ø        LSR A
9C4B D9FØ9D 126Ø        CMP SNDPRI,Y  větší moment. priorita
9C4E 9Ø1C   127Ø        BCC SNVEND    ano.požadavek se ignoruje
            128Ø ;
9C5Ø 99FØ9D 129Ø        STA SNDPRI,Y  sledovat prioritu
9C53 8A     13ØØ        TXA           index v tab.vzorků
9C54 99E49D 131Ø        STA SNINDX,Y  uložit pro interpretr
9C57 BDØØ9E 132Ø        LDA SNDTAB+Ø,X počáteční frekvence
9C5A 99E89D 133Ø        STA SNFREQ,Y  do reg.pro moment.prioritu
9C5D BDØ29E 134Ø        LDA SNDTAB+2,X počáteč.hlasitost
9C6Ø 99EC9D 135Ø        STA SNDVOL,Y  reg.pro aktual.hlasitost
9C63 BDØ59E 136Ø        LDA SNDTAB+5,X délku zvuku
9C66 99EØ9D 137Ø        STA SNDCNT,Y  zapsat do počítače
9C69 2ØC39C 138Ø        JSR SNDHRD    zvuk ihned spustí>>>
9C6C 6Ø     139Ø SNVEND RTS
            14ØØ ;
            141Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            142Ø ;Zvukový interpretr pro všechny kanály
            143Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            144Ø ;
9C6D A9Ø3   145Ø SNINTP LDA #3       zpracování všech kanálů
9C6F 8DF49D 146Ø        STA SNDCHN    odloženo v čítači
            147Ø ;
9C72 2Ø7B9C 148Ø SNNXCH JSR SNINTC    zprac. jedntl. kanálů
9C75 CEF49D 149Ø        DEC SNDCHN    již všechny 4 kanály?
9C78 1ØF8   15ØØ        BPL SNNXCH    ne,další kanál ---
9C7A 6Ø     151Ø        RTS
            152Ø ;
```

```
                1530 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                1540 ; zpracování kanálu,výpočet nové frekv. a
                1550 ; hlasit.  SNDCHN  :čís. kanálu
                1560 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                1570 ;
9C7B ACF49D     1580 SNINTC LDY SNDCHN    aktual.čís.kanálu
9C7E B9E09D     1590        LDA SNDCNT,Y  čítač kanálu pro délku
                                          zvuku.
9C81 F03F       1600        BEQ SNIEND    kanál není aktivní -->
                1610 ;
9C83 38         1620        SEC           jinak snížit délku
9C84 E901       1630        SBC #1        čítač
9C86 99E09D     1640        STA SNDCNT,Y  čítač vynulován ?
9C89 D01D       1650        BNE SNINOK    ne,potom norm.zpracování
                1660 ;
9C8B BEE49D     1670        LDX SNINDX,Y  jinak zkouška zřetězení
                                          zvuků
9C8E BD079E     1680        LDA SNDTAB+7,X příští vzorek
9C91 C9FF       1690        CMP #255      žádný další vzorek?
9C93 F004       1700        BEQ SNDAUS    ano,kanál vypnout! -->
                1710 ;
9C95 20379C     1720        JSR SNVORB    jinak připr.zvuk>>>
9C98 60         1730        RTS
                1740 ;
9C99 A900       1750 SNDAUS LDA #0        kanál vypnout
9C9B 99F09D     1760        STA SNDPRI,Y  priorita:=0
9C9E 99E89D     1770        STA SNFREQ,Y  frekvence:=0
9CA1 99EC9D     1780        STA SNDVOL,Y  hlasitost:=0
9CA4 20C39C     1790        JSR SNDHRD
9CA7 60         1800        RTS
                1810 ;
9CA8 BEE49D     1820 SNINOK LDX SNINDX,Y  index vzorku
9CAB BD019E     1830        LDA SNDTAB+1,X změna frek.dle tab.
9CAE 18         1840        CLC
9CAF 79E89D     1850        ADC SNFREQ,Y  výpoč.nové frekvence
9CB2 99E89D     1860        STA SNFREQ,Y  a uchování v paměti
9CB5 BD039E     1870        LDA SNDTAB+3,X změnu hlasitosti dle tab.
9CB8 18         1880        CLC
9CB9 79EC9D     1890        ADC SNDVOL,Y  připoč.k současn.hlasitosti
9CBC 99EC9D     1900        STA SNDVOL,Y  novou hlasitost
9CBF 20C39C     1910        JSR SNDHRD    zapsat do hardware>>>
9CC2 60         1920 SNIEND RTS
                1930 ;
                1940 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                1950 ;Hlasitost a frekvenci zapsat do POKEY
                1960 ;<Y>:čís.kanálu,<X>:čís.vzorku * 8
                1970 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                1980 ;
9CC3 B9E89D     1990 SNDHRD LDA SNFREQ,Y  aktuální výška tónu
9CC6 48         2000        PHA           sledovat
9CC7 B9EC9D     2010        LDA SNDVOL,Y  aktuální hlasitost
9CCA 4A         2020        LSR A         dělit 16-ti
9CCB 4A         2030        LSR A
9CCC 4A         2040        LSR A
9CCD 4A         2050        LSR A
9CCE 1D049E     2060        ORA SNDTAB+4,X k tomu bity zkreslení
9CD1 48         2070        PHA           a také sledovat
9CD2 98         2080        TYA           index pro hardw.registr
9CD3 0A         2090        ASL A         =<Y> *2
9CD4 A8         2100        TAY
```

```
9CD5 68       2110        PLA            hlasitost a zkreslení
9CD6 99Ø1D2   2120        STA AUDC1,Y    kontr.reg.kanálu
9CD9 68       2130        PLA            výška tonu
9CDA 99ØØD2   2140        STA AUDF1,Y    frekv.registr kanálu
9CDD 6Ø       2150        RTS
              2160 ;
              2170 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2180 ; Inicializační rutina
              2190 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2200 ;
9CDE A9ØØ     2210 SNDINI LDA #Ø         vypnout všechny kanály
9CEØ 8DFEØ6   2220        STA SNDFLG     vymazat flagy
9CE3 A9FF     2230        LDA #255
9CE5 8DFFØ6   2240        STA SNDNUM     snížit čis.inicializace
9CE8 2ØØ29D   2250        JSR SNDPOK     inicializace POKEY
9CEB A29C     2270        LDX #SNDVBI/256 HI byte rutiny VBI
9CED AØØ8     2280        LDY #SNDVBI&255 LO byte
9CEF A9Ø7     2290        LDA #7
9CF1 2Ø5CE4   2300        JSR SETVBV     vložit do VBI >>>
9CF4 6Ø       2310        RTS
              2320 ;
              2330 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2340 ;Vypnout zvuk VBI
              2350 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2360 ;
9CF5 A2E4     2370 SNDEXT LDX #XITVBV/256 vektor restaurováhí VBI
9CF7 AØ62     2380        LDY #XITVBV &255 LO byte
9CF9 A9Ø7 2390            LDA #7
9CFB 2Ø5CE4   2400        JSR SETVBV     vypnutí >>>
9CFE 2ØØ29D   2410        JSR SNDPOK     všechny kanály v klidu
9DØ1 6Ø       2420        RTS
              2430 ;
              2440 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2450 ; POKEY rezervace
              2460 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2470 ;
9DØ2 A9ØØ     2480 SNDPOK LDA #Ø         všechny kanály vypnout
9DØ4 A2Ø7     2490        LDX #7         celkem 8 registrů
              2500 ;
9DØ6 9DØØD2   2510 SNDP 1 STA AUDF1,X    AUDF/C 1/2
9DØ9 CA       2520        DEX            již všechny?
9DØA 1ØFA     2530        BPL SNDP1      ne dále -->
              2540
9DØC 8DØ8D2   2550        STA AUDCTL     zvukový kontrolníreg. zpět
9DØF A9Ø3     2560        LDA #3         seriový CTL vypnout
9D11 8DØFD2   2570        STA SKCTL
9D14 6Ø       2580        RTS
```

A T A R I j a k o b u b e n í k

Jestliže vám chybí ten pravý rytmus, použijete jednoduše váš počítač ATARI jako bicí jednotku. V následujícím programu jsou naprogramovány různé známé rytmy a s troškou šikovností můžete vytvářet vlastní rytmy. Jestliže vám dosud chyběla bicí jednotka, nyní ji máte: - váš počítač. A ještě něco. Poněvadž počítač je o něco inteligentnější než bicí jednotka, můžete hrát komplikované melodie.

Co je zapotřebí ?

Program využívá zvukovou rutinu VBI, která byla vysvětlena v předcházejících kapitolách.

POZOR ! K tomuto programu potřebujete bezpodmínečně dříve popsaný zvukový editor SNEDIT.BAS. Pomocí tohoto editoru budou tvořeny zvuky jednotlivých bubnů a činelů.

V programu pro bicí jednotku se využívá program Soundloader SNDLOAD1.BAS popsaný v předcházející kapitole. Jestliže už máte tento program nahraný, použijete jej jako základu. Řádky 1ØØ až 25Ø, které jsou pouze demonstrační, musíte vymazat. V řádku 3253Ø musíte ještě změnit název souboru na DRUMS.SND. Poslední datový soubor, který obsahuje zvuky jednotlivých nástrojů, musíte vytvořit pomocí zvukového editoru (SNDEDIT.BAS). K tomuto účelu potřebujete následující tabulku pro tvorbu tónů:

Vzorek	Ø	1	2	3	4	5
Poč.frekv. Změna frek.	1 Ø	18 Ø	24Ø Ø	5 Ø	2 Ø	1 Ø
Poč.hlasit. Změna hlas.	96 -13	224 -34	224 -2Ø	138 -13	224 -7	64 8
Mód Délka Kanál Priorita	17 bit 7 Ø Ø	5 17 bit 7 1 Ø	4 bit 12 2 Ø	4 bit 11 Ø Ø	17 bit 32 3 Ø	5 17 bit 17 Ø Ø
Příští vzor. Nástroj Nastav.hod.	NIL Hi Hat 1	NIL Snare 2	NIL Bass 4	NIL Ride 8	NIL Crash 16	NIL HiHAT2 32

Jestliže máte vše nastaveno, stlačte klávesu OPTION a nahrajte soubor D:DRUMS.SND. Na disketě HK najdete oba soubory. Program pro bicí jednotku je k dispozici pod názvem DRUMKIT. BAS.

Programovatelná jednotka bicí :

Vyvolání jednotlivých souborů nástrojů se děje přes proměnné SNDFLG. Program musí dovolit současný podnět pro více zvuků. Je často třeba nechat současně znít více bubnů nebo činel. Rytmus se jednoduše vyrobí tím, že se do SNDFLG periodicky zapíše nová hodnota. Tyto hodnoty jsou uloženy v datových řádcích (od řádku 1ØØØ) a můžete je volně měnit podle svého přání. Hodnotu dostanete sečtením jednotlivých nástrojů.

Hodnota nástrojů: 1-HiHat(charleston)
2-Snaredrum(pochodový buben)
4-Bassdrum(hluboký buben)
8-Ride-Cymbal(činely pro rytmus)
16-Crash-Cymbal(činely)
32-HiHat Spezialeffekt(spec.efekty)

Současné znění např. HiHat a Basdrum vyžaduje zadat hodnotu 1+4=5. Dále HiHat a Snaredrum dáváhodnotu 3. S těmito dvěma hodnotami, můžete již vyzvořit jednoduchý rytmus. Od řádku 7Ø1Ø je rezervováno místo pro vaše vlastní údaje. Napište např.:

```
7Ø1Ø DATA 5,Ø,1,Ø,3,Ø,1,Ø,-1
```

Spustte program znovu a zvolte nyní hodnotu 7. Uslišíte relativně jednoduchý rytmus, který můžete kdykoliv přirozeně zrušit. Nuly jsou přitom prázdné kroky, které můžete využít pro přerušení zvuku a podobné efekty. Konec bloku tvoří "-1", která zařídí, že blok začne opět od začátku.

Hodně zábavy s bubny !

```
1ØØ  REM xxx ATARI jako bubeník
11Ø  REM xxx využívá VBI rutiny pro zvuk
12Ø  REM xxx potřebuje soubor DRUMS.SND na D1:
13Ø  GOSUB 31ØØØ
15Ø  GOSUB 1ØØØØ: REM Napsání titulu
16Ø  GOSUB 11ØØØ: REM Výběr
17Ø  GOSUB 2ØØ  : REM Hra
19Ø  GOTO 15Ø
2ØØ  REM xxx Zde se bubnuje
21Ø  RESTORE RHYT
22Ø  READ A: IF PEEK(764)<>255 THEN RETURN
23Ø  IF  A<Ø THEN 2ØØ
```

```
24Ø  POKE SNDFLG,A: FOR T=1 TO DALAY: NEXT T
25Ø  ST=STICK(Ø): DELAY=DELAY+(ST=7 AND DELAY<1ØØ)-(ST=
     11 AND DELAY>25)
29Ø  GOTO 22Ø
1ØØØ REM xxx ROCK'n Roll xxx
1Ø1Ø DATA 5,Ø,1,Ø,3,Ø,1,Ø
1Ø2Ø DATA 5,Ø,1,Ø,3,Ø,1,2
1Ø3Ø DATA 5,Ø,1,Ø,3,Ø,1,1
1Ø4Ø DATA 5,3,5,3,3,3,3,3
1Ø5Ø DATA 2Ø,Ø,1,Ø,3,Ø,1,Ø
1Ø6Ø DATA 5,Ø,1,Ø,3,Ø,1,2
1Ø7Ø DATA 5,2,5,Ø,3,Ø,36,Ø
1Ø8Ø DATA 5,Ø,36,Ø,7,Ø,1,1,-1
2ØØØ REM xxx WALZER xxx
2Ø1Ø DATA 5,Ø,3,Ø,3,Ø
2Ø2Ø DATA 5,Ø,3,Ø,3,1
2Ø3Ø DATA 5,Ø,3,Ø,3,Ø
2Ø4Ø DATA 5,1,3,3,5,3,-1
3ØØØ REM xxx RUMBA xxx
3Ø1Ø DATA 5,1,1,5,1,1,3,1
3Ø2Ø DATA 5,1,1,5,1,1,3,1
3Ø3Ø DATA 5,1,1,5,1,1,3,1
3Ø4Ø DATA 5,1,1,7,1,1,3,3
3Ø5Ø DATA 5,1,1,5,1,1,3,1
3Ø6Ø DATA 5,1,1,5,1,1,3,1
3Ø7Ø DATA 5,1,1,5,1,1,3,3
3Ø8Ø DATA 19,3,3,7,1,3,7,3,-1
4ØØØ REM xxx Blues xxx
4Ø1Ø DATA 5,Ø,5,Ø,1,Ø,1,Ø,3,Ø,1,Ø
4Ø2Ø DATA 5,Ø,5,Ø,1,1,1,Ø,3,Ø,1,5
4Ø3Ø DATA 5,Ø,5,Ø,1,Ø,1,Ø,3,Ø,1,1
4Ø4Ø DATA 5,1,3,1,1,5,3,3,3,1,3,3,-1
5ØØØ REM xxx Slow-Rock xxx
5Ø1Ø DATA 12,Ø,8,Ø,1Ø,8,12,Ø
5Ø2Ø DATA 12,Ø,8,Ø,1Ø,Ø,8,Ø
5Ø3Ø DATA 12,8,8,Ø,1Ø,Ø,12,8
5Ø4Ø DATA 2Ø,1Ø,1Ø,1Ø,Ø,1Ø,1Ø,1Ø,-1
6ØØØ REM xxx Drum solo xxx
6Ø1Ø DATA 2,Ø,Ø,Ø,Ø,Ø,2,Ø,Ø,Ø,Ø,Ø
6Ø2Ø DATA 2,Ø,Ø,Ø,2,Ø,Ø,Ø,2,Ø,Ø,Ø
6Ø3Ø DATA 2,Ø,Ø,2,Ø,Ø,2,Ø,Ø,2,Ø,Ø
6Ø4Ø DATA 2,Ø,2,Ø,2,Ø,2,Ø,2,Ø,2,Ø
6Ø5Ø DATA 2,2,2,2,2,2,5,2,5,2,5,5
6Ø6Ø DATA 2Ø,1,1,7,1,1,7,1,1,2Ø,1,1
6Ø7Ø DATA 7,1,1,7,1,1,2Ø,1,1,7,1,1
6Ø8Ø DATA 2Ø,2,2,Ø,2,2,2,Ø,2,2,2,Ø
6Ø9Ø DATA 2Ø,Ø,2,7,14,8,1,1,14,1,3,3
61ØØ DATA 14,8,14,8,14,14,2Ø,8,1,38,6,6
611Ø DATA 6,6,6,36,6,6,2,2,2,2Ø,2,2
612Ø DATA 2Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,-1
7ØØØ REM xxx vlastní rytmy
7Ø1Ø DATA -1
1ØØØØ REM xxx Napsání titulu xxx
1ØØ1Ø GRAPHICS 2+16: SETCOLOR 2,6,1Ø
1ØØ2Ø ?#6;" PETER'S DRUM-KIT  " : REM Kursiva-inverzně
1ØØ3Ø POSITION 2,2 : ?#6;" 1 ROCK'n ROLL"
1ØØ4Ø POSITION 2,3 : ?#6;" 2 BLUES"
1ØØ5Ø POSITION 2,4 : ?#6;" 3 WALZER"
```

```
10060 POSITION 2,5: ? #6;" 4 RUMBA "
10070 POSITION 2,6: ? #6;"5 SLOW-ROCK"
10075 POSITION 2,7: ? #6;"6 DRUM SOLO "
10080 POSITION 2,8: ? #6;"7 VLASTNI TVORBA"
10085 POSITION 2,9: ? #6;"0 KONEC"
10090 POSITION 2,11: ? #6;"PROSIM NAPIS CISLO";
10095 RETURN
11000 REM xxx Výběr xxx
11010 OPEN #1,4,0,"K:":GET #1,A: CLOSE #1
11040 IF A=49 THEN RHYT=1010: DELAY=40: RETURN
11050 IF A=50 THEN RHYT=4010: DELAY=50: RETURN
11060 IF A=51 THEN RHYT=2010: DELAY=55: RETURN
11070 IF A=52 THEN RHYT=3010: DELAY=65: RETURN
11080 IF A=53 THEN RHYT=5010: DELAY=60: RETURN
11090 IF A=54 THEN RHYT=6010: DELAY=40: RETURN
11100 IF A=55 THEN RHYT=7010: DELAY=40: RETURN
11110 IF A=48 THEN A=USR(SNDAUS): END
11190 GOTO 11000
31000 REM xxx BASIC-Loader pro zvukový program xxx
31010 POKE 106,144: GRAPHICS 0: POSITION 10,9: ? "Inicializace"
31020 GOSUB 32000
31030 GOSUB 32500
31030 GOSUB 32500
31040 A=USR(SNDEIN)
31090 RETURN
32000 REM xxx Zvukový strojový program xxx
32010 S=0: RESTORE 32100
32020 FOR A=39936 TO 40212: READ D: POKE A,D: S=S+D;NEXT A
32030 IF S<>36579 THEN ? "CHYBA DAT !": STOP
32040 SNDEIN=39936: SNDAUS=39940: SNDTAB=40448: SNDFLG=
      1790: SNDNUM=1791
32090 RETURN
32100 DATA 104,76,222,156,104,76,245,156,216,32,109,156,32,
      18,156
32110 DATA 76,98,228,169,7,141,244,157,14,254,6,144,6,173,
      244,157
32120 DATA 32,55,156,206,244,157,16,240,173,255,6,201,255,
      240,8,32
32130 DATA 55,156,169,255,141,255,6,96,10,10,10,170,189,
      5,158,240
32140 DATA 44,189,6,158,41,3,168,189,6,158,74,74,217,240,
      157,144,28
32150 DATA 153,240,157,138,153,228,157,189,0,158,153,232,
      157,189,2
32160 DATA 158,153,236,157,189,5,158,153,224,157,32,195,
      156,96,169
32170 DATA 3,141,244,157,32,123,156,206,244,157,16,248,
      96,172,244
32180 DATA 157,185,224,157,240,63,56,233,1,153,224,157,
      208,29,190
32190 DATA 228,157,189,7,158,201,255,240,4,32,55,156,96,
      169,0,153
32200 DATA 240,157,153,232,157,153,236,157,32,195,156,96,
      190,228,157
32210 DATA 189,1,158,24,121,232,157,153,232,157,189,3,158,
      24,121,236
32220 DATA 157,153,236,157,32,195,156,96,185,232,157,72,
      185,236,157
```

```
32230 DATA 74,74,74,74,29,4,158,72,152,10,168,104,153,1,
      210,104,153
32240 DATA 0,210,96,169,0,141,254,6,169,255,141,255,6,32,
      2,157,162
32250 DATA 156,160,8,169,7,32,92,228,96,162,228,160,98,
      169,7,32,92
32260 DATA 228,32,2,157,96,169,0,162,7,157,0,210,202,16,
      250,141,8
32270 DATA 210,169,3,141,15,210,96
32500 REM xxx BLOAD xxx Inicializace strojového programu
32510 BLOAD=1536: RESTORE 32600
32520 FOR A=BLOAD TO BLOAD+33: READ X: POKE A,X: NEXT A
32530 OPEN #3,4,0"D:DRUMS.SND": REM zde nahrát zvukové soubory
32540 X=USR(BLOAD,SNDTAB,256): REM xxxNahrát soubory
32550 CLOSE #3: RETURN
32600 DATA 104,162,48,169,7,157,66,3,104,157,69,3,104,157,
      68,3,104
32610 DATA 157,73,3,104,157,72,3,32,86,228,132,212,169,0,
      133,231,96
```

JAK RYCHLÝ JE VÁŠ ATARI ?

" S procesorem 3,Ø MHz" – "Hodinový kmitočet 1,79 MHz," "Rychlejší když je obrazovka vypnutá" - to jsou všechno údaje, které jste už určitě četli v prospektu nebo časopisech. Ale - jak rychle skutečně běží váš ATARI? Může počítat rychleji než např. C-64 nebo Apple?
Začněme tím nejjednodušim: srdcem vašeho počítače Atari je mikroprocesor 65Ø2, stejný integrovaný obvod, jaký slouží v Apple a také v C-64. V C-64 je sice 651Ø, ten je však až na několik maličkostí totožný s 65Ø2. Pracovní rychlost procesoru závisí na kmitočtu taktovacího signálu, který si lze názorně představit jako základní rytmus jeho práce. V taktovacím cyklu lze uskutečnit výběr z paměti nebo vnitřní procesorovou operací. Váš ATARI pracuje s taktovací frekvencí 1,79 MHz, zvládne tedy téměř dva miliony cyklů za sekundu. Když si uvědomíme, že nejjednodušší příkaz pro sčítání potřebuje dva takové cykly, znamená to, že váš ATARI by mohl za sekundu dokázat skoro milion sčítání. Teď již můžeme porovnávat:

Apple	1,Ø23 MHz
ATARI	1,79 MHz
Commodore 64	Ø,98 MHz

Podle toho by mohl být počítač ATARI téměř dvakrát rychlejší než Apple nebo C-64. Bohužel, jak se už v životě stává, zdání klame. Jsou zde dva faktory, které procesora brání, aby pracoval tak rychle,jak by vlastně mohl: proces obnovování paměti (Refresh) a přímý přístup do paměti (DMA).

Paměť s krátkodobým zapamatováním, obnovováním-zotavováním údajů

Paměť vašeho ATARI je postavena z dynamických paměťových obvodů RAM. Jsou to paměťové prvky, které vaše informace mohou podržet jen krátkodobé (řádově milisekundy). Pro ty z vás, kteří mají ponětí o elektronice, budiž řečeno,že se přito jedná o náboje kondenzátorů, které se přes odpory stále zmenšují. Aby se tento efekt dal rozumně využít pro účely paměti dat, bylo nutno sáhnout ke triku: celá paměť

se pravidelně "osvěžuje" (proto Refresh), přičemž je čtena po blocích. Obnovovací proces pochopitelně stojí čas a ten se získává na úkor 65Ø2 pomocí tzv. "kradení cyklů - Cycle stealing". Konkrétně to znamená, že když je nutný obnovovací cyklus, musí se 65Ø2 na dobu jednoho cyklu zastavit a tím o něj "okrást". Obnovení žádá mikroprocesor ANTIC, jenž se také stará o přímý přístup do paměti, takže není divu, že obnovování paměti je úzce svázáno s vyvtvářením obrazu: v normálním případě bude během trvání jednoho řádku devět obnovovacích cyklů. "Normálním" proto, že u jistých grafických modů existují výjimky.
Úplný obraz se skládá ze 312 řádků (nikoliv 192, zobrazitelných televizorem), za sekundu se generuje 5Ø obrazů, takže pro obnovení potřebujeme:

9 * 312 * 5Ø = 14Ø4ØØ cyklů.

Ve skutečnosti je jich víc, protože zotavování probíhá i při vertikálním zetmnění, ty však můžeme proti uvedenému počtu klidně zanedbat. Pro počítání zbývá celkem ještě

1 79Ø ØØØ - 14Ø 4ØØ = 1649 6ØØ cyklů,

což odpovídá teoretickému taktovacími kmitočtu asi 1,65 MHz.

Grafiak stojí čas - DMA

Má-li být něco vidět na obrazovce - což se většinou požaduje - musejí se obrazové informave dostat z paměti do obrazového signálu. To je zařízeno přímým přístupem do paměti (DMA - Direck Memory Access),čímž se rozumí postup, při kterém jsou data přenášena do paměti nebo z paměti, aniž by se tím procesor aktivně zabýval. Přesněji řečeno, procesor se sice tímto přenosem dat nezabývá, je jím však ovlivňován.Princip je stejné kradení cyklů jako u obnovování paměti - přečtení jednoho byte pro DMA, odebere procesoru 65Ø2 jeden cyklus. Výpočet součtu cyklů, které odpadly kvůli DMA pro účely zobrazení, není už tak jednoduchý, protože počet čtených byte značně závisí na grafickém modu, a těch je u ATARI víc než dost. Největší nároky na spotřebu DMA má z nich nejčastěji používaný textový mod GRAPHICS Ø. Ten potřebuje na vytvoření obrazu 433 4ØØ cyklů za sekundu (viz výpočet v rámečku). Spočítáme-li si dosavadní bilance, zbývá nám:

1 79Ø ØØØ - 14Ø 4ØØ - 433 4ØØ = 1 216 2ØØ cyklů

Cykly pro	
Display list	: 1
čtení čísel znaků	: 4Ø
čtení sady znaků	: 8 * 4Ø
24 řádků/obraz	: 361 * 24
obnovení D-list	: 8664 + 8
5Ø obrazů/s	: 8672 * 5Ø
cyklů/s celkem	: 433 4ØØ

Výpočet cyklů připadajících na DMA v GRAPIHCS Ø

Tak docela náš výpočet ještě nesouhlasí, protože, jak už bylo poznamenáno výše, zotavení nepotřebuje vždy devět cyklů. To platí právě v GR. Ø kde při řádkování u horního okraje probíhá zotavovací cyklus během textového řádku jednou. Důvodem je časová tíseň DMA, která se v tomto řádku projevuje, neboť musí být přečtena jak čísla znaků, tak první řádek znakové sady. Z původních 114 cyklů, které byly k dispozici pro jeden řádek stínítka, zbývá pro řádek tohoto typu (v nejhorším případě) pouhých 30 ! Celkem se ušetří (8 * 24 řádky * 5Ø snímku) 9 6ØØ zotavovacích cyklů za sekundu, takže počet cyklů, jež odpadnou kvůli zotavení,se snižuje ze 14Ø 4ØØ na 13Ø 8ØØ.

To ještě není všechno:

Další příčinu zpomalení výpočetní rychlosti nalezneme v systémovém software: VBI (viz odstavec stejného názvu). Tento program je požadován padesátkrát za sekundu a potřebuje pokažde asi 1ØØØ cyklu (přirozeně jen tehdy, pokud jste do VBI nepřidali žádné přídavné programy). Sečteno a podtrženo to dává dalších 5Ø ØØØ cyklů, ve kterých procesor není k dispozici. Z toho plyne závěr, že v jedné sekundě je použitelných

1 79Ø ØØØ - 13Ø 8ØØ - 433 4ØØ - 5Ø ØØØ = 1 175 8ØØ cyklů
(cyklů/s) (zotavení) (DMA) (VBI)

což odpovídá procesoru 65Ø2 s taktovacím kmitočtem cca 1,17 MHz. Přísně vzato, nedá se pochopitelně mluvit o taktovací frekvenci, protože zpožďovací vlivy (z hlediska procesoru)

probíhají poměrně nepravidelně. Mimochodem to je také důvod,proč syntezátor řeči "S.A.M." od firmy Dorit Ash Software při zapojeném DMA a VBI zní značně chraptivě. Když používáte jště DMA pro grafiku Player-Missile (hráč-střela), musíte počítat s odpadnutím dalších 64 ØØØ cyklů/s.

Porovnání doby chodu testovacího programu

Počet cyklů v programu : 59 136 9ØØ
Údaje v tabulce: vypočtená takt.frekvence v MHz/doba v s.

	ATARI	C - 64	APPLE
Textový mod	1,16	0,92	1,02
VBI aktivní	5Ø,9	54,6	60,5
DMA vypnuto	1,62	0,98	_"_
VBI vypnuto	36,4	60,3	

Abychom této šedivé teorii přidali trochu barvy,změřili jsme dobu chodu téhož programu na různých počítačích s mikroprocesorem 65Ø2 a výsledky uspořádali do výše uvedené tabulky pro vzájemné porovnání. K testu byl použit krátký program ve strojovém kódu, otištěný na konci této kapitoly. Pokročilí programátoři z něj hned poznají, jak se v assembleru vypočítá celkový počet cyklů.
Jak vidíte váš počítač ATARI je stále ještě o 26 % rychlejší než Comodore C64 a pokud se u obou vypne zobrazování a VBI, je ATARI dokonce o 65% vpředu.
Ostatně, abychom předešli nedorozumnění - rychlost procesoru má jen nepřímý vliv na rychlost výpočtu v Basicu. Zde hrají rozhodující roli ještě jiné faktory, jako např.: Přesnost výpočetních operací a druh aritmetiky (BCD resp. binární).

Testovací program pro měření doby chodu:

```
Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Ø11Ø ; Testovací program pro zjištění doby chodu
Ø12Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Ø13Ø ;
```

```
     =ØØCD   Ø14Ø COUNT  =   ØCD      ;stránka Ø,pomocný reg.
             Ø15Ø ;
ØØØØ         Ø16Ø        *=  ØØ6ØØ
             Ø17Ø ;
Ø6ØØ A964    Ø18Ø START  LDA #1ØØ     ;*2*  1ØØ průchodů
Ø6Ø2 B5CD    Ø19Ø        STA COUNT    ;*2*
Ø6Ø4 AØØØ    Ø2ØØ SCHLF1 LDY #Ø       ;*2* 256 průchodů
Ø6Ø6 A2ØØ    Ø21Ø SCHLF2 LDX #Ø       ;*2*
Ø6Ø8 EA      Ø22Ø SCHLF3 NOP          ;*2*
Ø6Ø9 EA      Ø23Ø        NOP          ;*2*  vnitřní smyčka
Ø6ØA CA      Ø24Ø        DEX          ;*2*  9 cyklů
Ø6ØB DØFB    Ø25Ø        BNE SCHLF3   ;*2/3*
             Ø26Ø ;
Ø6ØD BB      Ø27Ø        DEY          ;*2*  prostř.smyčka
Ø6ØE DØF6    Ø28Ø        BNE SCHLF2   ;*2/3*
             Ø29Ø ;
Ø61Ø C6CD    Ø3ØØ        DEC COUNT    ;*5*   vnější smyčka
Ø612 DØFØ    Ø31Ø        BNE SCHLF1   ;*2/3*
             Ø32Ø ;
Ø614 6Ø      Ø33Ø        RTS          ;*6*
             Ø34Ø ;
             Ø35Ø ; Počet cyklů:
             Ø36Ø ; ------------
             Ø37Ø ;
             Ø38Ø ;Vnitřní smyčka: (9*256) - 1 = 23Ø3
             Ø385 ;(BNE jen 2 cykly při posledním průchodu)
             Ø39Ø ;prostřední smyčka:(23Ø3+7) * 256-1=591 359
             Ø4ØØ ;vnější smyčka:(591 359+1Ø) * 1ØØ-1=59 136 899
             Ø41Ø ; ostatní  : 59 136 899+11 59 136 91Ø
             Ø42Ø ;
             Ø43Ø ; celkový součet: cca 59 136 9ØØ cyklů
             Ø44Ø ;
```

TŘI N O V É GRAFICKÉ MÓDY :

Četli jste správně - kromě 17 grafických modů, které váš ATARI dosud ovládal, dřímají v jeho elektronických hlubinách ještě další tři. Tím máte k volnému použití celých 20 graf. modů, které při větší hojnosti můžete ještě téměř libovolně směšovat. Který jiný domácí počítač s tím dokáže držet krok?

Tři nové grafické mody jsou tzv. znakové mody, tj. jsou vybudovány z jednotlivých znaků a vyžadují zvláštní znakovou sadu. Zvláštností těchto stupňů grafiky je to, že jeden znak může současně obsahovat až 16 různých barev a celá obrazovka přitom vyžaduje pouze necelý kilobyte paměti.

Dále jsou označeny jako GRAPHICS 9/0, 10/0 a 11/0. Proč bylo zvoleno právě toto označení, hned uvidíte.
Podrobné údaje o grafických stupních:

Mod	znaků na řádek	výška v TV řádcích	použité barvy
GR. 9/0	40	8	16 stupňů jasu jedné barvy
GR.10/0	40	8	9 barev řízen. barv.reg.
GR.11/0	40	8	16 barev jednoho jasu

Možná jste už poznali, že přitom musí mít prsty ve hře "GTIA" - hádáte správně, ale nejprve podrobný

pohled do vnitřního dění :

Množství druhu obrazové grafiky počítače ATARI je umožněno úzkou spolupráci dvou integrovaných obvodů ANTIC a GTIA. ANTIC, který lze již téměř nazvat "video-mikroprocesorem", obstarává data pro obrazovku prostřednictvím DMA a předává je dále na GTIA, jenž spolupracuje na řízení barev a nakonec vitváří videosignál. Z tohoto uspořádání vyplývají dva zásedně rozdílné druhy grafických modů: první tvoří skupinu čtrnácti takzvaných modů ANTIC, určených obrazovým programem DL (Display-list), druhý je dán třemi mody GTIA, které vzniknou, když jeden tentýž mod ANTIC je v GTIA rozdílně interpretován.

Podívejme se nejprve na základní grafické stupně GRAPHICS 9,10,11, neboť je představují výše tady zmíněné mody GTIA. Rozlišovací schopnost je ve všech třech případech 80(H) x 192(V), potřebují paměť 7680 byte (plus Display-list), stejně tolik, jako je zapotřebí pro GR. 8, grafiku s vysokou rozlišitelností 320(H) x 192(V) bodů. To celkem nepřekvapuje, protože GR.9, 10 a 11 je použit stejný mod ANTIC (ANTIC"F") jako pro GR.8, jen data dodávaná obvodem ANTIC jsou zcela jinak interpretovaná. V GR.8 je každý jednotlivý bit vyhodnocen jako jasová informace barvového bodu. Naproti tomu GR.9 shrnuje vždy 4 po sobě následující bity a používá je jako podrobnou jasovou informaci pro rozsáhlejší jasový bod, takže vzniká možnost ovládání šestnácti stupňů jasu jedné barvy. Tím se samozřejmě omezuje vodorovná rozlišovací schopnost ze 320 na 80 bodů pro řádek, svislá však zůstává jako předtím 192 bodů. Tento mod, jehož rozlišovací schopnost se vlastně již nedá nazvat vysokou, se však výborně hodí pro digitalizované obrazy, které mohou stínováním vytvořit neobyčejně plastický dojem. Stejnou metodou se vytvoří mody 10 a 11, kde v prvním případě je čtyřbitová informace využívaná pro výběr z devíti barvových registrů (GR.10). U tohoto posledního modu by teoreticky bylo možno 16 barev, protže ale v hardware je pouze 9 barvových registrů, musíme se s nimi spokojit.

Nápadný společný rys těchto modů :

jak plyne z výše uvedené úvahy, je způsob jejich vytváření. Vznikají jednoduchým přepínáním obvodu GTIA z jednoho a téhož grafického stupně ANTIC-F. Stejnou bodovou rozlišovací schopnost, ale ve znakovém mpdu, vytvoří normální GR,0 (ANTIC-mod 2) pro zobrazení textu. Jediný rozdíl mezi GR.8 a GR.0 je ve větší práci, kterou má ANTIC s GR.0, protože informace o obrazových bodech místo příjmu z obrazové paměti přichází oklikou přes znakovou sadu. Nabízí se otázka, jestli se nedají tři mody GTIA použít také s GR.0 a ono to funguje! Chcete-li to hned vyzkoušet, zadejte jen (v BASICu) příkaz

POKE 623,128

a tím jsme zapnuli GRAPHICS 10/0. Paměťové místo 623 je "stínem GTIA registru PRIOR, jehož bity 6-7 se používají k přepínání modů GTIA:

hardw.reg. GTIA — stínový reg. v paměti

PRIOR bit	7	6	5	4	3	1	0	GPRIOR
$D01B	0	0	- mod GTIA vypnut					$2B
53275	0	1	- GR.9/0 zapnut					623
	1	0	- GR.10/0 zapnut					
	1	1	- GR.11/0 zapnut					

Registry pro mody GTIA

Na nové grafické mody se můžete v BASICu dostat takto:

POKE 623,64	GRAPHICS 9/0
POKE 623,128	GRAPHICS 10/0
POKE 623,192	GRAPHICS 11/0

Když zkusíte místo stínového registru přepnout přímo hardwarový registr PRIOR, budete odměněni jen krátkým bliknutím grafiky GTIA: během 1/50 s se PRIOR přepíše svou starou hodnotou ze stínového registru. Přímá změna PRIORu může mít však také dobrou stránku. Když se ke změně PRIORu využije přerušení typu DL, lze míchat mod GTIA s normálním modem ANTIC.
Když teď tedy stisknete několik písmenových kláves, uvidíte pestré abstraktní znaky, které se svým dřívějším vzhledem v GR.0 nemají nic společného a zde je problém, aby tyto grafické mody bylo možno rozumně využít, potřebujeme speciálně přizpůsobenou znakovou sadu. Znak se nyní skládá z matice 2(H) x 8(V) a vypadá takto:

bit	7	6	5	4	3	2	1	0	
									každé čtyři bity určují
									GR.9/0 : jas
									GR.10/0: Barvový reg.
									GR.11/0: barva
									Nová znaková matice

S tím se sice už nedají zobrazit žádná písmena nebo číslice, ale pro mnohobarevné grafické znaky je tento druh znakového modu velmi užitečný. Při vhodné volbě znakové sady můžete do jednoho kilobyte dostat obrazy, které dříve v bitově mapovaných grafických stupních 9,10,11 vyžadovaly téměř 8 kB. Nebo představte si pohybující se herní pole o rozměrech několikanásobně větších než je plocha stínitka obrazovky, v 16 zářivých barvách ...

Příklady - příklady - příklady

Ve výpisu 1 je příklad programu pro GR.11/0 s 15 barevnými svislými pruhy (plus barva pozadí). K tomu se vytváří (od řádku 1000) znaková sada, kde je písmenům A-O přiřazen vždy blok jedné barvy. Chcete-li experimentovat sami, potřebujete pouze zastavit program stiskem BREAK a můžete klávesami A-O vyvolat nové znaky.

Jas můžete nastavovat pomocí
SETCOLOR 4,Ø,<jas>; jas od Ø do 14
(v následujícím příkladu viz řádek 9Ø), je jen důležité, aby v pozici pro určení barvy byla nula, protože uskutečněná informace o barvě barvového bodu vznikne logickým součtem zde udané hodnoty a čytř bitů pro barvový bod. Zpět do normálního textového modu se nejjednodušeji dostanete stiskem "Systém RESET".

```
1Ø   REM ***** Ukázka GRAPHICS 11/Ø *****
2Ø   POKE 1Ø6,144: GRAPHICS Ø: REM Rezervování místa pro
     sadu znaků
3Ø   CHRSET=(9*16+12)*256: REM Adresa nové sady znaků
4Ø   POKE 752,1: ?"\": REM Vypnutí kurzoru, "\"=ESC CTRL-CLEAR
5Ø   GOSUB 1ØØØ: REM Sestavení sady znaků
6Ø   POKE 756,CHRSET/256: REM Přepnutí na novou sadu znaků
7Ø   POKE 623,192: REM Vyvolání GRAPHICS 11/Ø
9Ø   SETCOLOR 4,Ø,8: REM Nastavení jasu
1ØØ  FOR I=Ø TO 15: REM Kreslení zobrazovaných vzorů
11Ø  COLOR ASC ("A")+I: X=I*2+5: PLOT X,Ø: DRAWTO X,23
12Ø  NEXT I
19Ø  GOTO 19Ø: REM Pro zachování grafiky
1ØØØ REM * GR.11/Ø - Sestavení sady znaků
```

```
1Ø1Ø FOR Z=Ø TO 15: REM 16 nových znaků
1Ø2Ø OFFSET=(Z+32)*8: REM Poloha v paměti
1Ø3Ø FMUST=Z*17: REM Barevny vzor
1Ø4Ø FOR Z1=Ø TO 7: POKE CHRSET+OFFSET+Z1,FMUST: NEXT Z1
1Ø5Ø NEXT Z
1Ø9Ø RETURN
```

(Výpis 1: Demonstrační program pro GRAPHICS 11/Ø)

V modu GR.1Ø/Ø lze již provádět skutečně efektivní barevnou animaci pomocí barvových registrů, příklad najdete ve výpisu 2. Princip je zcela jednoduchý: bitové vzory ve znakové sadě nepopisují jako v GR.9/Ø pevně danou barvu, nýbrž odkazují nepřímo na barevný registr. Vytvoříte-li vhodný obraz a necháte barvovými registry probíhat nějakou hodnotu, uvidíte jen ty bitové vzory, jejichž příslušné barvové registry právě obsahují informaci o barvě. Podobným způsobem pracuje program z výpisu 2, jen zde putuje přes barvové registry celý proud hodnot barev. Dosahuje se tím psychedelického efektu, jenž připomíná let tunelem.

Tyto nové grafické mody můžeme také míchat s jinými, ačkoliv to zde není tak jednoduché jako s normálními mody ANTIC. To už je vidět ze skutečnosti, že v obyčejných grafických stupních GR.9-11 nejsou textová okénka, jako např. v GR.8 (přečtěte si o tom také v následujícím odstavci).

Výpis 2:

```
1Ø   REM *** Ukázka GR. 1Ø/Ø-ANIMACE BARVOVYMI REGISTRY ***
2Ø   POKE LØ6,144: GR.Ø: REM Vyhrazení místa pro sadu znaků
3Ø   CHRSET=(9*16+12)*256: REM Adresa nové sady znaků
4Ø   POKE 752,1: REM Vypnutí kurzoru
5Ø   GOSUB 1ØØØ: REM Přepnutí na novou sadu znaků
6Ø   POKE 756,CHRSET/256: REM Přep.na novou sadu znaků
8Ø   POKE 623,128: REM Vyvolání GRAPHICS 1Ø/Ø
9Ø   FOR I=1 TO 8: POKE 7Ø4+I,I*2: NEXT I: REM Vytvořit
     obraz na obrazovce
1ØØ  FOR I=Ø TO 11: COLOR ASC("A")+I-8*(I>7)
11Ø  PLOTI,I: DRAWTO 39-I,I: DRAWTO 39-I,23-I: DRAWTO I,23-I:
     DRAWTOI,I
```

```
12Ø  NEXT I
15Ø  FOR I=Ø TO 8: A=712-I: C=PEEK(A)+2: IF C>255 THEN
     C=Ø
16Ø  POKE A,C: NEXT I: REM Přeskupování hodnot barv.reg.
18Ø  FOR T=Ø TO 1Ø: NEXT T
19Ø  GOTO 15Ø
1ØØØ REM GR.1Ø/Ø - Sestavení sady znaků
1Ø1Ø FOR Z=Ø TO 15: REM 16 nových znaků
1Ø2Ø OFFSET=(Z+32)*8: REM Poloha v paměti
1Ø3Ø FMUST=Z*17: REM Barevný vzorek
1Ø4Ø FOR Z1=Ø TO 7: POKE CHRSET+OFFSET+Z1,FMUST: NEXT Z1
1Ø5Ø NEXT Z
1Ø6Ø RETURN
```

Výpis 3:

```
1Ø   REM  *** GR. 9/Ø - Ukázka s textovým okénkem ***
2Ø   POKE 1Ø6,144: GR.Ø: REM Vyhrazení místa pro sadu zn.
3Ø   CHRSET=(9*16+12)*256: REM Adresa nové sady znaků
4Ø   POKE 752,1: REM Vypnutí kurzoru
5Ø   GOSUB 1ØØØ: REM Sestavení sady znaků
6Ø   GOSUB 3ØØØ: REM Úprava strojového programu pro DLI
     pomocí POKE
7Ø   GOSUB 2ØØØ: REM Aktivace DLI
8Ø   POKE 756,CHRSET/256: REM Přepnout na novou sadu znaků
9Ø   POKE 623,64: REM Příkaz pro vyvolání GRAPHICS 9/Ø
1ØØ  SETCOLOR 4,2,Ø: REM Nastavení barvy
11Ø  FOR I=Ø TO 19: REM Kreslení obraz.vzorů
12Ø  POSITION Ø,I: ?"KLMNONMLKJIHGFEDCBA ABCDRFGHIJKLMNO
     NMLKJ";
13Ø  NEXT I
14Ø  POKE 7Ø3,4: REM Zapnutí textového okénka
15Ø  POKE 752,Ø: REM Kurzor opět zapnut
16Ø  LIST : END
1ØØØ REM GR.9/Ø - Sestavení sady znaků
1Ø1Ø FOR Z=Ø TO 15: REM 16 nových znaků
1Ø2Ø OFFSET=(Z+32)*8: REM Poloha v paměti
1Ø3Ø FMUST=Z*17: REM Barevný vzorek
1Ø4Ø FOR Z1=Ø TO 7: POKE CHRSET+OFFSET+Z1,FMUST:NEXT Z1
1Ø5Ø NEXT Z
```

```
1Ø9Ø RETURN
2ØØØ REM Aktivování DLI pto textové okénko
2Ø1Ø DLIVEC=512: NMIEN=54286: SDLDTL=56Ø
2Ø2Ø DLIST=PEEK(561)*256: REM Adresa DL
2Ø3Ø POKE DLIST+24,2+128: REM Nastavení bitu pro DLI
2Ø4Ø POKE DLIVEC,Ø: POKE DLIVEC+1,6: REM Nasměrování
     vektoru DLI
2Ø5Ø POKE NMIEN,192: REM DLI bude aktivní
2Ø9Ø RETURN
3ØØØ REM Upravit strojový program pro DLI
3Ø1Ø RESTORE 31ØØ
3Ø2Ø FOR A=1536 TO 1558: READ D: POKE A,D: NEXT A
3Ø9Ø RETURN
31ØØ DATA 72,138,72,173,111,2,41,63,162,224,141,1Ø,
     212,141,27,2Ø8
311Ø DATA 142,9,212,1Ø4,17Ø,1Ø4,64
```

Barva obrazu je určena příkazem SETCOLOR 4,2,Ø na řádku 1ØØ. Je zde zase důležité, abyste pro hodnotu jasu zadali nulu, protože jas barvového bodu je výsledkem logického součtu této nuly se čtveřicí bitů pro barevný bod.

Při směšování s jinými mody musí být GTIA během výstavby obrazu přepínán prostřednictvím tak zvaných přerušení obrazového programu (DLI). V našem příkladu jsou bity 6 a 7 stínového registru GPRIOR vynulovány, tato hodnota je zapsána do registru PRIOR a tím je vypnut mod GTIA. V dalším průběhu bude ukazovátko znakové sady opět nasměrováno na normální znakovou sadu v ROM. K lepšímu porozumnění těchto pochodů následuje výpis programu v assembleru pro strojový program DLI, který ve výše uvedeném programu v BASICu (výpis 3) je obsažen ve formě řádků dat.

Ještě několik slov závěrem. Použití nových grafických modů je podmíněno obvodem GTIA. Na starších přístrojích, které ještě obsahovaly CTIA, neprodokují výše uvedené programy žádnou rozumnou grafiku. Pro útěchu budiž řečeno že většina počítačů ATARI prodávaných v EVROPĚ a všechny nové modely XL obsahují integrovaný obvod GTIA.

Výpis 4 :

```
                Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø11Ø ;Přerušení obraz.programu-DL Interrupt
                Ø12Ø ;pro textové okénko v modu GTIA
                Ø13Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø14Ø ;
     =Ø26F      Ø15Ø GPRIOR=623          ;stínový reg.PRIORu
     =DØ1B      Ø16Ø PRIOR  =   $DØ1B
     =D4ØA      Ø17Ø WSYNC  =   $D4ØA    ;čekání na horiz.synchro
     =D4Ø9      Ø175 CHBASE =   $D4Ø9    ;adresový registr sady znaků
                Ø18Ø ;
ØØØØ            Ø19Ø        *=  $Ø6ØØ    ; na straně 6
                Ø2ØØ ;
Ø6ØØ 48         Ø210 DLI    PHA          ;uschování obsahu akumu-
                                           látoru
Ø6Ø1 8A         Ø215        TXA          ; uschování obsahu X registru
Ø6Ø2 48         Ø217        PHA
Ø6Ø3 AD6FØ2     Ø22Ø        LDA GPRIOR
Ø6Ø6 293F       Ø23Ø        AND #$3      ;vynulování bitů 6/7
Ø6Ø8 A2EØ       Ø235        LDX #224     ;normální sada znaků
Ø6ØA 8DØAD4     Ø24Ø        STA WSYNC    ;čekání na konec řádku
Ø6ØD 8D1BDØ     Ø25Ø        STA PRIOR    ; zapnutí normál. GTIA
Ø61Ø 8EØ9D4     Ø255        STX CHBASE   ;obnovení původ. sady znaků
Ø613 68         Ø26Ø        PLA          ; vrácení registru X
Ø614 AA         Ø261        TAX
Ø615 68         Ø262        PLA          ;vrácení akumulátoru
Ø616 4Ø         Ø27Ø        RTI          ;ukončení DLI
                Ø28Ø;
```

Textové okénko v modech GRAPHICS 9/10/11

Předcházející strojový program pro DLI lze také velmi výhodně využít v normálních grafických modech GTIA 9-11 a také v nich zavést obvyklé čtyřřádkové textové okénko. Tím se mody GTIA stanou mnohem příjemnější pro uživatele, proto je také možno ručně zadávat příkazy PLOT nebo DRAWTO a pozorovat výsledek, aniž by bylo nutné zase přepínat na GR.Ø.

Při vyvolávání těchto grafických modů je ovšem nutný trik: nesmíme požadovat přímo GR. 9-11, místo toho vyvoláte GR.8, instalujete DLI do programu Display List a teprve pak přepnete pomocí GPRIOR (viz minulý oddíl) na příslušný mod GTIA.

V másledujícím výpisu naleznete příklad právě popsaného postupu. Podprogram od řádku 31ØØØ můžete přímo použít ve svých programech jako náhradu za příkazy GR. 9-11.

```
1Ø    REM  Mody GTIA s textovým okénkem (příklad GR.9)
2Ø    REM Zde může být váš program
3Ø    GR=9: GOSUB 31ØØØ: REM GR rovná se grafickému modu
                                (9-11)
1ØØ   SETCOLOR 4,1,Ø: REM Nastavení barvy
11Ø   FOR I=Ø TO 15: REM Kreslení zobrazovaných vzorů
12Ø   COLOR I: PLOT I*5,Ø: DRAWTO I*5,159
13Ø   NEXT I
16Ø   LIST : STOP
18Ø   REM
19Ø   REM začátek podprogramu k vyvolání GR. 9-11/TXT
2ØØ   REM grafický mod musí být udán proměnnou GR
21Ø   REM
31ØØØ REM GRAPHICS GR s textovým okénkem - zapnutí
31ØØ5 GOSUB 32ØØØ: REM DLI - program ve stroj.kodu
31Ø1Ø GR.8: REM nejprve GR.8 s textovým okénkem
31Ø2Ø DLIVEC=512: NMIEN=54286: SDLDTL=56Ø
31Ø3Ø DLIS=PEEK(56Ø) + PEEK(561):256: REM Adresa DL
31Ø4Ø POKE DLIST+166,15+128: REM Nastavení bitu pro DLI
31Ø5Ø POKE DLIVEC,Ø: POKE DLIVEC+1,6: REM Směrování vekto-
      ru DLI
31Ø6Ø POKE NMIEN,192: REM DLI bude aktivní
31Ø7Ø GTIA=64: IF GR=1Ø THEN GTIA=128: REM Hledání hodnoty
      pro GPRIOR
31Ø8Ø IF GR=11 THEN GTIA=192
31Ø9Ø POKE 623,GTIA: REM Přepnout GTIA
311ØØ POKE 87,GR: REM O.S. nastavit na správný graf.mod
3111Ø RETURN
32ØØØ REM Zavedení programu ve strojovém kodu
32Ø1Ø RESTORE 321ØØ
```

```
32Ø2Ø FOR A=1536 TO 1558: READ D: POKE A,D: NEXT A
32Ø9Ø RETURN
321ØØ DATA 72,138,72,173,111,2,41,63,162,224,141,1Ø,212,
      141,27,2Ø8
3211Ø DATA 142,9,212,1Ø4,17Ø,1Ø4,64
```

Disketový VSTUP / VÝSTUP ve strojovém kodu

Disketové I/O operace mohou probíhat ve dvou rovinách. První je datová rovina I/O, ve které se obvykle zpracovávají soubory a druhá je Handlerova rovina (sestavení návěstí souborů, které udávají počítači, jak bude využívána obrazovka, disketa ap.), ve které jsou možné jenom jednoduché příkazy, např. čtení nebo psaní sektoru. Budeme se zabývat výlučně rovinou datovou. Se soubory budeme pracovat pomocí OPEN, CLOSE a příkazy pro psaní a čtení dat. Tyto funkce jsou také k dispozici v assembleru, jenom musíte vědět jak je vyvolat.

Základy: DATOVÉ SOUBORY

Co to je soubor? U diskety soubor představuje blok dat, který se nachází na disketě pod svým jménem souboru, uvedeným v hlavičce diskety.
Následující příklad vytvoří malý sektor- soubor:

```
1Ø OPEN #1,8,Ø,"D:PRIKLAD.BIN"
2Ø FOR I=Ø TO 999: PUT #1,Ø: NEXT I
3Ø CLOSE #1
```

Když se teď podíváme do seznamu na disketě, najdeme náš příklad uložený pod jménem PRIKLAD.BIN. Pro čtení takového souborumusíme udělat následující: Nejprve soubor pod jeho jménem otevřeme pro čtení, potom Byte po Byte načteme a soubor znovu uzavřeme. Jsou dvě možnosti: Buď víme, jak je soubor dlouhý, nebo to uděláme jako v následujícím příkladu:

```
1Ø OPEN #1,4,Ø,"D:PRIKLAD.BIN"
2Ø TRAP 5Ø
3Ø GET #1,D:?D: GOTO 3Ø
5Ø CLOSE #1:TRAP 4ØØØØ
```

Soubor bude načítán tak dlouho, pokud budou Byte k dispozici. Když bude načten soubor, jeho konec, objeví se chyba (ERROR 136), kterou zachytíme příkazem TRAP. Při zkoušení uvedených příkladů se ukáže, jak je BASIC pro I/O pomalý.
Chceme-li použít I/O operace v assembleru, musíme znát,jak I/O příkazy volat ve strojovém kodu.

CENTRÁLNÍ rutina pro vstup/výstup CIO

Celá paleta I/O operací v datové rovině je zvládnuta jednou systémovou rutinou, která se volá přes vektor CIOV (ØE456). Této rutině musíme zadat, který příkaz I/O chceme provádět, musíme nejprve vyplnit jistý druh "formuláře", do kterého zaznačíme kody příkaza eventuelní parametry operací I/O. Osm tzv."formulářů" se v operačním systému ATARI jmenuje IOCB (input/output control block),, začínají na paměťovém místě Ø34Ø. Ukážeme si nejprve stavbu těchto IOCB.:

IOCB byte	Jméno pole	Funkce
+Ø	ICHID	Identifikace. Byte je nastaven v CIO, je-li <>255, je IOCB volné
+1	ICDNO	Číslo zařízení pro VSTUP/VÝSTUP. Důležité pro diskety, zde se nachází číslo driveru. Rovněž tento Byte je nastaven v CIO během příkazu OPEN.
+2	ICCMD	Zde je zaznamenán kod příkazu.
+3	ICSTA	Stavový byte, je nastaven v CIO,hodnota>128, znamená chybu
+4	ICBAL	Počáteční adresa buferu souboru L/H. Zde je
+5	ICBAH	zaznamenána buď adresa souboru (při OPEN ...) nebo rozsah buferu (při příkazu zápis/čtení)
+6	ICPTL	Speciální ukazatel na adresu (-1) rutiny
+7	ICPTH	PUT CHARACTERS (L/H), v běžném případě se nepoužívá.

+8	ICBLL	Před voláním CIO zde musí být uložena délka
+9	ICBLH	aktuálního buferu, CIO zde uloží počet skutečně načtených byte.
+1Ø	ICAX1	Přídavná informace, např. je-li soubor otev-
+11	ICAX2	řen pro zápis nebo čtení. ICAX1/2 nesmí být
+12	ICAX3	změněn po OPEN.
		Přídavné informace, které využívá handler. Tyto byte se používají v NOTE a POINT.
+15	ICAX 6	

Po vyplnění formuláře IOCB (podrobně viz dále) musíme zavolat CIO, přirozeně až po zápisu do X registru, který IOCB chceme použít. Každý IOCB má délku 16 byte, X-registr představuje ukazatel na aktuální IOCB relativně vztažený k IOCB č. Ø. V X-reg. bude číslo IOCB násobené 16-ti.

Když informace postupuje zpět, získáme ji přes IOCB, kromě dat nacházejících se v buferu,které byly předány IOCB. Hlášení stavu statusu obdržíme dodatečně dotazem na pole ICSTA. Status je v CIO zaznamenán i do Y-registru.

IOCB je celkem 8, můžeme tedy současně otevřít 8 souborů. V praxi to není úplná pravda, protože operační systém využívá některé IOCB pro sebe, např.IOCB#Ø je trvale obsazen screen-editorem, IOCB#6 a #7 jsou používány v souvislosti s grafickými režimy a příkazy LOAD/SAVE a LPRINT. Vy máte k dispozici IOCB 1-5

a) OTEVŘENÍ SOUBORU

otevření souboru pro zápis:

```
                    OPEN #2,8,Ø,"D:PRIKLAD.BIN"
```

Definujeme pole IOCB jen pro IOCB č. Ø (od $34Ø) a do X-reg. uložíme číslo IOCB násobené 16-ti. Když se budeme obracet na IOCB, použijeme indexové adresování přes registr X:

```
ICCMD = $34Ø+3     ;pole pro kod příkazu IOCB Ø
....
LDX  #2*16         ;použito  IOCB#2
....
STA ICCMD,X        ;příkaz uložit do IOCB#2
```

Při této metodě stačí jednou definovat návěstí IOCB. Chceme-li využít jiný IOCB, změníme jenom hodnotu X-registru.

Nyní můžeme volat IOCB přes vektor CIOV (ØE456) jako podprogram, přitom musí X-reg. obsahovat č. IOCB násobené 16-ti. Toto číslo jsme do X-reg uložili už na začátku, takže stačí provést JSR CIOV.

V Y-reg. obdržíme zpětné hlášení o úspěchu (nebo neúspěchu) operace I/O. Je-li Y reg. menší než 128, t.j. není nastaven H-flag, všechno proběhlo vpořádku. Při Y≥128 nastala chyba. Chybové kody Y-reg. jsou stejné s chybovými hlášeními BASICu.

Podprogram provádějící výše uvedenou operaci OPEN:

```
OPEN    LDX #2*16        ;číslo IOCB krát 16
        LDA #3           ;kod příkazu OPEN
        STA ICCMD,X      ;uložit do IOCB
        LDA #FNAME&255   ;ukazatel na soubor, LSB
        STA ICBAL,X      ;uložit jako adresu buferu
        LDA #FNAME/256   ;ukazatel, MSB
        STA ICBAH,X
        LDA #8           ;soubor otevřít pro zápis
        STA ICAX1,X      ;uložit do AUX1
;                         X obsahuje č.IOCB krát 16
        JSR CIOV         ;CIOV = Ø E456
;                         Y obsahuje hlášení statusu
        RTS
        ...
FNAME   .BYTE "D:PRIKLAD.BIN", Ø9B  ;Ø9B je EOL
```

Stručný popis: Kód příkazu OPEN (Ø3) je uložen do IOCB, do pole pro adresu buferu je zapsán ukazatel na soubor a konečně je stanoven směr (následného) přenosu dat přes AUX1. Přípustné hodnoty pro AUX1 jsou:

4 : soubor je otevřen pro čtení
8 : soubor je otevřen pro zápis
6 : seznam diskety otevřen pro čtení
9 : Append:soubor otevřen pro zápis, data budou zapsána na konec existujícího souboru
12 : soubor otevřen současně pro čtení i zápis, pro soubory Random-Acces (s náhodným přístupem)

b) Zápis dat

Data můžeme zapsat do otevřeného souboru pomocí následujícího programu, který odpovídá příkazu PUT# v BASICu. Zapisovaná hodnota musí být v reg A.

```
PUT     PHA             ;úschova hodnoty
        LDX #2*16       ;čísloIOCB krát 16
        LDA #11         ;příkaz "PUT Charakters"
        STA ICCMD,X     ;zápis do IOCB
        LDA #Ø          ;zápis jen jedné hodnoty
        STA ICBLL,X     ;tomu odpovídá nulová délka
                         buferu
        STA ICBLH,X
        PLA             ;vyzvednutí hodnoty
        JSR CIOV        ; a zápis
        CPY #Ø          ;chyba?(viz text)
        BMI ERROR       ;ano, do rutiny chyb
```

Instrukci CPY #Ø můžeme vypustit, protože flagy jsou nastaveny podle reg. Y, takže postačí jen instrukce BMI. Příkaz "PUT Characters" byl použit jenom pro zápis jednoho byte. Předání dat zprostředkuje registr A.

c) Zápis bloku dat

Potřebujem např. zapsat rozsah paměti $8ØØØ až $9FFF (délka 8 kB, $2ØØØ byte), můžeme to zařídit následujícím programem (OPEN provést jako v bodě a)).

```
PUTBLK LDX #2*16        ;číslo IOCB krát 16
       LDA #11          ;příkaz "PUT Characters"
       STA ICMD,X       ;do IOCB
       LDA #Ø           ;počáteční adresa LSB
       STA ICBAL,X      ;adresa buferu LSB
       LDA #$8Ø         ;počáteční adresa MSB
       STA ICBAH,X      ;adresa buferu MSB
       LDA #Ø           ;délka datového souboru bloku
                         LSB
STA ICBLL,X             ;délka buferu MSB
       LDA #$2Ø         ;délka MSB($2ØØØ)
       STA ICBLH,X      ;délka buferu MSB
       JSR CIOV         ;uložit datový blok
       BMI ERROR        ;chyba --->
```

d) Čtení d a t

Tímto programem můžeme načíst datové bloky zapsané způsobem uvedeným v době c). Předem musíme samozřejmě zadat vhodný příkaz OPEN, který do AUX1 dosadí hodnotu 4 (otevřeno pro čtení),(ICAX1:= 4)

```
GETBLK LDX #2*16      ;číslo IOCB krát 16
       LDA #7         ;příkaz "GET Characters"
       STA ICCMD,X    ;do IOCB
       LDA #Ø         ;poč.adr. LSB
       STA ICBAL,X    ;adr. buferu LSB
       LDA #Ø8Ø       ;poč.adr. MSB
       STA ICBAH,X    ;adr. buferu MSB
       LDA #Ø         ;délka datového bloku LSB
       STA ICBLL,X    ;délka buferu LSB
       LDA #Ø2Ø       ;délka MSB
       STA ICBLH,X    ;délka bufru MSB
       JSB CIOV       ;čtení datového bloku
       BMI ERROR      ;chyba? ------>
       ....
```

e) Čtení jednotlivých b y t e

Příklad v bodě d) musíme modifikovat pro čtení jednotlivých byte

```
GET    LDX #2*16      ;číslo IOCB krát 16
       LDA #7         ;příkaz "GET Charakters"
       STA ICCMD,X    ;do IOCB
       LDA #Ø         ;délka buferu :=Ø
       STA ICBLL,X
       STA ICBLH,X
       JSR CIOV       ;čtení jednoho byte
                       byte je v reg. A
       BMI ERROR      ;chyba? ----->
       ...
```

Tato část programu odpovídá příkazu GET# v BASICu.

f) Uzavření souboru

Je to jednoduchá záležitost. Je třeba jenom zadat kod příkazu CLOSE do IOCB a zavolat CIO. Ostatní parametry nepotřebujeme zadávat.

```
CLOSE  LDX #2*16      ;číslo IOCB krát 16
       LDA #12        ;kód příkazu CLOSE
       STA ICCMD,X    ;do IOCB
       JSB CIOV       ;provedení CLOSE
       ...
```

Z á v ě r

Je třeba ještě poznamenat, že popsané metody přenosu dat pomocí CIO lze použít nejenom u disket. Místo specifikace souboru můžeme zadat zkrácené I/O zařízení, např. "C:" pro kazetu nebo "P:" pro tiskárnu. Můžeme tedy přenášet data na jiné zařízení. Důležité je zde poznamenat, že existují ještě dodatečné příkazy pro přenos dat (řetězcově orientované). Blíže se tím zabývat nebudeme, jenom řekneme, že se jedná o přenos tzv. RECORDS, jež jsou vždy zakončeny znakem EOL (Ø9B). Tyto metody jsou použity v příkazech BASICu INPUT a PRINT. Čísla kódů příslušných odkazů na CIO "GET/PUT RECORD" jsou uvedena v tabulce.

Tabulka důležitých příkazů CIO:

Příkaz	ICCMD	ICBAL/ ICBAH	ICBLL/ ICBLH	ICAX1	Návěstí příkazů
OPEN	3	ukazatel na	./.	Modus	COPN
CLOSE	12	soubor	./.		CLOSE
STATUS	13	ukazatel na soubor	./.		CSTAT
GET CHARACTERS	7	adresa buferu	délka buferu		CGBINR
PUT CHARACTERS	11	adresa buferu	délka buferu		CPBINR
GET RECORD	5	adresa buferu	max.délka buferu		CGTXTR
PUT RECORD	9	adresa buferu	max.délka buferu		CPTXTR

Důležitá hlášení o chybách

128 : stlačeno tlačítko BREAK
129 : žádané IOCB je obsazeno
130 : zařízení neexistuje
133 : soubor neotevřen (OPEN špatně zadán nebo chybí)
134 : neplatné číslo IOCB
136 : END-of-File, čtení až na konec souboru
137 : Record nevejde do buferu, zadat větší bufer
138 : Timeout operace trvala dlouho
139 : Device NAK, neproveditelný příkaz, např. Sektor Ø
144 : Device done, např. zničený sektor
162 : disketa je plná
164 : FMS - řetězení sektorů nesouhlasí
167 : Soubor chráněný pomocí DOSu možno odstranit
17Ø : soubor nenalezen

B L S - BASIC a binární soubory

Příkazy GET/PUT CHARACTERS pro rychlé psaní a čtení datových bloků nemůžeme bohužel využít přímo v BASICu. Rutina v assembleru BLS (binární čtení a psaní) se vám hodí, i když má určité omezení. S pomocí BLS můžete celé paměťové oblasti zapsat na disketu zadáním počáteční adresy a délky, ale také načíst z diskety. Je ideální, když můžete načíst z diskety do programu v BASICu Hi-Res obrázky, řetězce, data pro PM grafiku, strojové programy nebo je na disketu zaznamenat.

T r i k 1 7

Abychom získali co nejkratší strojový program, použijeme následující trik: Příkazy OPEN a CLOSE provádíme v BASICu a strojový program přejímá výlučně úlohu přenosu dat. Ušetříme přitom předávání názvu souboru do strojového programu a získáme kromě flexibility mezi OPEN, přenosem bloku dat a CLOSE ještě několik byte, takže můžeme říci, že čtení nebo psaní souboru "jde od ruky". Dále je tedy možné, abychom při ukládání Hi-Res obrázků pomocí příkazu PUT# uložili obsahy barvových registrů a naopak při načítání obrázků psali s GET# do příslušných barvových registrů.

BLS program je v následujících příkladech obsažen v řádcích DATA. Pro lepší pochopení je na konci kapitoly uveden výpis programu v asembleru.
BLS můžeme volat následovně:

X = USR(BLS,<flag>,<počáteční adresa>,<délka bloků>)

přičemž BLS-1536 je počáteční adresa strojového programu (stránka 6), flag musí být pro čtení nulový, pro psaní musí být nenulový, parametry, počáteční adresa a délka jsou dostatečně známy z předcházejícího článku. Před zavedením BLS, jak jsme se již zmínili, musí být správný příkaz OPEN,
OPEN #3,8,Ø,"D: ..." pro zápis souboru
OPEN #3,4,Ø,"D: ..." pro čtení.
Důležité přitom je vždy používat IOCB#3 (tedy vždy OPEN#3), protože strojový program je sestaven pro tento I/O kanál.

Načítání strojového programu v BASICu

Oddělení assemblerovského programu od BASICu se příznivě projeví také tehdy, když chceme načíst dovnitř strojové programy, které byly uloženy např. assemblerem nebo z DOSu (Binary Save). Soubory tohoto typu (tzv. soubory binárně zaváděné) obsahují totiž před datovými bloky ještě řídící informace: dva identifikační byte určující typ souboru (ØFF, ØFF), počáteční a koncovou adresou L/H v normálním formátu 6502. Oba identifikační byte, počáteční a koncovou adresu musíme získat pomocí příkazu GET z BASICu a správně přepočítat pro rutinu BLS. Následující program vám ukáže, jak se to dělá.

```
1ØØ REM *** Načtení strojového programu v BASICu ***
11Ø GOSUB 32ØØØ: REM inicializace strojového programu
12Ø OPEN#3,4,Ø,"D: FILENAME.EXT": REM Zde dosadit název souboru
13Ø GET #3,X1: GET #3,X2: REM Čtení identifikačních byte
14Ø IF X1<>255 OR X2<>255 THEN CLOSE#3: ?"Není binární
    soubor!": STOP
15Ø GET #3,AL: GET #3,AH: ANFADR=AL+AH*256: REMpočáteční adr.
16Ø GET #3,AL: GET #3,AH:ENADR=AL+AH*256: REM koncová adresa
17Ø LAENGE=ANADR+1: REM počet byte
18Ø X=USR(BLS,Ø,ANFANDR,LAENGE): REM načtení souboru
19Ø CLOSE #3
2ØØ IF X>128 THEN ? "Chyba:";X
29Ø END
32ØØØ REM Odbavení BLS
32Ø1Ø S=Ø: RESTORE 321ØØ
32Ø2Ø FOR A=1536 TO 1576: READ D: POKE A,D: S=S+D: NEXT A
32Ø3Ø IF S<>4119 THEN ?"Chyba v datech!", STOP
32Ø4Ø BLS=1536
32Ø9Ø RETURN
321ØØ DATA 1Ø4,162,48,1Ø4,16Ø,7,1Ø4,24Ø,2,16Ø,11,152,157,
      66,3,1Ø4
3211Ø DATA 157,69,3,1Ø4,157,68,3,1Ø4,157,73,3,1Ø4,157,72,
      3,32,86,228
3212Ø DATA 132,212,169,Ø,133,213,96
```

Není žádným problémem pomocí BLS rutíny načíst nabo zapsat HI-Res obrázek (např. GR.8). Potřebujete k tomu jen po-

čáteční adresu obrazové paměti (najdete ji v 88/89 dec L/H) a délku obrázku GR.8 (4Ø byte nařádek krát 192 řádků =768Ø B). Dále je uveden program, pomocí kterého můžete zapsat demo - obrázek

```
1ØØ REM *** DEMO: GR.8 - zápis obrázku ***
11Ø GOSUB 32ØØØ: REM strojový program
115 GRAPHICS 8+16: REM GR.8 bez textového okénka
12Ø SETCOLOR 2,Ø,Ø: RE? pozadí černé
125 COLOR 1: PLOT 159,95: REM GR.8 vytvoření obrázku
13Ø FOR A=Ø TO 5Ø: DRAWTO 319*RND(1): NEXT A
135 SCREEN=PEEK(88)+PEEK(89)*256: REM obrazová adresa
14Ø LAENGE=768Ø: REM délka GR.8/7+obraz
15Ø OPEN #3,8,Ø"D:TEST.PIC": REM dosadit název souboru
16Ø X=USR(BLS,1,SCREEN,LAENGE)
17Ø CLOSE #3
18Ø IF X>128 THEN ? "Chyba:"; X
19Ø END
32ØØØ REM Odbavení BLS
32Ø1Ø S=Ø: RESTORE 321ØØ
32Ø2Ø FOR A=1536 TO A=1576: READ A: POKE A,D: S=S+D:NEXT A
32Ø3Ø IF S<> 4119 THEN ?"Chyba v datech"
32Ø4Ø BLS=1536
32Ø9Ø RETURN
321ØØ DATA 1Ø4,162,48,1Ø4,16Ø,7,1Ø4,24Ø,2,16Ø,11,152,157,
      66,3,1Ø4
3211Ø DATA 157,69,3,1Ø4,157,68,3,1Ø4,157,73,3,1Ø4,157,72,3,
      32,86,228
3212Ø DATA 132,212,169,Ø,133,213,96
```

Pomocí dalšího programu můžeme znovu načíst zapsaný Hi-Res obrázek.

```
1ØØ REM *** Načtení obrázku GR.8 ***
11Ø GOSUB 32ØØØ: REM strojový program
12Ø GR.8+16: REM celá obrazovka GR.8
13Ø SETCOLOR 2,Ø,Ø: REM pozadí černé
135 SCREEN=PEEK(88)+PEEK(89)*256: REM adresa obrazu
14Ø LAENGE=768Ø: REM délka GR.8/7+obraz
15Ø OPEN #3,4,Ø,"D:TEST.PIC": REM dosadit název souboru
16Ø X=USR (BLS,Ø,SCREEN,LAENGE)
```

```
17Ø CLOSE #3
18Ø IF X>128 THEN ? "Chyba:"; X
19Ø GOTO 19Ø: REM aby se nezničila grafika
32ØØØ REM Odbavení BLS
32Ø1Ø S=Ø: RESTORE 321ØØ
32Ø2Ø FOR A=1536 TO 1576: READ D: POKE A,D: S=S+D:NEXT A
32Ø3Ø IF S<>4119 THEN ? "Chyba v datech": STOP
32Ø4Ø BLS=1536
32Ø9Ø RETURN
321ØØ DATA 1Ø4,162,48,1Ø4,16Ø,7,1Ø4,24Ø,2,16Ø,11,152,157,
      66,3,1Ø4
3211Ø DATA 157,69,3,1Ø4,157,68,3,1Ø4,157,73,3,1Ø4,157,72,3,
      32,86,228
3212Ø DATA  132,212,169,Ø,133,213,96
```

Dále najdete komentovaný výpis BLS programu v assembleru Stručný popis tohoto programu.

```
18Ø      číslo použitého IOCB*16
19Ø-2ØØ  kódy pro příkazy GET/PUT CHARACTERS
22Ø-26Ø  definice parametrů IOCB.Příkaz, adresa buferu, dél-
         ka buferu
28Ø      vektor skoku do IOCB
39Ø      program je uložen na stránce 6
4ØØ      počet USR argumentů pro převzetí ze sklípku
41Ø      do X-registru načíst IOCB (offset)
42Ø-5ØØ  Flag=Ø -> GET CHARACTERS, příkaz zapsat do IOCB
         flag=1 -> PUT CHARACTERS
51Ø-54Ø  adresu buferu ze sklípku zapsat do IOCB
6ØØ      volání CIO, status bude předán do Y registru
62Ø-640  předání statusu do výsledku USR
```

```
                Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø11Ø ; BLS-čtení/zápis binárních souborů v BASICu
                Ø12Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø14Ø ;
                Ø15Ø ;
                Ø16Ø ; definice CIO
                Ø17Ø ;
=ØØ3Ø           Ø18Ø IOCB3  =   $3Ø     IOCB číslo 3
=ØØØ7           Ø19Ø CGBINR =   $Ø7     příkaz CIO: čtení znaku
=ØØØB           Ø2ØØ CPBINR =   $ØB     příkaz CIO: zápis znaku
                Ø21Ø ;
=Ø342           Ø22Ø ICCMD  =   $Ø342   zde je uložen příkaz CIO
=Ø344           Ø23Ø ICBAL  =   $Ø344   adresa buferu nižší byte
=Ø345           Ø24Ø ICBLL  =   $Ø345   vyšší byte
=Ø348           Ø25Ø ICBLL  =   $Ø348   délka buferu nižší byte
=Ø349           Ø26Ø ICBLH  =   $Ø349   vyšší byte
                Ø27Ø ;
=E456           Ø28Ø CIOV   =   $E456   skok do CIO
                Ø29Ø ;
=ØØD4           Ø3ØØ FRO    =   $D4     výsledek do příkazu USR
                Ø32Ø ;
                Ø33Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø34Ø ;USR rutina: LOAD/SAVE binární soubor
                Ø35Ø ;volání: X=USR(1526,flag,start,adresa,délka
                Ø36Ø ;flag: Ø: = LOAD,1: =SAVE
                Ø37Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø38Ø ;
ØØØØ            Ø39Ø        *=  $Ø6ØØ   na stránku 6(kam jinam?!)
Ø6ØØ 68         Ø4ØØ        PLA         počet argumentů
Ø6Ø1 A23Ø       Ø41Ø        LDX #IOCB3  bude používáno IOCB3
Ø6Ø3 69         Ø42Ø        PLA         čtení/zápis vyššího byte flagu
Ø6Ø4 AØØ7       Ø43Ø        LDY  $CGBINR předběžné obsazení pro
                                          čtení značek
Ø6Ø6 68         Ø44Ø        PLA         flag: čtení=Ø, zápis=1
Ø6Ø7            Ø45Ø        BEQ BCMD    čtení?
Ø6Ø9 AØØ8       Ø47Ø        LDY $CPBINR ne, zápis
Ø6ØB 98         Ø49Ø BCMD   TYA         příkaz do akumulátoru
Ø6ØC 9D42Ø3     Ø5ØØ        STA ICCMD,X příkaz do IOCB
Ø6ØF 68         Ø51Ø        PLA         počáteční adresa,vyšší byte
Ø61Ø 9D45Ø3     Ø52Ø        STA ICBAH,X uložit do adr. buferu
```

```
Ø613 68       Ø53Ø        PLA          nižší byte
Ø614 9D44Ø3   Ø54Ø        STA ICBAL,X
Ø617 68       Ø55Ø        PLA          délka buferu,vyšší byte
Ø618 9D49Ø3   Ø56Ø        STA ICBLH,X
Ø61B 68       Ø57Ø        PLA          nižší byte
Ø61C 9D48Ø3   Ø58Ø        STA ICBLL,X
Ø61F 2Ø56E4   Ø59Ø        JSR CIOV     volání centrální I/O ru-
                                       tiny>>>
              Ø6ØØ ;
Ø622 84D4     Ø62Ø        STY FRO      status do USR
Ø624 A9ØØ     Ø63Ø        LDA #Ø
Ø626 85D5     Ø64Ø        STA FRO+1
Ø628 6Ø       Ø65Ø        RTS
```

Nové příkazy pro DOS XL

DOS XL od Optimized System Software nemůže zapřít příbuznost s CP/M, populárním operačním systémem pro mikroprocesory (Z-8Ø). A právě to ho dělá univerzálním a výkonným nástrojem pro programátora. Zadávání je povelově orientované, což se zdá začátečníkovi těžší než výběr z menu (ATARI -DOS). Pro pokročilejší uživatele to ovšem má své výhody, např. možnost zpracování Batch (popisováno na jiném místě). Struktura DOS XL je otevřena pro rozšiřování a seznam příkazů lze snadno uživatelem zvětšovat. Přitom je možné DOS XL přizpůsobovat podle vlastní potřeby.

Typy příkazů DOS XL

Rozlišujeme dva typy příkazů v DOS XL.: jedny jsou "vlastní příkazy", které jsou pevnou součástí DOSu a proto jsou pokaždé volatelné (např. DIR,LOAD, SAVE).
Druhé jsou tzv. "vnější příkazy", které nejsou k dispozici v paměti a při použití musí být nataženy z diskety. Do této druhé skupiny řadíme např. povely COPY, DUPDSK a INIT.

Příklad I N I T

Co se děje při volání tzv "vnějšího příkazu, např. příkazu INIT?".
Když vypíšeme INIT, DOS nejprve zjišťuje ve svém seznamu příkazů, jednáli se o jeho vnitřní příkaz. Hledání potom pokračuje v seznamu diskety a nakonec (pokud je založena správná disketa) najde soubor INIT.COM. Tento soubor bude načten a pokud nemá vlastní volací adresu, odstartuje se od počátku programu. Teď jsme dosáhli bodu, kdy se na obrazovce objeví menu příkazu INIT.
Vidíme, že rozšíření zásoby příkazu DOSu XL je jednoduché. Potřebujeme jenom strojový program, který je uložen na disketě pod požadovaným jménem příkazu s příponou .COM a už je součástí DOS XL.

Předávání paremetrů

DOS XL vám poskytuje elegantní možnost: Můžete vyzvednout parametr a přídavnou informaci přímo ze zadáváacíhe řádku, ve kterém byl příkaz zavolán. Příkaz COPY uvedeme jako příklad. Chceme-li kopírovat soubor, zadáme následující příkaz.

D1: COPY D1: FILE1.TST D1: FILE2.TST<return>

Začátek probíhá stejně, jak už bylo uvedeno v příkazu INIT. COPY taky nepatří k vnitřním příkazům a proto bude muset být načten z diskety soubor COPY.COM. Nyní si COPY vyzvedne svoje paraametry (oba soubory) přímo ze zadávacího řádku. Stejný druh předávání parametrů můžete použít také ve vlastních programech. Představte si, že máte napsaný program Hard copy (např. HARDCOPY.COM), který umí vytisknout na tiskárně obrazový soubor. Metodou popsanou v této kapitole můžete programu sdělit, který soubor má vytisknout:

D1: HARDCOPY D1: BILD1.PIC

Uvědomujete si, jaké zajímavé možnosti se přitom nabízí?

Příkazový bufer

DOS XL uloží kompletní příkazový řádek (včetně příkazových slov) do svého příkazového buferu a nastaví si ukazatel buferu tam, kde je dokončeno zpracování příkazu pro DOS XL, konkrétně na konec příkazového slova.

<CPALOC>+CPCMDB (Ø3F) ↵:= EOL (Ø9B)

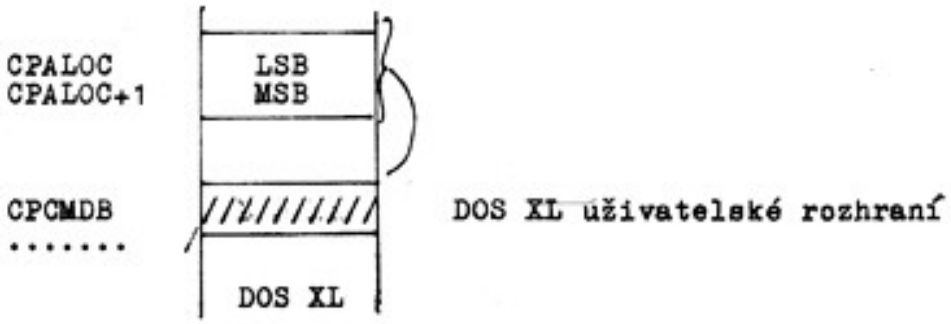
```
D1:COPY D1: FILE1.TST D1: FILE2.TST
```

ukazatel buferu <CPALOC>+CPBUFP (ØØA)

Výraz pro počáteční adresu buferu vypadá dost komplikovaně, má to ale podstatný důvod, kompatibilitu. Vaše programy nemusí běžet jen se současnou verzí DOS XL, ale taky se staršími nebo novějšími verzemi. Programátoři DOS XL na to pamatovali a předvídají univerzální rozhraní pro uživatele. Všechny proměnné, volání podprogramů a adresy buferů jsou vztaženy relativně k počáteční adrese DOSu XL, kterou najdete v paměťovém místě CPALOC a CPALOC+1 (ØØA/ØB L/H). Uživatelé ATARI-DOSu je znají spíše pod označením DOSVEC.

Poznámka

Dále použitá jména proměnných a označení konstant odpovídají použitým návěstím v souboru SYSEQU.ASM. Tento soubor najdete na systémové disketě DOS XL. Ušetříte si psaní u programů v assembleru, jestliže tento soubor pomocí INCLUDE připojíte ke svému programu. Ve svých programech máte potom zahrnuty všechny důležité proměnné a konstanty, včetně IOCB konstant.



Adresu příkazového buferu můžeme určit, když k základní adrese ve vektoru CPALOC přičtete odchylku buferu (CPCMDB, Ø3F). Výsledek bude uložen do nulové stránky, odkud ho můžete použít jako nepřímou základní adresu pro čtení buferu.:

```
REGIL      = ØCE   pomocný registr v nulté stránce (LSB)
REG1H      = #CF   MSB

...
CLC                přípravа sečítání
LDA CPALOC         LSB DOS - počáteční adresa
ADC #CPCMDB        ODCHYLKA PŘÍKAZOVÉHO BUFERU
STA REG1L          do pomocného registru
LDA CPALOC+1       MSB DOS - začátek
ADC #Ø             případný přenos
STA REG1H          do MSB pomocného registru
....
```

Teď potřebujeme ještě jednou hodnotu ukazatele buferu, kterou dostaneme z odchylky od CPBUFP (ØØA).

```
LDY #CPUFP         odchylka ukazatele buferu
LDA (CPALOC),Y     hodnota ukazatele buferu
TAY        použít jako nový index
```

Pomocí základní adresy v REG1 L/H a indexu v Y registru, můžeme číst příkazový bufer znak po znaku:

```
LDA (CPALOC),Y      znak z příkazového buferu
INV                 další znak
...
```

Více komfortu

Pokud se jedná o parametry, jde nejčastěji o názvy souborů a jejich specifikace. DOS XL bere ohled na tento stav a dává programátorovi k dispozici podprogram "GET FILENAME" pomocí kterého může převzít soubor z příkazového buferu, začínajícího od aktuální hodnoty ukazatele buferu do připraveného buferu - souboru. Přitom je komfortnější chybějící název zařízení doplnit aktuálním drivrem (nejčastěji D1:) a soubor ukončit pomocí EOL(Ø9B). Takto upravené specifikace souboru můžeme použít přímo v příkazu OPEN. Současně bude zaveden ukazatel buferu, takže další volání "GET FILENAME" přenese další soubor.

Adresa skoku (CPGNFN, ØØ3) podprogramu je, stejně jako jiné adresy rozhraní, myšlena jako relativní k začátku DOSu. Bohužel 65Ø2 nemá žádné nepřímé volání podprogramu, takže si musíme pomoci jinak. Vektor skoku bude, jako v předešlém příkladu znovu určen pomocí sečítání vektoru CPALOC s odchylkou GET FILENAME a potom jako adresa uložena do instrukce JMP (jde o samomodifikující se program!).

```
GETARD  CLC                 volání podprogramu
        LDA CPALOC          LSB začátek DOSu
        ADC #CPGNFN         odchylka adresy skoku
        STA GETFN+1         uložit do JMP
        LDA CPALOC+1        MSB DOS- začátek
        ADC #Ø              případný přenos
        STA GETFN + 2       MSB do instrukce JMP
       RTS                  hotovo !

;
GETFN   JMP Ø               adresa bude definována
```

Pokaždé, když zavoláme uvedený podprogram GETARD, přinese každý JSR GETFN specifikaci vektoru z příkazového buferu do buferu názvu souboru. Průběh programu vede přitom přes definovanou instrukci JMP ke skutečné adrese skoku, RTS se nachází v podprogramu GET FILENAME v DOSu XL. Chybí ještě možnost přezkoušet, je-li vůbec název soubo-

ru k dispozici. Je to možné udělat pomocí ukazatele buferu, musíme jen hlídat jeho hodnotu a po GETFN Ověřit, zda se změnila. Následující program ukazuje, jak to můžeme naprogramovat:

```
GETNAM LDY #CPBUFP        odchylka ukazatele buferu
       LDA (CPALOC),Y     hodnota ukazatele buferu
       PHA                prozatím hlídat
       JSR GETFN          vyzvednout soubor
       PLA                vyzvednout starý ukazatel buferu
       CMP (CPALOC),Y     = nová hodnota ?
       BEQ KEINFN         ano nebyl žádný soubor ---->
       .....
```

Pokud volání GETFN bylo úspěšné, tzn. ukazatel buferu se změnil, potom můžeme vyzvednout kompletní soubor z buferu názvem souboru (Odchylka od DOS - úvodu: CPFNAM = $21).

Příklad :

```
       LDY #CPFNAM        odchylka buferu názvu souboru
       LDX #Ø             čítač bite nastavit na Ø
NXTCHR LDA (CPALOC),Y     vyzvednout znak z buferu názvu
                          souboru
       STA FNAME,X        uložit do vlastního buferu
       CMP #EOL           už hotovo? (EOL = $9B)
       BEQ FERTIG         ano --->
       INY                ukazatel na další znak
       INX                čítač byte
       CPX #16            už 17-tý znak?
       BNE NXTCHR         ne, další znak
       JMP ERROR          není EOL
FERTIG ...
;
FNAME *=*+16              16-ti bytový bufer pro název
                          souboru
```

Příklad použití této techniky je uveden v další kapitole pro program APPEND.

Kam s přídavnými příkazy do paměti?

Jednoduchou možnostíposkytuje jako vždy strana 6. Bohužel tam můžeme uložit jenom programy do délky 256 byte a to je pro mnohé aplikace málo. V dalších případech se můžeme spoléhat na paměťový rozsah nad DOS XL, což přináší tyto nevýhody:

1. Horní hranice paměti využívaná DOSem XL se může pohybovat a závisí na tom, zda se použije menu DOSu, nebo zda místo DOS XL nasadíme DOS HK (viz přílohu). Má-li být váš COM soubor přenositelný mezi různými verzemi DOSu, nesmí být umístěn vpředu v paměti.
2. V tomto rozsahu paměti se mohou nacházet programy, které lze zničit voláním příkazu DOS. Ujistěte se o tom, zda se zde nenachází něco podstatného dříve, než zavoláte "vnější" příkaz DOSu.

Startovací adresa

Soubor COM bude DOSem XL startován vlastním podprogramem, podprogramovým skokem na svou fyzickou počáteční adresu.Můžete si určit jinou adresu skoku, toho dosáhnete, když přidáte do svého programu ještě RUN - adresu. Napište, nejlépe na konec svého programu tuto sekvenci:

* = $02E0	zde bude uložena RUN-adresa
. WORD START	START je adresa skoku vašeho programu

Když pak bude program ukončen instrukcí RTS, vrátí se řízení zpět do DOSU XL.

Příkaz APPEND pro dos XL

V této kapitole se dočtete, jak lze používat v předchozí kapitole popsaná rozhraní DOS XL. Mimo to dostanete do rukou velmi potřebný příkaz, pomocí kterého můžete spojovat dohromady binární soubory. Je to velmi užitečné, když např. dlouhé strojové programy rozepisujete do více částí (čtěte také kapitoly programování v assembleru, segmentování) a potom jednotlivé části chcete spojit do jednoho celku.Nebo chcete navzájem navázat ke strojovému programu řetězec ...

Viděli jste program, který během načítání zobrazí na obrazovce úvodní obraz(nápis)? Taky toto může zařídit APPEND.

Co se za tím skrývá ?

Z odstavce "BLS" již znáte skladbu binárního souboru. Dva poznávací byte ($FF), počáteční adresa a koncová adresa ve tvaru L/H. Vedle toho existují komplikovanější skládané binární soubory, které se skládají z více datových bloků, a proto se též nazývají "COMPOUND soubory".

Struktura souboru COMPOUND		
	$FF $FF	poznávací byte
	ANFADR LSB ANFADR MSB	počáteční adresa I
	ENDADR LSB ENDADR MSB	koncová adresa I
	datový blok	
	ANFADR LSB ANFADR MSB	počáteční adresa II
	ENDADR LSB ENDADR MSB	koncová adresa II
	datový blok	
	...	...

Vidíte rozdíl oproti "normálnímu" binárnímu souboru. Po prvním datovém bloku následuje další počáteční adresa, koncová adresa a přídavný datový blok. Poznávací byte jsou vypouštěny, ale nevydilo by, kdyby tam byly. Tato struktura se může opakovat, konec takového souboru bude zjištěn při přečtení posledního byte pomocí status EOF. Pomocí APPENND můžeme ze dvou binárních souborů vytvořit Compound soubor nebo též stávající Compound soubor rozšířit.

Vyvolání APPENDu

APPEND můžeme použít jako jiné příkazy DOS XL, předpokladem je, že máte APPEND.COM na disketě. Ostatní si jej mohou napsat pomocí např. Editassembleru (nebo EASMD,MAC/65). COM soubor dostanete pomocí asemblování jednoduše na disketu: ASM,,#D:APPEND.COM. Nezapomeňte na čárky.

Vyvolání příkazu APPEND je následující:

APPEND <připojovaný soubor> <cílový soubor>

Příklad:

Soubor NEU.OBJ chceme připojit k souboru ALT.OBJ.

Zadáme: D1:APPEND D1: NEU.OBJ D1: ALT.OBJ

Výsledkem činnosti APPEND je soubor ALT.OBJ, který obsahuje soubor NEU.OBJ. NEU.OBJ zůstane nezměněn na disketě. D1: před oběma soubory není nutné zadávat.

POZOR:
Uvědomte si, že kopie cílového souboru po APPENDu přepíše původní soubor !

Jak to funguje ?

Ve strojovém programu APPEND se nejprve zjistí dva názvy programů z příkazového buferu DOSu XL. Metodu najdete v kapitole "Nové příkazy DOS XL". Název zdrojového souboru (připojovaný soubor, v budoucnosti SRC soubor) je v FNAME1 potom a cílový soubor (za který připojujeme, DST soubor) je v FNAME. Potom začne následující průběh:

1. Otevření SRC souboru
2. Překopírování názvu DST souboru do FNAME1, přitom podprogram (UP) můžeme znovu použít pro OPEN.
3. Otevřeme SRC soubor pomocí APPEND pro zápis, tzn. všechna zapsaná data připojíme za soubor.
4. Zkoušíme hlavičku souboru SRC: mají oba soubory na začátku $FF? Pokud ne, není binární soubor.
5. Začíná přenos dat (UPTRANS): Nejprve bude načten SRC soubor do buferu, který začíná od $3000 až do konce paměti (MEMTOP-256). Mohou nastat dva případy:

a) Soubor se vejde do buferu, potom bude ENDFLG "správně" nastaven.
b) Soubor je větší než bufer potom zůstane ENDFLG nastaven "falešně".

6. Čtený rozsah buferu bude zapsán do DST souboru. Počet načtených byte vyzvedneme ze SRC IOCB.
7. Když je teď ENDFLG "správně" nastaven, je činnost ukončena, protože SRC soubor byl celý přečten. Je-li ENDFLG nastaven "falešný", bude se dále pokračovat bodem 5.
8. SRC a DST soubory se uzavřou.

Pokud během této procedury nastane jakákoliv chyba, program sepřeruší a program přejde na rutinu ERROR. Tam se uzavřou všechny soubory, bude přerušen běžící provoz BATCH a objeví se na obrazovce hlášení xxx Chyba xxx.

Triky s EPPEND

S APPEND můžeme udělat strojový program (zaváděč), který lze provozovat během načítání programu. Zaváděcí program musí znát INIT adresu, která je buď zavedena pomocí ATARI DOS (Binary SAVE) nebo prostřednictvím instrukcí:

```
*  =  $2E2          INIT adresa
.WORD START         začátek programu
```

na konci vašeho programu. Teď APPENDem připojíme strojový program za zaváděč (tato posloupnost je důležitá!), např.:

```
D1: APPEND SPIEL.COM VORSPAN.COM
```

a získáme tak program se zaváděčem. V uvedeném příkladu je soubor VORSPAN.COM konečným produktem, který obsahuje SPIEL.COM. Použití zaváděcího programu je obzvlášť vhodné při dlouhých programech, protože uživatel se nemá co dívat, pokud se program nahrává. Touto metodou se může např. nahrát přípravný program, který po splnění svého úkolu bude přepsán hlavním programem. APPEND můžete přirozeně použít též uvnitř Execute souboru, tímto způsobem lze elegantně spojit dohromady mnoho jednotlivých částí programu do jednoho hotového souboru.

```
        Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        Ø11Ø ;APPEND - příkaz pro DOS XL              x
        Ø12Ø ;Volání:APPEND D: SRC.EXT D: DST.EXT     x
        Ø13Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        Ø14Ø ;
        Ø15Ø ;SRC := připojovaný soubor
        Ø16Ø ;DST := soubor, na který připojujeme
        Ø17Ø ;
        Ø18Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        Ø19Ø ; Proměnné operačního systému
        Ø2ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        Ø21Ø ;
=Ø2E5   Ø22Ø MEMTOP =  $Ø2E5   horní hranice paměti
=E456   Ø23Ø CIOV   =  $E456   vektor centr.rutiny I/O
        Ø24Ø ;
=Ø342   Ø25Ø ICCMD  =  $Ø342   definice IOCB
=Ø343   Ø26Ø ICSTA  =  $Ø343
=Ø344   Ø27Ø ICBAL  =  $Ø344
=Ø345   Ø28Ø ICBAH  =  $Ø345
=Ø348   Ø29Ø ICBLL  =  $Ø348
=Ø349   Ø3ØØ ICBLH  =  $Ø349
=Ø34A   Ø31Ø ICAX1  =  $Ø34A
=Ø34B   Ø32Ø ICAX2  =  $Ø34B
        Ø33Ø ;
        Ø34Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        Ø35Ø ;Konstanty
        Ø360 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        Ø37Ø ;
=ØØØ3   Ø38Ø COPN   =  3       příkaz OPEN
=ØØØ4   Ø39Ø OPIN   =  4       AUX1-kód pro OPEN INPUT
=ØØØ9   Ø4ØØ OPAPND =  9       AUX1-kód pro OPEN APPEND
=ØØØ7   Ø41Ø CGBINR =  7       příkaz GET CHARACTERS
=ØØØB   Ø42Ø        =  $ØB     příkaz PUT CHARACTERS
=ØØØ9   Ø43Ø CPTXTR =  $Ø9     příkaz PUT RECORD
=ØØØC   Ø44Ø CCLOSE =  $ØC     příkaz CLOSE
        Ø45Ø ;
=ØØØØ   Ø46Ø IOCBØ  =  Ø       IOCB#Ø pro výstup textu
=ØØ3Ø   Ø47Ø IOCB3  =  $3Ø     IOCB#3 načtení souboru
=ØØ4Ø   Ø48Ø IOCB4  =  $4Ø     IOCB#4 Append soubor
```

```
                Ø49Ø ;
   =ØØØA        Ø5ØØ CPALOC = $ØA      vektor začátku DOS XL
   =ØØØ3        Ø51Ø CPGNFN = $Ø3      odchylka GET FILENAME rut
   =ØØ21        Ø52Ø CPFNAM = $21      odchyl.buferu jména soub
   =ØØØA        Ø53Ø CPBUFP = $ØA      odchyl.ukazat. buferu
   =ØØØB        Ø54Ø CPEXFL = $ØB      odchylka Execute flagu
                Ø55Ø ;
   =ØØ9B        Ø56Ø EOL    = $9B      značka konce řádku
   =ØØ88        Ø57Ø EOF    = $88      hlášení stavu konce soub.
   =ØØFF        Ø58Ø LO     = $FF      maska pro LSB
   =Ø1ØØ        Ø59Ø HI     = $Ø1ØØ    dělitel pro MSB
                Ø6ØØ ;
                Ø61Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø62Ø ;Použité proměnné a rozsahy paměti
                Ø63Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø64Ø ;
   =3ØØØ        Ø65Ø BUFFER = $3ØØØ    bufer pro připojovaný
                                       soubor
   =ØØCC        Ø66Ø ENDFLG = $ØØCC    flag pro konec soub($FF)
                Ø67Ø ;
                Ø68Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø69Ø ; Hlavní program
                Ø7ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø71Ø ;
ØØØØ            Ø72Ø        *= $2EØØ
                Ø73Ø ;
2EØØ 2Ø6Ø2E     Ø74Ø        JSR GETADR GET FN sestavení vektoru
2EØ3 2Ø732E     Ø75Ø        JSR GETNAM  obstarat SRC-název soub.
2EØ6 3Ø2B       Ø76Ø        BMI ERROR   chyba --->
                Ø77Ø ;
2EØ8 2Ø4B2F     Ø78Ø        JSR AMOVE   v buferu FNAME1 utvořit
2EØB 2Ø732E     Ø79Ø        JSR GETNAM  obstarat DST název soub.
2EØE 3Ø23       Ø8ØØ        BMI ERROR   chyba --->
                Ø81Ø ;
2E1Ø A23Ø       Ø82Ø        LDX #IOCB3  soubor  3(SRC)
2E12 A9Ø4       Ø83Ø        LDA #OPIN   otevřít pro čtení
2E14 2Ø9B2E     Ø84Ø        JSR AOPEN   provest OPEN
2E17 3Ø1A       Ø85Ø        BMI ERROR   chyba --->
                Ø86Ø ;
2E19 2Ø4B2F     Ø87Ø        JSR AMOVE   název soub.DST do buferu
```

```
2E1C A24Ø     Ø88Ø        LDX #IOCB4  otevřít IOCB#4 (DST)
2E1E A9Ø9     Ø89Ø        LDA #OPAPND soubor bude prodloužený
2E2Ø 2Ø9B2E   Ø9ØØ        JSR AOPEN
2E23 3ØØE     Ø91Ø        BMI ERROR   chyba? --->
              Ø92Ø ;
2E25 2ØB62E   Ø93Ø        JSR HREAD   číst/zkoušet hlavičku
                                      souboru SRC
2E28 3ØØ9     Ø94Ø        BMI ERROR   chyba? --->
              Ø95Ø ;
2E2A 2ØDD2E   Ø96Ø        JSR TRANS   provést přenos dat
2E2D 3ØØ4     Ø97Ø        BMI ERROR   došlo při tom k chybě?
              Ø98Ø ;
2E2F 2Ø362F   Ø99Ø        JSR ACLOSE  uzavřít IOCB #3/4
2E32 6Ø       1ØØØ        RTS         hotovo !
              1Ø1Ø ;
              1Ø2Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              1Ø3Ø ;Rutina pro ošetření chyb
              1Ø4Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              1Ø5Ø ;
2E33 2Ø572F   1Ø6Ø ERROR  JSR APRERR  hlášení
2E36 2Ø362F   1Ø7Ø        JSR ACLOSE  uzavřít všech.soubory
2E39 A9ØØ     1Ø8Ø        LDA #Ø      ukončit případné
2E3B AØØB     1Ø9Ø        LDY #CPEXFL provedení
2E3D 91ØA     11ØØ        STA (CPALOC),Y .EXC souborů
2E3F 6Ø       111Ø        RTS
              112Ø ;
              113Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              114Ø ; Rozsah buferu pro názvy souborů
              115Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              116Ø ;
2E4Ø          117Ø FNAME  *=  *+Ø1Ø   mezirozsah paměti
2E5Ø          118Ø FNAME1 *=  *+1Ø    bufer názvu souboru
                                      pro OPEN
              119Ø ;
              12ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              121Ø ; Sestavení vektoru pro rutinu GET FILENAME
              122Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              123Ø ;
2E6Ø 18       124Ø GETADR CLC
2E61 A5ØA     125Ø        LDA CPALOC  DOS-počat.adr. (LSB)
```

```
2E63 69Ø3      126Ø        ADC #CPGNFN odchylka rutiny GET
                                       FILENAME
2E65 BD712E    127Ø        STA GETFN+1 sečti a ulož do instr.
                                       JMP
2E68 A5ØB      128Ø        LDA CPALOC+1  MSB
2E6A 69ØØ      129Ø        ADC #Ø      případný přenos
2E6C 6Ø        13ØØ        STA GETFN+2
2E6F 6Ø        131Ø        RTS
               132Ø ;
2E7Ø FFØØØØ    133Ø GETFN  JMP Ø       vektor bude zapsán
               134Ø ;
               135Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               136Ø ;Obstarat názvy souboru z DOS XL
               137Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
               138Ø ;
2E73 AØØA      139Ø GETNAM LDY #CBUFP  odchylka pro ukazatel
                                       buferu
2E75 B1ØA      14ØØ        LDA (CPALOC),Y  načíst ukazatel
                                           buferu
2E77 48        141Ø        PHA         prozatím hlídat
2E78 2Ø7Ø2E    143Ø        JSR GETFN   název souboru
2E7B AØØA      145Ø        LDY #CBUFP  odchylka ukazat.buferu
2E7D 68        146Ø        PLA         stará hodnota
2E7E D1ØA      147Ø        CMP (CPALOC),Y  nová hodnota stejná?
2E8Ø FØ13      148Ø        BEQ GFERR   ano,není více FN --->
               149Ø ;
               15ØØ ; FN kopírovat do mezibuferu
2E82 AØ21      152Ø        LDY #CPFNAM odchylka buferu názvu
                                       souboru
2E84 A2ØØ      153Ø        LDX #Ø      čítač :=Ø
               154Ø ;
2E86 B1ØA      155Ø GFNXT  LDA (CPALOC),Y vyzvednout znak
2E88 9D4Ø2E    156Ø        STA FNAME,X uložit do mezibuferu
2E8B C99B      157Ø        CMP #EOL    konec FN ?
2E8D FØØ9      158Ø        BEQ GFEND   ano, hotovo --->
               159Ø ;
2E8F C8        16ØØ        INY         inkrement ukazatele
2E9Ø E8        1610        INX
2E91 EØ1Ø      162Ø        CPX #16     více než 16 znaků ?
2E93 DØF1      163Ø        BNE GFNXT   ne, dále --->
               164Ø ;
```

```
2E95 A9FF     1650 GFERR  LDA #$FF     není EOL v buferu !!
2E97 60       1660        RTS          nesmí se to stát !!
              1670 ;
2E98 A900     1680 GFEND  LDA #0       všechno v pořádku
2E9A 60       1690        RTS
              1700 ;
              1710 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              1720 ;Otevři soubor, FN do buferu FNAME1
              1730 ;<X>:číslo IOCB,<A>:kód AUX1
              1740 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              1750 ;
2E9B 9D4A03   1760 AOPEN  STA ICAX1,X uložit hodnoru AUX1
2E9E A903     1770        LDA #COPN    povel OPEN
2EA0 9D4203   1780        STA ICCMD,X do IOCB
2EA3 A950     1790        LDA #FNAME1&LO ukazatel na název
                                         souboru
2EA5 9D4403   1800        STA ICBAL,X
2EAB A92E     1810        LDA #FNAME1/HI
2EAA 9D4503   1820        STA ICBAH,X  MSB
2EAD A900     1830       LDA #0
2EAF 9D4B03   1840        STA ICAX2,X  AUX2 :=0
2EB2 2056E4   1850        JSR CIOV     proveď OPEN
2EB5 60       1860        RTS
              1870 ;
              1880 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              1890 ;čtení a test hlavičky souboru SRC
              1900 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              1910 ;
2EB6 A230     1920 HREAD  LDX #IOCB3   SRC soubor IOCB 3
2EB8 A907     1930        LDA #CGBINR příkaz GET CHARACTER
2EBA 9D4203   1940        STA ICCMD,X
2EBD A900     1950        LDA #0       délku buferu nastavit
                                       na nulu
2EBF 9D4803   1960        STA ICBLL,X pro čtení jedntl. byte
2EC2 9D4903   1970        STA ICBLH,X
2EC5 2056E4   1980        JSR CIOV     načtení prvního byte
2EC8 3010     1990        BMI HRERR    chyba při čtení?
              2000 ;
2ECA C9FF     2010        CMP #$FF     hlavička binár.souboru?
2ECC D00C     2020        BNE HRERR    ne --->
              2030 ;
```

```
2ECE 2Ø56E4   2Ø4Ø        JSR CIOV     načtení druhého byte
2ED1 3ØØ7     2Ø5Ø        BMI HRERR    chyba? ---
2ED3 C9FF     2Ø7Ø        CMP #ØFF     hlavička bin.souboru?
2ED5 DØØ3     2Ø8Ø        BNE HRERR    ne --->
              2Ø9Ø ;
2ED7 A9ØØ     21ØØ        LDA #Ø       flag"žádná chyba"
2ED9 6Ø       211Ø        RTS
              212Ø ;
2EDA A9FF     213Ø HRERR  LDA #ØFF     není binár.soubor !!
2EDC 6Ø       214Ø        RTS
              215Ø ;
              216Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              217Ø ; Přenos dat
              218Ø ; Čtení souboru SRC,zápis do souboru DST
              219Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              22ØØ ;
2EDD A9ØØ     221Ø TRANS  LDA #Ø       flag pro konec soubo-
                                       ru := špatně
2EDF 85CC     222Ø        STA ENDFLG
2EE1 A23Ø     223Ø        LDX #IOCB3
2EE3 A9Ø7     224Ø        LDA #CGBINR příkaz GET CHARACTERS
2EE5 2Ø282F   225Ø        JSR SETUP    definovat IOCB
2EE8 ADE6Ø2   227Ø        LDA MEMTOP+1 vyšší byte MEMTOP
2EEB 38       228Ø        SEC          vypočíst délku buferu
2EEC E931     229Ø        SBC #BUFFER/256+1 MEMTOP-BUFFER-1
2EEE 9D49Ø3   23ØØ        STA ICBLH,X délka buferu (MSB)
2EF1 A9ØØ     231Ø        LDA #Ø       LSB :=Ø
2EF3 9D48Ø3   232Ø        STA ICBLL,X
              233Ø ;
2EF6 2Ø56E4   234Ø        JSR CIOV     čtení buferu
2EF9 CØ88     235Ø        CPY #EOF     načten konec souboru?
2EFB DØØ6     236Ø        BNE TRA1     ne --->
2EFD A9FF     238Ø        LDA #ØFF     flag konec souboru na
                                      "dobrý"
2EFF 85CC     239Ø        STA ENDFLG
2FØ1 DØØ2     24ØØ        BNE TRA2     vždy ===>
              241Ø ;
2FØ3 3Ø2Ø     242Ø TRA1   BMI TRAERR   ostatní chyby? --->
              243Ø ;
              244Ø ; Teď se připojí soubor
              245Ø ;
```

```
2FØ5 BD48Ø3   246Ø TRA2   LDA ICBLL,X počet načtených byte
2FØ8 BC49Ø3   247Ø        LDY ICBLH,X do DST-IOCB
2FØB A24Ø     248Ø        LDX #IOCB4
2FØD 9D48Ø3   249Ø        STA ICBLL,X
2F1Ø 98       25ØØ        TYA
2F11 9D49Ø3   251Ø        STA ICBLH,X
2F14 A9ØB     252Ø        LDA #CPBINR příkaz PUT CHARACTERS
2F16 2Ø282F   253Ø        JSR SETUP   definovat IOCB
2F19 2Ø56E4   254Ø        JSR CIOV    připojit soubor
2F1C 3ØØ7     255Ø        BMI TRAERR  nastala chyba?
2F1E A5CC     257Ø        LDA ENDFLG  byl SRC soubor už u
                                      konce?
2F2Ø FØBB     258Ø        BEQ TRANS   ne --->
              259Ø ;
2F22 A9ØØ     26ØØ        LDA #Ø      ukončení bez chyby
2F24 6Ø       261Ø        RTS
              262Ø ;
2F25 A9FF     263Ø TRAERR LDA #ØFF    nastala chyba !
2F27 6Ø       264Ø        RTS
              265Ø ;
              266Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              267Ø ;Definování IOCB pro přenos buferu
              268Ø ;<X>:číslo IOCB * 16;<A>:kód příkazu
              269Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              27ØØ ;
2F28 9D42Ø3   271Ø SETUP  STA ICCMD,X uložit příkaz
2F2B A9ØØ     272Ø        LDA #BUFFER&LO  adresu buferu
2F2D 9D44Ø3   273Ø        STA ICBAL,X do IOCB
2F3Ø A93Ø     274Ø        LDA #BUFFER/HI
2F32 9D45Ø3   275Ø        STA ICBAH,X
2F35 6Ø       276Ø        RTS
              277Ø ;
              278Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              279Ø ;CLOSE: Uzavření souborů SRC a DST
              28ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              281Ø ;
2F36 A9ØC     282Ø ACLOSE LDA #CCLOSE
2F38 A23Ø     283Ø        LDX #IOCB3  nejprve IOCB#3
2F3A 9D42Ø3   284Ø        STA ICCMD,X
```

```
2F3D 2056E4   2850        JSR CIOV
2F40 A90C     2860        LDA #CCLOSE
2F42 A240     2870        LDX #IOCB4  potom IOCB#4
2F44 9D4203   2880        STA ICCMD,X
2F47 2056E4   2890        JSR CIOV
2F4A 60       2900        RTS
              2910 ;
              2920 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2930 ;Kopírování názvu souboru do buferu FNAME1
              2940 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              2950 ;
2F4B A20F     2960 AMOVE  LDX #15      16 byte
2F4D BD402E   2980 AMO1   LDA FNAME,X z mezibuferu
2F50 9D502E   2990        STA FNAME1,X  do FNAME1
2F53 CA       3000        DEX          už všechno?
2F54 10F7     3010        BPL AMO1     ; ---
2F56 60       3020        RTS
              3030 ;
              3040 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              3050 ;Výstup hlášení o chybách
              3060 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              3070 ;
2F57 A200     3080 APRERR LDX #IOCB0  IOCB č. 0(E: )
2F59 A909     3090        LDA #CPTXTR výstup textu
2F5B 9D4203   3100        STA ICCMD,X
2F5E A976     3110        LDA #MESERR&LO  ukazatel na hlášení
2F60 9D4403   3120        STA ICBAL,X zapsat do IOCB
2F63 A92F     3130        LDA #MESERR/HI
2F65 9D4503   3140        STA ICBAH,X
2F68 A927     3150        LDA #39      nastavit max.délku přenosu
2F6A 9D4803   3160        STA ICBLL,X (ve skutečnosti na 39)
2F6D A900     3170        LDA #0       MSB
2F6F 9D4903   3180        STA ICBLH,X
2F72 2056E4   3190        JSR CIOV
2F75 60       3200        RTS
              3210 ;
2F76 2A2A2A20 3220 MESERR .BYTE "*** CHYBA ***", EOL
2F7A43485942
2F7E 412020A2A
2F82 2A9B
```

Dávkové zpracování neboli tajemství souboru EXECUTE

Výhodnou vlastností DOS XL (a taky DOS HK, viz příloha) je dávkové zpracování tzv. Execute souborů, nazývané taky zpracování "BATCH". Výhody dávkového zpracování si ukážeme na příkladu:
Předpokládejme, že máme program, který se skládá ze strojového programu, řetězce znaků, titulního obrázku a snad také zvukového modulu, popsaného v knize. Dále předpokládejme, že titulní obrázek a řetězec znaků je vytvořen odpovídajícím pomocným souborem a oba tyto programy jsou k dispozici jako binární soubor. Pro odskoušení takového programu musíme zadat následující příkazy (D1: pochází z DOS XL):

```
D1:LOAD SPIEL.OBJ      (strojový program)
D1:LOAD ZSATZ.OBJ      (načtení řetězce )
D1:LOAD TBILD.OBJ      načtení titulního obrázku)
D1:LOAD SOUND.OBJ      (zvukový stimul  )
D1:RUN  3000           (start programu  )
```

V programech jsou většinou skryty záludné chyby, které v nejhorším případě zablokují počítač. V takovém případě musíme načíst zdrojový text, najít v něm a odstranit chyby, provést nový překlad a znovu zapsat uvedenou posloupnost povelů. Tuto činnost musíme provádět tak dlouho, dokud se nám nepodaří odstranit všechny chyby. Je to náročná a únavná práce.
Kdo se ocitl v podobné situaci, přál by si takovou posloupnost příkazů automatizovat. ATARI DOS 2.0 (a také nový DOS 2.5) nechá programátora v takové situaci programátora na holičkách. Nezoufejte, protože DOS XL má elegantní možnost: soubor Execute,

Jak sestavit soubor Execute ?

Potřebujete k tomu jakýkoliv textový editor. Nezáleží na tom, zda pracuje s čísly řádků nebo ne. Napíšeme textový soubor, který bude obsahovat příkazy DOS ve stejném pořadí, jako bychom je zadávali ručně.

Pro uvedený příklad bude textový soubor následující:

```
100 REM Soubor Execute pro testování programu
110 LOAD SPIEL.OBJ
120 LOAD ZSATZ.OBJ
130 LOAD TBILD.OBJ
140 LOAD SOUND.OBJ
150 END
```

Tento soubor uložíme pod názvem souboru SPIELTST.EXC. Můžete zvolit jakýkoliv jiný tvar názvu, musíte však použít příponu .EXC.

Pár slov k textovému editoru: pro napsání souboru EXC můžete použít libovolný dostupný editor, např. cartridge Editor/Assembler, nebo MEDIT, ATARI Schreiber, casrtridge Editor ACTION. Čísla řádků není nutné psát, ale nevadí jsou-li použity. Použijete-li cartridge Editor/Assembler, musíte zadat čísla řádků. Jednoduchou možnost poskytuje taky DOS XL.

Napište:

```
D1:TYPE E: SPIETST.EXC
```

a můžete psát text. Soubor ukončíte znakem CTRL 3 (konec souboru). POZOR ! Nesmíte použít tlačítka pro řízení kurzoru. BASIC editor vůbec nemůžete použít, protože se pokouší interpretovat text jako program BASICu, což vede k chybám.

Sestavený soubor EXC můžeme pokaždé uvést do činnosti zadáním jeho názvu, přičemž před název souboru zadáváme "zavináč". V našem případě:

```
D1:@SPIELTST
```

Příponu EXC nesmíme zadávat, protože bychom obdrželi hlášení "Soubor nenalezen" (File not found).

Příkazy pro soubory EXC

Speciální příkaz Execute už dávno známe z BASICu. Je to příkaz REM. Všechno, co stojí za REM, je interpretováno jako komentář a nemá žádný vliv na průběh EXC.
Příklad obsahuje také příkaz END, pomocí kterého snadno zastavíme dávkové zpracování. Dále jsou k dispozici ještě dva příkazy: NOSCREEN a SCREEN (zkráceně NOS a SCR). Odpovědí na tyto příkazy je vypnutí a zapnutí obrazovky. Logicky mů-

žete všechny "vnitřní" a "vnější" příkazy DOS XL použít uvnitř souboru EXC. Můžete přirozeně použít i příkazy, které si sami zapíšete, např. příkaz APPEND z předcházející kapitoly, nebo příkaz BCOM, který je popsán v následující kapitole.

POZOR !: V případě, že místo DOS XL ještě používáte starší verzi OS/A+ DOS, nesmí být zadávací řádek delší než 6Ø znaků, jinak přeteče vnitřní bufer. DOS XL (a také DOS HK) dovolují délku řádku 128 byte.

Soubor STARTUP.EXC

Tento soubor už určitě znáte. Většinou je použit k zobrazení obsahu diskety ihned po natažení, k tomu obsahuje příkaz DIR. STARTUP.EXC může mnohem více. Jedná se přitom o zvlášť obsloužený soubor z DOSu (podobně jako soubor AUTORUN.SYS u ATARI DOS), který je zavolán hned po natažení. Na rozdíl od souboru AUTORUN.SYS není STARTUP.EXC volán jako strojový program, ale je proveden jako soubor EXC. Budete-li tedy mít svůj soubor EXC pod názvem STARTUP.EXC, bude tento soubor proveden hned po natažení operačního systému. Příklad najdete na programové disketě HK.

Proměnné systému Execute

Stavy postupu Execute budou uloženy do vnitřní paměti:

<CPALOC>+	
CPEXFL (ØØB)	Flag Execute, ukazuje, zda Execute běží = 128 -Execute je aktivní
CPEXFN (ØØC)	Počátek názvu souboru EXC
CPEXNP (Ø1C)	Ukazatel NOTE/POINT v souboru EXC

Adresové údaje jsou relativně vztaženy k začátku DOSu (vektor CPALOC).

Změníte-li proměnné ve vlastním programu, můžete ovlivnit průběh souboru EXC. Můžete tak realizovat smyčkové struktury a pod. K tomu musíte použít hodnotu CPEXNP, kterou DOS

XL hlídá pomocí funkce NOTE, jak daleko je již zpracován soubor Execute. Pro nový úkol je nutný příkazový řádek.DOS XL umísti ukazatel souboru do souboru EXC, podle hodnoty CPEXNP přejde na aktuální řádek, načte jej a zaznamená si novou hodnotu NOTE (jako ukazatel na následující řádek) do CPEXNP.

Jednoduchý příklad pro využití systémových proměnných: Zastavení souboru Execute pomocí strojového programu. Tato rutina byla využívána např. v programu APPEND, když nastala chyba.

```
LDY #CPEXFL        odchylka flagu Eecute
LDA #Ø             smazat flag Execute
STA (CPALOC),Y     konec provozu Execute
```

Problémy jsou přirozeně i se soubory Execute. Problémy nastanou, když se pokusíme dávkovým zpracováním spustit programy v BASICu. Jakmile nastane příkaz CAR (Cartridge), ohlásí se BASIC nápisem READY a dávkové zpracování bude zastaveno. Po zadání příkazu DOS bude sice soubor EXC pokračovat, ale během dávkového zpracování nebudeme nic zadávat, Jak si v tomto případě pomoci? Jsou dvě možnosti:

1. Místo Cartridge BASIC použijeme soubor BASIC/A$^+$, kterému můžeme při jeho volání zadat, který program v BASICu načíst a odstartovat:

   D1: BASIC D: BEISPIEL.BAS

   Toto zadání nejprve načte z diskety BASIC/A$^+$ a potom zpracuje program BEISPIEL.BAS. Nevýhodou této metody je, že uvažované programy musíme převzít ve formátu BASIC/A$^+$ a ten musíme znovu načíst do paměti při každém volání programu v BASICu a konečně BASIC/A$^+$ vám taky zabere část paměti v RAM a na vaší pracovní disketě.

2. Elegantnější je možnost s Cartridge BASIC. Místo příkazu CAR použijeme malý trik: program BCOM, podrobně popsaný v následující kapitole, který v režimu DOS může provádět příkazy v Basicu. Příklad bude vypadat následovně:

   D1: BCOM RUN "D:BEISPIEL.BAS"

Pokud program BEISPIEL.BAS obsahuje jako poslední příkaz

DOS, bude zpracování souboru EXC po ukončení programů v BASICu pokračovat, není-li proces Execute záměrně ukončen.

BCOM - Příkazy BASICu v DOS

Při použití souboru Execute je potřebné mít možnost provádět přijaté příkazy BASICu v režimu DOS. Jak jsme již viděli v minulé kapitole, nejde to provést jednoduše, protože při zavolání BASICu se proces Execute zastaví a pokračování je možné jenompo zásahu z klávesnice. Pomocí BCOM překlenemo tuto překážku a můžeme příkazy z BASICu provádět v režimu DOS. Takto je možné zadávat z DOSu krátké programy a odstartovat je. Dále je taky možné BCOM spojit s jinými programovacími jazyky na Cartridge.

Volání BCOM

Abychom BCOM mohli použít, musíme mít nejprve soubor BCOM.COM na disketě v driveru 1 a dále musí být přítomný BASIC, tzn. musí být zasunuta Cartridge nebo modely XL/XE musíme zapnout bez stlačení OPTION.

Syntax zadání:

D1: BCOM <řetězec příkazů BASIC>

Ještě jedna zvláštnost: středník bude v příkazovém řetězci BASICu převeden na znak EOL (Return(. Můžete tedy vměstnat více zadání do jednoho řádku právě tak, jako byste po každém příkazu zadávali RETURN.

D1: BCOM SOUND Ø,1ØØ,1Ø,8; SOUND 1,1Ø1,1Ø,8; DOS

odpovídá následujícímu ručnímu zadání:

CAR	<RET>	(bude simulován BCOM)
SOUND Ø,1ØØ,1Ø,8	<RET>	(1. příkaz BASICu)
SOUND 1,1Ø1,1Ø,8	<RET>	(2. příkaz BASICu)
DOS	<RET>	(návrat do DOSu)

V tomto jednoduchém příkladě by bylo možno nahradit středník dvojtečkou a příkazový řetězec deklarovat jako jediný BASICový řádek.

Jsou ale případy, ve kterých je oddělené zadání nutné.
Příklad:
Chceme načíst program jako ATASCII soubor, řízený souborem Execute. Jenže BASIC ukončí zpracování souboru EXC po příkazu ENTER, takže pro:

```
D1: BCOM ENTER "D:BEISPIEL.LST":DOS
```

se místo očekávaného návratu do DOSu objeví READY. Správné řešení:

```
D1: BCOM ENTER "BEISPIEL.LST"; DOS
```

Způsob práce s BCOM

Basic dostává svoje zadání voláním CIO přes Screen Editor, pro který je vždy otevřen IOCB č.Ø. Trik s BCOM je v tom, že zdroj zadání (Screen Editor) krátkodobě přesměrujeme tak, že zadání nejde z klávesnice, ale z buferu v paměti. Basic to nezpozoruje, protože zadání dostane z editoru a je mu celkem jedno, odkud se zadání do editoru dostalo.
Abychom porozuměli celému postupu, musíme si blíže objasnit způsob práce CIO. Při otevření kanálu I/O porovnává CIO zadaný název zařízení (E:, C:, P:) s tabulkou (HATABS, Ø31A), ve které jsou uvedena všechna další zařízení, která jsou k dispozici a jejich příslušnými adresami. Když je žádané zařízení nalezeno, bude index v tabulce HATABS uložen do IOCB pod ICHID. CIO nyní musí provést přenos dat, takže nahlédne do HATABS pod uvedeným indexem a najde vedle zkratky zařízení jeho adresu, která ukazuje znovu na tabulku skoků, ve které jsou uloženy adresy skoků pro elementární funkce zařízení.
Taková tabulka vnitřních rutin existuje pro každé zařízení (editor je pro CIO taky zařízení) a obsahuje adresy podprogramů, které realizují skutečné příkazy.

Tabulka vnitřních rutin

změna ------

OPEN - 1	
CLOSE - 1	
READ - 1	
WRITE - 1	adresa - 1
STATUS - 1	specifické podprogramy
SPECIAL - 1	podle zařízení
JMP INIT	

Příklad: Má být proveden příkaz GET CHARACTERS. Jak už bylo vysvětleno, tabulka vnitřních rutin je z HATABS a CIO si vyzvedne adresu rutiny (GETCHR), pomocí které může obdržet jednotlivé znaky. Rozhodnutí, kolik znaků je čteno zůstane na CIO, rutina GETCHR po zavolání vrátí jeden znak ve střadači.

Nyní zpět k BCOM. Musíme zdroj zadání Screen Editoru, tedy rutinu GETCHR nahradit vlastní rutinou. Přímo to nelze provést, protože tabulka skoků editoru je v ROM. V RAM je ale tabulka HATABS, do které uložíme adresu tabulky skoků, ukazující na bufer v RAM (EDITAB), kam jsme předtím tabulku skoků překopírovali. V této okopírované tabulce můžeme změnit adresu skoku rutiny editoru GETCHR a necháme ji ukazovat na naší vlastní rutinu GETCHR (v BCOM nazvanou TXTGET). Tento celý postup bude proveden podprogramem EDIAEN.

Dále bude část povelového buferu DOSu XL, který obsahuje BASICový řetězec, zkopírován do vnitřního buferu programu BCOM. Pro získání delšího buferu můžeme příkazy BASICu přepsat podprogramem EDIAEN, který už nebudeme potřebovat. Výpočet adresy a překopírování příkazového buferu (viz kapitola Nové příkazy pro DOS XL) bude provedeno podprogramem CMDKOP. Nová rutina GET CHARACTERS se jmenuje TXTCHR a jedinou její činností je, že vrátí do střadače znak z vnitřního buferu (TXTBUF). Středník bude přitom změněn na EOL. Konec textového buferu bude označen znakem EOL, po němž přijde ke slovu rutina TXTEND, která vektor HATABS nastaví zpět na původní tabulku skoků editoru v ROM. Tím se znovu nastaví zpět původní stav editoru.

Celý program (kromě buferu) je umístěn na stránce 6, což má své výhody. Pro BASIC máme k dispozici celou paměť a nemusíme měnit ukazatele MEMLO nebo RAMTOP.

Poznámka: DOS XL obsahuje soubor DO.COM, který pracuje podobně jako BCOM. Velký rozdíl mezi oběma programy je v tom, že BCOM rozumí i jiným větším přáním, kdežto DO.COM ne.

```
                Ø1ØØ ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø11Ø ; BCOM: Příkazy BASICu v DOS XL
                Ø12Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø18Ø ;
                Ø19Ø ;Volání v DOS XL: BCOM<řetězec příkazů
                      BASICU>
                Ø2ØØ ;
                Ø22Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø23Ø ;Proměnné operačního systému
                Ø235 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø24Ø ;
=Ø31A           Ø25Ø HATABS =   $Ø31A     tab.vnitř.rutin
=ØØ21           Ø26Ø MAXDEV =   $21       max.délka tab.
=Ø342           Ø28Ø ICCMD  =   $Ø342     proměnné IOCB
=Ø344           Ø29Ø ICBAL  =   $Ø344
=Ø345           Ø3ØØ ICBAH  =   $Ø345
=Ø348           Ø31Ø ICBLL  =   $Ø348
=Ø349           Ø32Ø ICBLH  =   $Ø349
=ØØØ9           Ø33Ø CPTXTR =   $Ø9       příkaz PUT RECORD
=E456           Ø34Ø CIOV   =   $E456     vektor CIO
=BFFA           Ø35Ø CARLOC =   $BFFA     vektor Catridge Go
=ØØØA           Ø352 CPALOC =   $ØA       vektor DOS XL
=ØØØA           Ø353 CPBUFP =   $ØA       odhcylka ukazatele
                                          buferu CMD
=ØØ3F           Ø354 CPCMDB =   $3F
=ØØØB           Ø355 CPEXFL =   $ØB       odchylka flagu Execute
                Ø38Ø ;
                Ø 381 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø39Ø ;Programové proměnné a konstanty
                Ø391 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                Ø4ØØ ;
```

```
=ØØCB      Ø41Ø TBSAVL =   $CB       starý vektor tab.editoru
=ØØCC      Ø411 TBSAVH =   $CC       MSB
=ØØCD      Ø42Ø TXTPNT =   $CD       ukazatel v text.buferu
=ØØCE      Ø421 DEVID  =   $CE       pamatováno na index
                                     v HATABS
=ØØCF      Ø43Ø REG1L  =   $CE       pomocný reg (2 byte)
=ØØDØ      Ø44Ø REG1H  =   $DØ
=ØØ5F      Ø46Ø BUFLEN =   $Ø6FF-TEXT  max.délka buferu
=ØØ9B      Ø47Ø EOL    =   $9B       znak End-of-Line
=ØØFF      Ø471 LO     =   $FF       maska pro LSB
=Ø1ØØ      Ø472 HI     =   $Ø1ØØ     dělitel pro MSB
           Ø54Ø ;
           Ø55Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           Ø56Ø ; Hlavní program
           Ø561 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           Ø57Ø ;
ØØØØ       Ø571        *=   $Ø6ØØ    stránka 6
           Ø572 ;
Ø6ØØ 2ØAØØ6 Ø58Ø        JSR EDIAEN    změna editoru
Ø6Ø3 BØØ8  Ø581         BCS ERROR     chyba! --->
Ø6Ø5 2Ø1AØ6 Ø59Ø        JSR CMDKOP    kopírovat bufer CMD
Ø6Ø8 BØØ3  Ø6ØØ         BCS ERROR     řetězec příliš dlouhý
Ø6ØA 6CFABF Ø62Ø        JMP (CARLOC)  volat Cartridge ===>
           Ø63Ø ;
Ø6ØD 2Ø65Ø6 Ø631 ERROR  JSR TXTEND    editor do původního
                                      stavu
Ø61Ø A9ØØ  Ø632         LDA #Ø
Ø612 AØØB  Ø633         LDY #CPEXFL   vypnout
Ø614 91ØA  Ø634         STA (CPALOC),Y  případně běžící
                                        proces EXC
Ø616 2Ø76Ø6 Ø64Ø        JSR MESOUT    hlášení o chybě
Ø819 6Ø    Ø65Ø         RTS
           Ø67Ø ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           Ø68Ø ;UP CMDKOP: Kopírování příkazového buferu
           Ø681 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           Ø69Ø ;
Ø61A 18    Ø7ØØ CMDKOP CLC
Ø61B A5ØA  Ø71Ø         LDA CPALOC    zákl.adresa DOS XL
Ø61D 693F  Ø72Ø         ADC #CPCMDB   přičtení odchylky
                                      buferu CMD
```

```
Ø61F B5CF    Ø73Ø          STA REG1L      LSB do pomoc.registru
Ø621 A5ØB    Ø74Ø          LDA CPALOC+1   MSB
Ø623 69ØØ    Ø75Ø          ADC #Ø         případný přenos
Ø625 85DØ    Ø76Ø          STA REG1H      MSB do pomoc.registru
             Ø77Ø ;
Ø627 A2ØØ    Ø78Ø          LDX #Ø         inic.ukaz.ve vlast.buferu
Ø629 AØØA    Ø79Ø          LDY #CPBUFP    odchylka ukazat.buferu
Ø62B B1ØA    Ø8ØØ          LDA (CPALOC),Y hodnotu ukazat.buferu
Ø62D AB      Ø81Ø          TAY            použít jako nový index
             Ø82Ø ;
Ø62E B1CF    Ø83Ø CMDK1    LDA (REG1L),Y vyzvednout znaky z CMD
                                          buferu
Ø63Ø 9DAØØ6  Ø84Ø          STA TEXT,X     a ulož do vlast.buferu
Ø633 C99B    Ø85Ø          CMP #EOL       konec zadávac. řádku
Ø635 FØØB    Ø86Ø          BEQ CMDEND     ano --->
             Ø87Ø ;
Ø637 C8      Ø88Ø          INY            další znak
Ø638 E8      Ø89Ø          INX
Ø639 EØ5F    Ø9ØØ          CPX #BUFLEN    bufer plný ?
Ø63B DØF1    Ø91Ø          BNE CMDK1      ne --->
Ø63D 38      Ø92Ø          SEC            chyba!
Ø63E 6Ø      Ø93Ø          RTS
             Ø931 ;
Ø63F 1B      Ø94Ø CMDEND   CLC            nenastala chyba
Ø64Ø 6Ø      Ø95Ø          RTS
             Ø951 ;
             Ø952 ; Rezervovat místo pro novou tab.editoru
             Ø953 ;
Ø641         Ø955 EDITAB *=  *+Ø1Ø
             Ø96Ø ;
             Ø961 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø97Ø ; Nová rutina CHRGET
             Ø971 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             Ø98Ø ;
Ø651 A4CD    Ø99Ø TXTGET   LDY TXTPNT     aktual.ukazat. textu
Ø653 B9AØØ6  1ØØØ          LDA TEXT,Y     znak z textového buferu
Ø656 C99B    1Ø1Ø          CMP #EOL       konec textu ?
Ø65B FØØB    1Ø2Ø          BEQ TXTEND     ano --->
             1Ø3Ø ;
```

```
Ø65A C93B    1Ø4Ø          CMP #";        je oddělovací znak ?
Ø65C DØØ2    1Ø5Ø          BNE TG1        ne --->
             1Ø6Ø ;
Ø65E A99B    1Ø7Ø          LDA #EOL       nahradit znakem EOL
             1Ø8Ø ;
Ø66Ø E6CD    1Ø9Ø TG1      INC TXTPNT     inkrement ukazat.textu
Ø662 AØØ1    111Ø          LDY #1         flaf pro "všechno v
                                          pořádku"
Ø664 6Ø      112Ø          RTS
             113Ø ;
     =Ø65Ø   114Ø TGVECT =  TXTGET-1      adresa skoku TXTGET
             115Ø ;
             1151 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             116Ø ;Konec textů, vektor zpět, vydat EOL
             1161 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             117Ø ;
Ø665 A4CE    118Ø TXTEND LDY DEVID        vyzvednout index za-
                                          řízení
Ø667 A5CB    119Ø          LDA TBSAVL     starý ukazatel editor.
                                          tabulky
Ø669 991BØ3  12ØØ          STA HATABS+1,Y vrátit do tab.rutin
Ø66C A5CC    121Ø          LDA TBSAVH     stejně tak MSB
Ø66E 991CØ3  122Ø          STA HATABS+2,Y
Ø671 AØØ1    123Ø          LDY #ØØ1       všechno jasné
Ø673 A99B    124Ø          LDA #EOL       vydat EOL
Ø675 6Ø      125Ø          RTS
             126Ø ;
             1261 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             1262 ;Hlášení o chybě
             1263 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             1264 ;
Ø676 A2ØØ    127Ø MESOUT LDX #Ø         použito IOCB Ø(editor)
Ø678 A9Ø9    128Ø          LDA #CPTXTR    příkaz:vyslat text.záznam
Ø67A 9D42Ø3  129Ø          STA ICCMD,I    do příkaz.registru
Ø67D A995    13ØØ          LDA #MESERR&LO ukazatel na hlášení
                                          chyba
Ø67F 9D44Ø3  131Ø          STA ICBAL,X    zapsat do IOCB
Ø682 A9Ø6    132Ø          LDA #MESERR/HI
Ø684 9D45Ø3  133Ø          STA ICBAH,H
Ø687 A927    134Ø          LDA #39        nastavit max.délku
```

```
0689 9D4803  1350          STA ICBLL,X
068C A900    1360          LDA 00
068E 9D4503  1370          STA ICBAH,X
0691 2056E4  1380          JSR CIOV       a zobrazit >>>
0693 60      1390          RTS
             1400 ;
             1401 ; text chybového hlášení
             1410 ;
0695 2A2046 45 1420 MESERR .BYTE "*FEHLER*",EOL
0699 484C4552
069D 202A9B
             1430 ;
     =06A0   1431 TEXT   = *           odtud rozsah buferu
             1432 ;
             1433 ; POZOR: následující program může být přep-
                           sán buferem !!!
             1450 ;
             1451 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             1460 ;Rutina GET CHARACTER pro změnu editoru,
             1470 ;zadání pak jde z textového buferu
             1471 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
             1480 ;
06A0 A945    1490 EDIAEN LDA #'E       hledání zápisu editru
                                       v tab. rutin
06A2 A021    1500        LDY #MAXDEV
             1510 ;
06A4 D91A03  1520 DEVS1  CMP HATABS,Y  je zařízení editor ?
06A7 F007    1530        BEQ DEVS2     ano --->
06A9 88      1540        DEY
06AA 88      1550        DEY
06AB 88      1560        DEY
06AC 10F6    1570        BPL DEVS1     jiné zařízení --->
             1580 ;
06AE 38      1590        SEC           Editor nenalazen !?
06AF 60      1600        RTS           to se nesmí stát !
             1610 ;
06B0 84CE    1620 DEVS2  STY DEVID     uložit index v tab.
                                       rutin
06B2 B91B03  1630        LDA HATABS+1,Y novou zákl.adresu
                                       tab.rutin
```

```
Ø6B585CB     164Ø          STA TBSAVL     uložit, starou adresu
Ø6B7 A941    165Ø          LDA #EDITAB LO  přitom uschovat
Ø6B9 991BØ3  166Ø          STA HATABS+1,Y
Ø6BC B91CØ3  167Ø          LDA HATABS+2,Y stejně tak pro MSB
Ø6BF 85CC    168Ø          STA TBSAVH
Ø6C1 A9Ø6    169Ø          LDA #EDITAB/HI
Ø6C3 991CØ3  17ØØ          STA HATABS+2,Y
             171Ø ;
Ø6C6 AØØF    172Ø          LDY #ØØF       tab.skoků editoru
Ø6C8 B1CB    173Ø EDIKOP LDA (TBSAVL),Y zkopírovat do
Ø6CA 9941Ø6  174Ø          STA EDITAB,Y  rezervovaného místa
Ø6CD 88      175Ø          DEY
Ø6CE 1ØF8    176Ø          BPL.EDIKOP
             177Ø ;
Ø6DØ A95Ø    178Ø          LDA #TGVECT&LO  v nové tab.
Ø6D2 8D45Ø6  179Ø          STA EDITAB+4  vektor GETCHR směrovat
Ø6D5 A9Ø6    18ØØ          LDA #TGVECT/HI na vlastní rutinu
Ø6D7 8D46Ø6  181Ø          STA EDITAB+5
             182Ø ;
Ø6DA A9ØØ    183Ø          LDA #Ø         ukazatel textu nasta-
                                           vit na nulu
Ø6DC 85CD    184Ø          STA TXTPNT
             185Ø ;
Ø6DE 18      186Ø          CLC            žádná chyba
Ø6DF 6Ø      187Ø          RTS
```

Tipy a triky s B C O M

Především v testovací a vývojové fázi programu se BCOM ukáže jako potřebný a také výkonný nástroj. Dále je uveden tip, co všechno lze s BCOM udělat.

- Smíšené nahrávání programů v BASICu a ve stroj.kódu.:

```
BCOM RUN "D:INIT.BAS"
LOAD BEISPIEL.OBJ
BCOM RUN "D:BEISPIEL.BAS"
```

V tomto souboru Execute bude nejprve odstartován BASICový

program INIT.BAS (např. pro rezervování paměti ap.), načte se strojový program BEISPIEL.OBJ a nakonec se načte program v BASICu a odstaňuje. Tímto způsobem můžeme zavést do paměti komplexní programové systémy skládající se ze strojových programů, programů v BASICu a binárních souborů (obr., řetězce, ...).

Poznámka: Aby program v BASICu předal zpět řízení souboru Execute, musí být ukončen příkazem DOS.

- Rezervování paměti :

Chcete-li mít současně v paměti program v BASICu i ve strojovém kódu, je nutné rezervovat odpovídající místo v paměti pro strojový program. Jedna možnost byla uvedena, rezervování provede INIT.BAS.

Rezervování můžete provést přímo jako soubor Execute.

```
BCOM POKE 106,144: GR.0; DOS
```

Tím bude nastavena horní hranice paměti $9000, rozsah $9000 - $9FFF bude rezervován (pro ATARI počítače s více než 40 kByte). Samozřejmě můžeme rezervovat větší nebo menší rozsah paměti, místo 144 použijeme jinou hraniční stránku (čtěte "Kam se strojovými programy").

- Odpověď v BCOM na příkaz INPUT :

Obsahuje-li program v BASICu, zavolaný pomocí BCOM, příkaz INPUT, můžeme na něj odpovědět přímo v BCOM.

Příklad:

```
10 PRINT "zadej A";: INPUT A: PRINT
20 PRINT "zadej B";: INPUT B: PRINT
30 PRINT "A+B="; A+B
```

vede k zadání

```
BCOM RUN "D:BEISPIEL.BAS";5;7
```

Program vytiskne:

```
A+B=12
```

Všimněte si příkazu PRINT po příkazu INPUT, bez kterého by byl kratší výpis na obrazovce, ale neobjevilo by se na obrazovce zadání BCOM.

- Změny programů v BASICu.:

Pomocí BCOM můžeme vkládat nebo vymazat programové řádky podobně, jako bychom to dělali ručně. (Pozor: všimněte si dalšího bodu !).

```
BCOM LOAD "D: BEISPIEL.BAS"; DOS
BCOM 1ØØ REM * Tento řádek je nový!; DOS
BCOM SAVE "D:BEISPIEL.BAS"; DOS
```

Tímto souborem Execute bude vložen do programu BEISPIEL řádek 1ØØ. Nahradíme-li druhý řádek souborem EXC:

```
BCOM 1Ø;2Ø;3Ø;4Ø; DOS
```

budou z programu BEISPIEL vymazány řádky 1Ø až 4Ø.

Tato vlastnost BCOM je důležitější, než se vám na první pohled zdá. Představte si, že máte program v BASICu, který je podrobně komentován řádky REM a navíc obsahuje podprogram, který je třeba jenom pro testování programu. V pracovní verzi programu musíme mít k dispozici co nejvíc paměti, a proto je nutné odstranit nepotřebné části programu. Smažeme-li teď všechny řádky REM, bude celá námaha s komentováním programu zbytečná a program se stane nepřehledným.

POMOC: Napište soubor Execute podle uvedeného vzoru, který testovací verzi vašeho programu převede na verzi pracovní. Program můžete ladit v přehledné testovací verzi a opravený program pak snadno můžete převést na pracovní verzi, šetřící paměť.

- Popisovaný soubor Execute má určité "špeky":

Při každé nové inicializaci cartridge BASIC pomocí BCOM se ztratí aktuální program v BASICu. Tuto vadu můžeme vyřešit tak, že na počátek souboru Execute zavedeme následující řádek:

```
BCOM POKE 8,1: DOS
```

Tipy k programování v ASSEMBLERU

Každý, kdo to myslí vážně s programováním svého ATARI počítače, se dříve nebo později dostane k tématu programování ve strojovém kódu. Dále je uvedeno několik tipů a triků, které vám ulehčí život.

Dokumentace

Důležitou zásadou při psaní programu v assembleru je každý krok opatřit komentářem. Stojí to více psaní a spotřebuje se více paměti pro zdrojový soubor, ale v konečném efektu se to vyplatí. Myslete taky na to, že na rozdíl od BASICu komentáře neobsazují paměť v přeloženém strojovém programu, protože při překladu zůstanou mimo. K dokumentaci rovněž patří nadpisy pro každý podprogram, volba účelných názvů návěstí a využití konstant definovaných na počátku programu místo číselných hodnot.

Který assembler použít ?

Pro ATARI počítače existuje celá řada více nebo méně komfortních assemblerů. Zde je pro porovnání uvedeno několik z nich.

1. Cartridge Editor/assembler:

Pokud můžete, dejte od něj ruce pryč. Nejednou doporučovaný pro začátečníky, kteří však většinou trpí flustraci v důsledku "hlemýždí" rychlosti překladu. Jedinou myslitelnou výhodou je, že většina zveřejněných programů využívá syntaxe E/A cartridge, která tak tvoří základ mnohých jiných assemblerů, včetně velkých jmenovců.

2. EASMD (Optimized Systems Software):

Většinou platí totéž co pro E/A cartridge. EASMD má navíc přídavnou funkci INCLUDE, je ale stejně pomalý jako cartridge. Podbně jako E/A obsahuje monitor, který ovšem má mnohé nedostatky.

3.) EDIT 65Ø2 (LJK-Software):

Velmi rychlý assembler s velkými možnostmi a s pseudoinstrukcemi. Delší data je možné překládat pomocí LINK. Bohužel ale nemá příkaz INCLUDE, který umožňuje segmentové programování. V něčím se liší od toho, na co jsme u ATARI zvyklí. Je použit řádkový editor, což znamená, že chcete-li editovat řádek 1ØØ, musíte nejprve zadat EDIT 1ØØ. Potom dostanete vylistovaný řádek s kurzorem na začátku a teprve nyní můžete řádek změnit. Je to rychlé, ale už se mi stalo, že v BASICu se objevilo najednou na obrazovce EDIT 1ØØØ ...

Bohužel syntax assembleru neodpovídá používanému standartu pro 65Ø2. Místo '='musíte použít 'EQU', místo '*='je zde 'ORG'ap. To neulehčuje přepis programů z knížek nebo časopisů. Doplňkem je rovněž výkonný monitor strojového jazyka (volání: MON), který zdržuje při zadávaní reverzní polskou notací, přitom ale umí výkonné příkazy jako např. zápis/čtení sektoru na disketě. Celkově lze říci: Dá se s ním PRACOVAT.

4.) MAC/65 (Optimized Systems Software):

Nyní jsme na rozpacích, nevíme, jak vám několika řádky popsat tolik kladů. MAC/65 považuji za nejlepší assembler, který byl kdy napsán pro ATARI počítače. Překlad včetně makroinstrukcí, probíhá velkou rychlostí. INCLUDE je samozřejmostí, stejně jako podmíněný překlad. Je k dispozici i příkaz .SBYTE, pomocí něhož lze uložit řetězce v kódu obrazovky. Zadávaný programový text je, podobně jako v BASICu, ihned zkoušen na chyby syntaxe. Zdrojové soubory můžeme zaznamenávat jako zhuštěné TOKEN soubory, takže zabírají méně místa na disketě. Jedna nevýhoda, pokud lze vůbec o nevýhodě mluvit, je, že chybí monitor (hlavně pro disketovou verzi). Přitom je k dispozici separátní program BUG/65, kterým je možné ladit program přeložený pomocí MAC/65. Abychom měli požitek z překladu "rychlostí světla", musíme na začátku programu odpojit listing příkazem .OPT NOLIST. Protože výstup listingu na obrazovku (pravděpodobně z důvodů kompatibility) využívá CIO normální cestou, je přitom omezení rychlosti dost podstatné. Doby překladu dosažené NOLIST jsou 3Ø - 5Ø krát kratší než s E/A cartridge.

SEGMENTOVÉ programování

je téma, se kterým se musíte vyrovnat, pokud se pouštíte do delších strojových programů. Zdrojový text takových programů může dosáhnout dnes nesmírných rozměrů. Pro průměrnou hru není zvláštností sto stránkový listing ve strojovém kódu. Ikdyž pracujete s "bleskovým" assemblerem MAC/65, může to být problém. Předpokládejme, že zdrojový text byl napsán v celku. Když se nevejde do paměti, můžeme jej rozdělit na více vzájemně propojených souborů. Když se v takovém programu objeví chyba, musíme jej pokaždé překládat v celé délce, což je náročné na čas. V tak dlouhých výpisech navíc rychle ztratíme přehled. Pomoc nám poskytne segmentování programu. Chápeme tím rozdělení dlouhého programu do jednotlivých segmentů zdrojového textu, které můžeme překládat odděleně. Když chceme programovat hru, musíme si představit, že existuje zvlášť zvukový modul (např.: stejný jako zvukový program na jiném místě této knihy), modulu grafiky, segment pro vydání výsledků a snad modul pro titulní hudbu.

Sestavování segmentů

Jak můžeme organizovat jednotlivé segmenty mezi sebou ? Věc není tak složitá. Co v první řadě potřebujeme, je assembler, který provádí příkaz INCLUDE. Tímto příkazem může být během překladu do souboru začleněn soubor jiný.

Jsou dva základní požadavky na spolupráci mezi dvěma segmenty:

1. Segment musí umět část programu volat z jiného segmentu.
2. DATA, která byla uložena jedním segmentem, musí taky umět číst jiný segment.

Oba tyto požadavky lze splnit, když zavedeme soubor 'Definice', který obsahuje pouze všechny podstatné informace a definice.:

a: Důležité peoměnné, které musí být přístupné pro všechny segmenty (globální proměnné).
b: Adresy skoků a počáteční adresy segmentů.

c. Konstanty, které jsou důležité pro všechny segmenty.

Jednoduchý soubor Definice pro program, který obsahuje hlavní program zvukový a hudební segment, může vypadat takto (Název soboru: DEFILE.SRC)

```
; globální proměnné
REG1    = $CD        registr v nulté stránce
REG2    = $CE        bude využíván všemi částmi programu
; počáteční adresa segmentu
HAUPT   = $9000      hlavní program
SOUND   = $9E00      zvukový modul
MUSIK   = $9800      hudební modul
; adresy skoků
SNDEIN  = SOUND+0    zvuk zapnut
SNDAUS  = SOUND+3    zvuk vypnut
MUSEIN  = MUSIK+0    hudbu zapnout
MUSAUS  = MUSIK+3    hudbu vypnout
```

Vidíme, že absolutní adresy jednotlivých programových segmentů budou taktéž stanoveny v souboru Definice. Každý segment musí nyní začínat příkazem INCLUDE se souborem Definice. Dále bude na začátek každého segmentu umístěna tabulka skoků a příkazů JMP, aby vstupní adresy měly pevnou hodnotu. Každé volání segmentů musí být provedeno přes tuto tabulku skoků.

Příklad segmentu (např.: segment SOUND):

```
    INCLUDE #D: DEFILE.SRC
    *=    SOUND          počát.adr.ze souboru Definice
                         tabulka skoků
    JMP SEIN             skok na: zvuk zapnout
    JMP SAUS             skok na: zvuk vypnout
    ...
SEIN LDA #0              náveští SEIN
    ...                  rutina pro 'zvuk zapnout'
```

Když nyní bude volán segment SOUND uvnitř jiného segmentu, pojmenovaného jako hlavní program, bude to vypadat následovně:

```
; hlavní program
  INCLUDE *D:DEFILE.SRC
  ...
  *= HAUPT
  ...
  JSR SNDEIN
  ...
```

Skok do podprogramu JSR SNDEIN nyní vede přes tabulku skoků zvukového segmentu na správnou adresu. Návěstí SNDEIN bylo už definováno v DEFILE.SRC, který bude připojen k hlavnímu programu pomocí INCLUDE.Každý segment, který bude připojen, má rovněž přístup na společné (globální) proměnné (REG1/2), které jsou taky obsažené v části INCLUDE.

Výhody takové struktury jsou zjevné: Jednotlivé segmenty mají malý rozsah, jsou přehledné a je-li v některém chyba, pro opravu stačí přeložit jen příslušný modul. Dále přirozeně můžeme moduly ze stávajícího programu převzít do programu nového.

Když chceme segmentovaný program spustit, musíme načíst do paměti všechny jednotlivé strojové soubory, v OS/A+DOS je to možné provést příkazy:

```
LOAD MUSIK.OBJ
LOAD SOUND.OBJ
...
```

Tuto únavnou úlohu můžeme svěřit souboru Execute, který obsahuje odpovídající příkazy (viz. kapitola Provoz BATCH pro DOS XL). Konečný tvar můžeme dát programu tím, že v všechny strojové segmenty navzájem pospojujeme pomocí programu APPEND (viz kapitola APPEND pro DOS XL) nebo jednoduše všechny soubory načteme do paměti a celou obsazenou paměť znovu uložíme jediným příkazem SAVE.

Kam s programy ve strojovém kódu

Pokaždé, když má BASIC spolupracovat sestrojovým programem, vyvstane stejná otázka: Kde je možné uložit do paměti strojový program ? Podobné problémy jsou s umístěním znakových řetězců dat pro Playr-Missile ap.

Problém je možné řešit několika způsoby:

a) Nejjednodušší možnost, samozřejmě jen pro velmi krátké programy, poskytuje známá stránka 6 (PAGE 6). Rozsah paměti od $06ØØ do $06FF (1536 až 1791 dec.) je jištěn před nežádoucím přístupem BASICu, DOSu a operačního systému, ovšem díky častému používání je už obvykle přeplněna nějakýma pomocnýma rutinama.

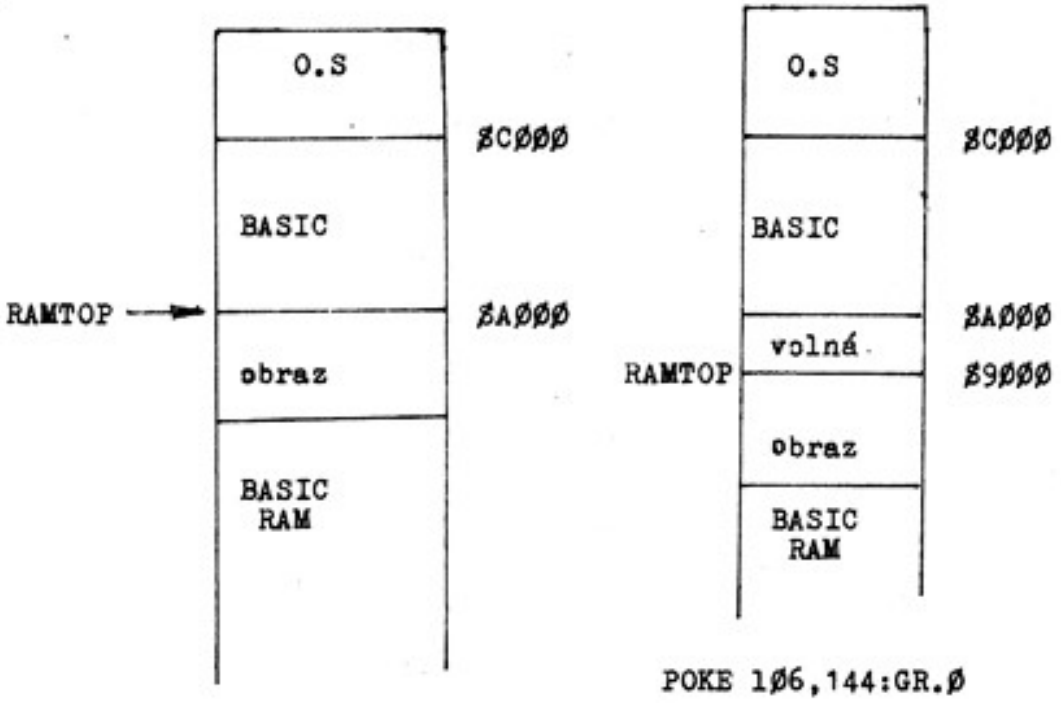
b) Uložení strojového programu do řetězce v BASICu. Je to elegantní možnost, ale hodí se jen pro relokativní (posouvatelné) rutiny v assembleru.

c) Nastavení RAMTOP ($6A, 1Ø6 dec.). Tímto způsobem můžeme rezervovat rozsah paměti nad RAM pro obrazovku. V RAMTOP je uloženo číslo poslední používané stránky plus jedna. V BASICu dává PEEK(1Ø6) hodnotu 16Ø (pro ATARI s minimálně 4Ø kByte), což znamená, že poslední používaná stránka je 159 ($9F).
Když, ale zapíšeme nižší hraniční stránku do RAMTOP a zadáme ještě příkaz GRAPHICS, tak se obrazová paměť i Dasplay-List posune na nižší adresu. Získáme tak chráněný rozsah paměti do nové hodnoty RAMTOP krát 256 až dokonce paměti.

Příklad: Chceme rezervovat z BASICu 4 kByte(16 stránek od $9ØØØ do $9FFF, min. 4Ø KB ATARI).

Musíme zadat:

```
POKE 1Ø6,16Ø - 16: GRAPHICS Ø
```



Když se teď podíváme v BASICu na využití paměti pomocí ?FRE(0), zjistíme že paměť pro BASIC je o 4 kByte menší. Nové rozdělení paměti je uvedeno na obrázku.

Při rezervování paměti je třeba mít na zřeteli tyto body:

-když je RAMTOP zvoleno tak, že obrazová paměť musí překročit 4kB - hranici (často při grafickém režimu náročném na paměť), nastane s hardwarových požadavků přeskupení zobrazování obrazu. Proto musíme měnit RAMTOP jen ve 4--kových krocích (1K), pro Hi-Res grafiku jen ve 16-kových krocích.

- Při stlačení SYSTEM RESET bude RAMTOP přepsán původní hodnotou. Problém lze řešit jenomzměnou systémové rutiny RESET (přes DOSINI), podrobně popsáno v soft. manuálu ATARI.

- Jakmile v průběhu programu v BASICu, dáme nový příkaz GRAPHICS, vynuluje se několik byte na začátku rezervovaného prostoru, což může způsobit nemilé překvapení, pokud se v tomto místě nachází strojový program. Proto první stránka paměti rezervovaná pomocí RAMTOP se nesmí používat !

d) Můžeme rovněž zvětšit MEMLO ($2E7, $2E8, L/H), ukazatele na nějnižší přístupné paměťové místo pro uživatele a tím uvolnit místo pro strojový program. Musíme to provést před inicializací BASICu, např.: v rámci souboru AUTORUN.SYS. Horní hranice DOSu v tomto případě začátek rezervované paměti, je ale pro různé verze DOSu různá. MEMLO bude při SYSTEM RESET nastaveno do původního stavu.

e) Pro modely XL/XE 64 kByte je ještě k dispozici od $C000 do $CFFF a od $D800 do $FFFF pod "ROM" skrytá paměť RAM. Není ale jednoduché ji použít, neboť je n nutné odpojit operační systém. K tomu je možné použít a přitom uvolnit uživateli paměť $C000 až $CFFF. Pro 400/800 nelze tento způsob použít.

U 130 XE máme k dispozici větší volný prostor, protože v okně od $40000 do $7FFFF můžeme ještě přepínat čtyři 16 k-RAM bloky. Musíme dávat pozor (byte $D301, je-li zapojen správný blok, když voláme program.

Automatické vytváření zavaděče v BASICu

Program DATGEN vám pomůže při převedení souboru v assembleru na BASIC-loader. Přitom se vytvoří program v BASICu, který můžete převzít pomocí ENTER do svého programu.

Počáteční a koncová adresa budou převzaty z assemblerovského souboru. Příkazem POKE budou data uvnitř smyčky FOR-NEXT umístěna na správné místo programu. Je rovněž vytvořena rutina pro kontrolní sumu. Pro domácí použití není nutná, ale je nezbytnou pomocí pro toho, kdo by chtěl váš program napsat z klávesnice. Data budou uložena do příkazu DATA. Délku překládaného řádku můžete umístit do proměnné ZLAENGE. Dále je možné sestavit více BASIC-loader programů do jednoho souboru. Stávající blok dat lze oslovit pomocí příkazu RESTORE (taktéž automaticky generovaného).

Návod: Odstartujte program DATGEN pomocí RUN, zadejte název binárního souboru ve standartních formátech ATARI (D: FN.EXT). Dále zadejte název generovaného BASIC-list souboru a počáteční číslo řádku. Po zadání bude zapsán na disketu LIST soubor. Můžete jej načíst do vlastního programu pomocí ENTER.

```
1Ø REM  DATGEN:Převod binárního souboru na BASIC-loader
3Ø DIM FØ(16),F1Ø(16)
4Ø ZLAEGE=65
1ØØ ? "Binar-File (D:FN.EXT)";: INPUT FØ
11Ø ? "List-File (D:FN.EXT)";:INPUT F1Ø
115 ? "Počáteční číslo řádku";: INPUT Z
12Ø OPEN #3,4,Ø,FØ: OPEN #4,8,Ø,F1Ø
13Ø GET #3,X1:GET #3,X2:REM čtení dvou byte určujících
                          typ souboru
14Ø IF X1<>255 OR X2<>255 THEN CLOSE#3;? "Není binár-
                          ní soubor": STOP
15Ø GET #3,AL:GET #3,AH:ANFADR=AL+AH*256:REM Počáteč.adr.
16Ø GET #3,AL: GET #3,AH:ENADR=AL+AH*256:REM Koncová adr.
2ØØ REM BASIC - rutina plnění s kontrolní sumou
```

```
21Ø ? #4;Z; "REM Plnění binárního souboru"
22Ø ? #4;Z+1Ø;"S=Ø:RESTORE ";Z+1ØØ
23Ø ? #4;Z+2Ø;" FOR A=";ANFADR;"TO";ENADR;":READ D:POKE
      A,D: S=S+D: NEXT A"
24Ø ?  4;Z+9Ø;"RETURN"
3ØØ REM Strojový program psaný jako data
31Ø ZN=Z+1ØØ: GOSUB 1ØØØ
33Ø L=L+LEN(STRØ(D))+1:IF L>ZLAENGE THEN ?#4:GOSUB 1ØØØ:
    GOTO 39Ø
34Ø IF A<>ENADR THEN ? #4;",";
39Ø NEXT A: ? #4
4ØØ ?#4;Z+3Ø;" IF S<> ";S;"THEN ?";CHRØ(34);"CHYBA DAT!";
      CHRØ(34);"STOP"
5ØØ CLOSE #3: CLOSE #4: END
1ØØØ REM Číslo řádku, výstup příkazu DATA
1Ø1Ø ?#4;ZN;"DATA";:L=LEN(STRØ(ZN))+6:ZN=ZN+1Ø:RETURN
```

P Ř Í L O H A : DOS XL kontra DOS-HK

Pokud jste si se zájmem přčetli předcházející kapitoly o DOS XL,ale bohužel jej nevlastníte, můžeme vám pomoci. Na programové diskete HK. přikládané ke knize, se nachází soubor AUTORUN.SYS, který velice dobře simuluje DOS XL.Můžete s ním vyzkoušet všechny postupy popsahé v knize. Naopak můžete všechno co jste napsali s DOS-HK použít také s DOS XL.

DOS-HK zná následující příkazy:

Příkaz	Význam	Příklad
DIR	Directory	DIR D2: P:
CAR	Cartridge	CAR
TYP	Type	TYP STATUO.EXC
PRO	Protect	PRO FILE.OBJ
UNP	Unprotect	UNP FILE.OBJ
REN	RENAME	REN ALT.OBJ NEU.OBJ
ERA	Erase	ERA FILE.OBJ
RUN	Run adress	RUN 4ØØØ
SAV	Save binary	SAV FILE.OBJ 4ØØØ 5ØØØ
LOA	Load binary	LOA FILE.OBJ
SCR	Screen	SCR

NOS	Noscreen	NOS
REM	Remark	REM *** Text
END	End execute	END
Dn:	Default drive	D2:
	Execute starten	STARTUP

Po DIR můžeme zadat dvě specifikace souborů, které jsou odděleny mezerou.
První udává způsob hledání pro speciální soubory (nebo disky), druhá určuje výstupní zařízení. V uvedeném příkladu bude vypsán seznam disku 2 na tiskárnu. DIR bez parametrů vydá directory základního driveru.

Příkazem TYPe můžeme vypsat textová data na obrazovku (např. TYP BCOM.SRC), na tiskárnu (TYP BCOM.SRC P:), nebo také překopírovat do jiného souboru (TYP BCOM.SRC D2:BCOM.SRC). ERA, REN, PRO, UNP pracují podobně jako u ATARI-DOS, takže není nutné je popisovat.

SAVE dovoluje záznam binárního souboru, k tomu musíme přidat za specifikaci souboru ještě počáteční a koncovou adresu, obě jsou očekávány hexadecimálně. Takto zaznamenaný soubor můžeme znovu načíst příkazem LOAD.

Příkazem RUN můžeme odstartovat strojový program od zadané adresy. Všechny číselné parametry musíme zadávat v hexadecimálním tvaru bez znaku $.

Příkazy @, SCR, NOS, REM a END slouží k řízení execute souborů.

Příkazem Dn: (n je číslo disku 1-8) bude změněn základní driver,rozumíme tím číslo disketové jednotky, která dodá DOS, nebylo-li číslo zadáno. Zadání D2 (RETURN) znamená že po DIR bude vypsán obsah z disketové jednotky 2 (přirozeně jen tehdy, máte-li drive 2). Přepnutí zpět provedeme příkazem D1: (RETURN).

DOS-HK vnitřně

využívá jádro FMS (manažér souborů DOS.SYS) z ATARI-DOS 2.5. Dále bude natažen soubor AUTORUN.SYS, který obsahuje takzvaný DOS-HK Consol Sommand Procesor, krátce CCP. CCP se zakotví do operačního systému ATARI, takže bude volán jako DOS. Když zadáte v BASICu "DOS", tak se nebude natahovat jako v DOS 2,5 nejprve DUP.SYS, ale dostanete se přímo do CCP. Přihlásí se poznámkou Copyright a očekává vaše zadání zprávou "D:"

DOS-HK může ještě více !

Může všechno, co zatím mohl DOS 2,5. Může používat režim zvýšené hustoty záznamu disketové jednotky 1Ø5Ø, stejně tak může obsluhovat bez problémů RAM-disk na 13Ø XE. Je to automatický zařízeno zavedením souboru RAMDISK.COM. S DOS-HK máme přednosti DOS-XL (CCP) a výhody DOS 2,5 (127 kB, RAM-disk pro 13Ø XE).

Duplikace DOS-HK

DOS-HK nemá, podobně jako DOS XL, zabudovaný příkaz pro inicializaci diskety. Protože pracuje na základě DOS 2,5 můžeme jej pomocí DOS 2.5 snadno duplikovat na další diskety.:

Postup je následující:

1. Natáhnout systémový tisk DOS 2.5
2. Prázdnou disketu formátovat zadáním OPTION "I "
3. Soubory DOS (DOS a DUP) nahrát na disketu pomocí OPTION "H"
4. DUP.SYS ppmocí OPTION "D" vymazat z diskety
5. Soubor AUTORUN.SYS z programové diskety HK zadáním OPTION "O" překopírovat na novou disketu - HOTOVO !

Když takto zhotovenou disketu zavedete bez BASICu, přejdete přímo do DOS-HK. Je-li BASIC aktivní, přejde program do BASICu.

O B S A H

VYDAL : ZO SVAZARM - ATARI klub Olomouc,
Klub elektroniky SES TLMAČE

NÁKLAD: 1 5ØØ kusů

N E P R O D E J N É

Neprošlo jazykovou úpravou. Přetisk
pouze se souhlasem redakce.

Předáno do tisku: 19. října 1987

TISK : MTZ Olomouc, n.p., provoz 42 Opava

TISK POVOLEN : OK ONV LEVICE č.j. 373/87
ze dne 13. 1Ø. 1987