zpravodaj atari klubu

VI

ZPRAVODAJ ATARI Klub Praha

Vydává 487. ZO Svazarmu — ATARI KLUB v Praze 4.
Šéfredaktor a vedoucí redakční rady JUDr. Jan Hlaváček.
Zástupce šéfredaktora ing. Stanislav Borský.
Obálku navrhl RNDr. J. Tamchyna.
Adresa redakce:
487. ZO Svazarmu - ATARI KLUB Praha
REDAKCE
poštovní přihrádka 51
100 00 P r a h a 1 0
Řídí redakční rada: V. Bílek, ing. J. Biskup, RNDr. J. Bok, CSc., ing. S. Borský, ing. V. Friedrich, ing. O. Hanuš, RNDr. L. Hejna, CSc., Z. Lazar, prom. fyz., CSc., ing. M. Vavrda.

Vychází šestkrát ročně. Neprodejné. Členům klubu distribuováno zdarma. Nepravidelné přílohy na objednávku jsou kompenzovány zvláštním klubovým příspěvkem.
Rozsah čísla 72 stran. Neprošlo jazykovou úpravou.
Tiskne Ústř. kulturní dům železničářů, Praha 2, nám. Míru 9
Do tisku předáno II/88
Vydávání schváleno OV Svazarmu Praha 4 a OŠK ONV Praha 4.
Evidenční číslo ÚVTEI 86 042.

příloha VI

S. P. LAWROW

MAC/65

Překlad
ing. Petr Jandík

Revize překladu
ing. Karel Šmuk

Předmluva

MAC/65 je zdokonalením předcházejícího produktu firmy OSS s názvem EASMD (Edit/ASsemble/Debug), který sám vycházel z modulu Atari Assembler/Editoru. Uživatelé obou těchto starších prostředků nebudou mít s přechodem na MAC/65 žádné potíže. Ti, kdo se dosud nesetkali s produkty této řadě firmy OSS bezpochyby zjistí, že MAC/65 se snadno používá, je výkonný a přizpůsobivý programovému okolí. I když rychlost nebyla při vývoji hlavním krédem, těžko by se na trhu domácích počítačů hledal rychlejší assembler. MAC/65 je vynikající nástroj s ohledem na velikost a výkon počítačů, pro které je určen.

MAC/65 navrhl a kompletně vypracoval Stephen D. Lawrow. Současná verze MAC/65 (dodávaná v modulu), je pouze poslední v řadě stále složitějších a rychlejších assemblerů, napsaných panem Lawrowem, sledujících linii EASMD. Měřítkem schopností MACu je to, že dokáže přeložit i sám sebe, což je jeden z nejtěžších úkolů, do kterého se lze pustit.

Předmluva překladatele

I když mají Američané ve zvyku při chvále svého zboží velmi přehánět, musím říci, že předmluva nelže. Při své programátorské praxi jsem se musel prokousat přes několik assemblerů, než jsem konečně došel k poslední verzi MACu, který mi vyhovuje téměř stoprocentně. Jeho rychost je hlavní devizou. Posuďte sami: program o rozsahu SpeedScriptu se v Assembler/Editoru kompiluje bez tisku listingu tři čtvrtě hodiny a jeho text nelze editovat, neboť se nevejde celý najednou do paměti. EASMD sice není o nic rychlejší, ale má již povel INCLUDE, takže lze text rozdělit do částí a editovat jej. Lepší je AMAC, Atari makroassembler, který zvládne tentýž program za 8 minut, ale občas se zboří, zatímco MAC zvládá totéž bez nejmenšího problému za necelé dvě minuty.

Kvality MACu byly důvodem, že jsem se rozhodl pustit se do tak obtížné práce, jakou je překlad referenční příručky, a poskytnout tak možnost práce se všemi jeho klady svým kolegům, uživatelům osmibitových počítačů Atari. Protože modul s poslední verzí MACu není dosud tak rozšířen, budu se v poznámkách zmiňovat o odlišnostech starší diskové verze a upozorňovat na povely a možnosti, které lze využít i u EASMD a Assembler/Editoru. Při překladech takovéto literatury se často naráží na nedostatek české odborné terminologie, takže překladatel musí často sám tvořit nové termíny. Doufám, že jsem se s tímto úkolem ve svém překladu vyrovnal se ctí. Přeji Vám příjemnou práci s MACem.

Petr Jandík

Úvod

Tento manuál předpokládá, že jeho uživatel ovládá jazyk symbolických adres neboli assembler. Není učebnicí assembleru. Tento manuál obsahuje popis povelů, příkazů, funkcí a syntaktických konvencí MAC/65. Dále se předpokládá, že uživatel ovládá standardní obrazovkový editor počítače Atari. Pokud tento editor neovládáte, prostudujte si nejprve příslušný manuál.

Potřebujete-li příručku typu učebnice, můžete sáhnout po knize, popisující procesor 6502. Z knih, beroucích ohled na počítače Atari lze doporučit "Machine Language for Beginners" od Richarda Mansfielda, publikovanou nakladatelsvím "COMPUTE! books", nebo "Programming the 6502" od Rodney Zackse. (Poznámka překladatele: z příruček, svépomocně vydaných v českém jazyce existuje publikace o procesoru 6502 od P. Dočekala z Ostravy nebo pro úplné začátečníky Průvodce assemblerem 6502, který podle německých pramenů sestavil A. Lindenthal a L. Podmolík. Za sebe bych doporučil hodnotný překlad učebnice assembleru pro počítače Atari od Marka Chasina, který pořídil Karel Šmuk).

Tento manuál je rozdělen do dvou hlavních sekcí. První dvě kapitoly popisují syntaxi a povely editoru, vstup zdrojové řádky a překlad zdrojového programu. Další tři kapitoly vysvětlují formát instrukce, direktivy assembleru, funkce a výrazy, makroinstrukce a podmíněný překlad.

Poznamenejme, že ladicí program DDT--the Dunion Debugging Tool--je popsán odděleně. (pozn.př.: DDT je součástí modulu MAC/65. Protože disketová verze MACu, která je u nás rozšířenější, neobsahuje tento program, není jeho popis v překladu manuálu zahrnut.)

MAC65 je rychlý a účinný prostředek pro vývoj strojových programů. Může přeložit i programy, přesahující rozsah paměti. MAC65 obsahuje také direktivy, navržené specificky pro práci s obrazovkou. MAC65 syntakticky kontroluje každou řádku v okamžiku zadání, čímž nás ušetří opakovaných překladů z důvodu chybné syntaxe. Můžeme tím pádem věnovat více času na vývoj samotného programu.

Zapnutí

Vypněte počítač a vsuňte modul MAC65 do otvoru pro moduly. Používáte-li diskovou jednotku, vložte do jednotky číslo 1 disketu s příslušným DOSem (DOS 2.5, DOS XL, apod.) a ujistěte se, že je jednotka zapnutá.

Zapněte počítač. Je-li disková jednotka zapnutá a je-li v ní správná disketa, natáhne se z ní DOS do paměti. V závislosti na druhu a verzi DOSu, který používáte, bude možná nutné zadat povel k přechodu do modulu MAC65. Je-li tomu tak, zadejte příslušný povel.

Nyní uvidíte na obrazovce název MAC/65 a informace o copyrightu, pod nimiž je identifikační výpis "EDIT". Pokud tomu tak není, prostudujte příručku DOSu a zapojení disku a zkuste to znovu.

Nyní jste připraven používat MAC/65.

poznámka překladatele: spuštění disketové verze MAC/65 se provede podobně, až na zasunutí modulu. Po načtení DOSu se na obrazovce objeví symbol "D1:". Za něj napíšeme povel "MAC65". Program se načte a spustí. Po přechodu do DOSu se do MAC65 vrátíme na rozdíl od verze v modulu povelem RUN. Disketová verze MAC65 nefunguje s DOSem typu 2.0 a 2.5, které jsou rozděleny do části rezidentní a části menu (DOS.SYS a DUP.SYS). Použitelné jsou DOSy s povelovou řádkou: DOS XL, OS/A+ a SPARTA DOS verze 3.2, který jako jediný z nich umožňuje práci s rozšířenou hustotou záznamu na diskové jednotce 1050.

Teplý start

Uživatel může povelem DOS (následovaným <RETURN>em) přejít z MAC/65 do DOSu. Návrat do MACu je možný povelem CAR <RETURN> (nebo v menu povelem T). Pokud jste v DOSu nepoužili externí povely, můžete se do MACu dostat "teplým startem" (tedy bez vymazání textu v paměti). Detaily lze najít v manuálu DOSu. Při používání DOSu ATARI funguje MAC/65 podobně, jako každý jiný modul. Povel DOS nás převede do DOS menu a povel "B" nás vrátí opět do MACu. Pokud používáte soubor MEM.SAV, program v paměti zůstane zachován. Podrobnosti lze nalézt v příručce DOSu 2.5

pozn.př: Při používání diskové verze MACu platí povel "DOS" stejně, jako pro modul. Zpětný přechod do MACu se na rozdíl od verze v modulu provede povelem "RUN". Povely DOSu ATARI není třeba diskutovat, protože disková verze s DOSem ATARI nefunguje.

Syntaxe

V syntaktických popisech tohoto manuálu se používají následující konvence:

1. Velká písmena označují ty povely, instrukce, funkce apod., které je třeba uvádět přesně tak, jak jsou napsány (např. ENTER, .INCLUDE, .NOT). (viz ještě poznámka dole.)

2. Malá písmena specifikují různé parametry. Mohou být následujících typů:

lno	- číslo řádky mezi 0 - 65535 včetně
hxnum	- hexadecimální číslo. Mohou to být adresy, nebo data. Hexadecimální čísla se pokládají za celá čísla bez znaménka.
dcnum	- kladné číslo. Dekadická čísla jsou zaokrouhlena na nejbližší dvoubytové celé číslo bez znaménka; 3.5 se zaokrouhluje na 4 a 100.1 na 100.
exp	- výraz.
string	- Retězec ASCII znaků, uzavřený v uvozovkách. (například: "THIS IS A STRING")
strvar	- Reprezentace stringu. Může být nahrazena řetězcem, jako nahoře, nebo stringovou proměnnou ve volání makroinstrukce (např. %$1).
filespec- nebo file	Retězec ASCII znaků, definující určité periferní zařízení, a eventuelně soubor dat. Podrobnosti lze najít v manuálu příslušného zařízení.

3. Parametry v hranatých závorkách označují nepovinnou část syntaxe. (například [,lno]). Je-li nepovinný parametr následován třemi tečkami (...), parametr (parametry) může (mohou) být opakován(y) tolikrát, kolikrát je třeba:

Příklad: .WORD exp [,exp ...]

4. Parametry v kulatých závorkách znamenají, že může být použit jen jeden z nich, např. (,0) (,A).

Poznámka: V EDIT modu nerozlišuje MAC/65 malá a velká písmena. Malá písmena se konvertují na velká, inverzní znaky se neinvertují. VÝJIMKY: znaky mezi uvozovkami, za apostrofem, nebo v poli komentáře se nemění. Nemění se ani text, napsaný v TEXT modu.

KAPITOLA 1: EDITOR

Editor umožňuje uživateli vkládat a editovat zdrojový kód MAC/65, nebo běžné textové soubory v kódu ASCII.

Editor odlišuje oba typy souborů natolik, že lze hovořit o dvou modech, které může uživatel použít, EDIT modus a TEXT modus. U obou forem však musí řádka začínat číslem mezi 0 a 65535 včetně, následovaným jednou mezerou.

```
příklady: 10 LABEL LDA #$32
          3020 Tohle platí v TEXT modu
```

První příklad platí jak v EDIT, tak v TEXT modu, zatímco druhý jen v modu TEXT. Uživatel si volí mezi oběma mody buď povelem NEW (který inicializuje EDIT modus), nebo TEXT (který dovoluje psaní libovolného textu). V dalším budou vlastnosti obou modů rozebírány častěji, nyní jen několik bodů společných oběma.

1.1 VSEOBECNÉ POUZITÍ EDITORU

Se zdrojovým souborem manipulujeme pomocí povelů editoru. Protože editor rozpoznává povely tím, že nemají číslo řádky, řádka, začínající číslem, se pokládá za platnou zdrojovou/textovou řádku. Tato je zatříděna, přidána, nebo vložena mezi zdrojové/textové řádky podle svého čísla. Vkládaná řádka, která má stejné číslo jako některá řádka v paměti, tuto řádku nahradí. Specielním případem nahrazení je vymazání řádky z paměti napsáním samotného čísla řádky. (Viz též povel DEL, kterým lze vyřadit větší skupinu řádek.)

Řádka, nezačínající číslem, se pokládá za povelovou řádku. Editor tuto řádku prohlíží, aby určil funkci, která se má provést. Je-li řádka platným povelem, editor jej provede. Editor oznamuje provedení nebo ukončení každého povelu vypsáním výzvy:

```
Edit  pro syntaktický (MAC/65 zdrojový) modus
Text: pro textový modus
```

Kurzor se nalézá na následující řádce. Protože některé povely mohou trvat určitý čas, výzva oznamuje uživateli, že je možné vkládat další povely. Povel může uživatel zrušit ještě před dokončením klávesou BREAK.

Poslední poznámka: Není-li řádka ani povel, ani zdrojová řádka, napíše editor:

```
WHAT?
```

1.2 TEXT modus

Editor umožňuje textový modus. Textový modus se zapíná povelem TEXT. Tento modus NEPROVÁDÍ syntaktickou kontrolu vložených řádek, což umožňuje vkládat a editovat i jiné soubory, než assemblerovské. V textovém modu funsují všechny povely editoru. Pamatujme však na to, že všechny řádky musí začínat číslem a mezera za ním je povinná i v textovém modu.

1.3 EDIT modus

MAC/65 je asi jediným systémem assembleru/editoru, který syntakticky kontroluje každou vloženou řádku před uložením do paměti. Díky širokým možnostem v syntaxi assembleru (zvláště jsou-li povoleny makroinstrukce), nelze syntaktickou kontrolou zjistit všechny chyby v uživatelově zdrojovém kódu. Například existenci návěští nebo adres v nulové stránce lze zkontrolovat až během překladu. Přesto si však myslíme, že tato okamžitá syntaktická kontrola je výhodná jak pro začátečníka, tak pro zkušeného programátora.

Znovu připomínáme, že zdrojová řádka musí začínat číslem řádky, za nímž musí být mezera. Za touto mezerou je pole návěští. Není-li návěští uvedeno, musí zde být další mezera. Instrukce mohou začínat v kterémkoliv dalším sloupci, minimálně jednu mezeru za návěštím. Operand může začínat kdekoliv za instrukcí a komentář kdekoliv za operandem nebo instrukcí. Syntaxe instrukcí je popsána v kapitole 3 o assembleru.

Jak již bylo poznamenáno, editor kontroluje při vkládání každou zdrojovou řádku. Je-li zjištěna chyba, editor vyříše řádku s kurzorem (inverzní mezerou), umístěným na místě chyby.

Poté je zdrojová řádka tokenizována a uložena v paměti. Řádky se umisťují odzdola nahoru. Výsledný tokenizovaný soubor je o 60 až 80% menší než stejný soubor ve formě kódu ASCII, což dovoluje umístit do paměti větší soubory a editovat je.

ZVLÁŠTNÍ POZNÁMKA: Pokud zdrojová řádka obsahuje při vložení chybu, je označena jako chybná. Přesto je editorem uložena do paměti. Tato vlastnost umožňuje uživateli vložit do paměti surový ASCII text (například z jiných assemblerů, obsahující eventuelně chyby z hlediska MAC/65) bez ztráty řádek. Uživatel si může řádky s chybami prohlédnout a opravit je později.

KAPITOLA 2: POVELY EDITORU

Tato kapitola obsahuje všechny platné povely editoru v abecedním pořadí s krátkým popisem účelu a funkce každého z nich. Znovu připomínáme, že každá řádka bez čísla je pokládána za povel editoru. Nastane-li během provádění povelu editoru chyba, editor přeruší práci a vypíše číslo a popis chyby před opětnou výzvou uživateli. Možné příčiny chyb jsou uvedeny v příloze.

odstavec 2.1

povel editoru: ASM

účel: překlad zdrojových souborů MAC/65 (ASseMble)

použití: ASM [#file1],[#file2],[#file3],[#file4]
nebo přesněji:
ASM [[#file1][,[#file2][,[#file3][,#file4]]]]

ASM přeloží specifikovaný zdrojový soubor a vytvoří listing (výpis programu) a výsledný strojový program (binární soubor, objekt ...); listing může obsahovat tabulku křížových odkazů všech nelokálních návěští. File1 je zdrojový soubor, file2 je zařízení pro výpis programu, file3 je zařízení pro uložení strojového programu a file4 je dočasný soubor pro generování tabulky křížových odkazů.

Kterýkoliv ze čtyř parametrů může být vynechán. V takovém případě předpokládá MAC/65 následující standardní volbu:

file1 - paměť (se zdrojovým programem)
file2 - obrazovka
file3 - paměť (POZOR: viz poznámka)
file4 - žádný, tím pádem se negeneruje tabulka

Specifikaci souboru (#file1, #file3, atd.) můžeme vynechat. V tomto případě bude specifikace nahrazena standardní volbou. Oddělující čárky však vynechat nelze.

Pro soubor listingu (a pouze pro něj) lze použít speciální formu "#-", což indikuje, že výpis programu nechceme.

Některé příklady:

Příklad: ASM #D2:SOURCE,#D:LIST,#D2:OBJECT

V tomto případě bude zdrojovým souborem D2:SOURCE, assembler vypíše soubor do D:LIST a zkompilovaný program se zapíše do D2:OBJECT.

Příklad: ASM , #P:, , #D:TEMP

V tomto případě se bude zdrojový soubor číst z paměti, přeložený program se bude zapisovat do paměti (ale jen v případě, je-li ve zdrojovém programu direktiva ".OPT OBJ"), a výpis programu i s tabulkou křížových odkazů se vytiskne na tiskárně. Soubor TEMP na disku 1 se založí a použije jako dočasný soubor pro křížové reference.

Příklad: ASM #D:SOURCE , #P:

V tomto případě se bude číst zdroj ze souboru D:SOURCE a výpis programu půjde na tiskárnu. Pokud je ve zdrojovém programu direktiva ".OPT OBJ", uloží se zkompilovaný program do paměti.

Příklad: ASM , #-

Tento příklad produkuje nejrychlejší možný překlad. Zdrojový program se čte z paměti, a výpis se nevytváří (kvůli "#-"). Pokud program neobsahuje ".OPT OBJ", jde o překlad pro zjištění chyb. (Pokud je zmíněná direktiva obsažena, je překlad extrémně rychlý.)

ZVLÁŠTNÍ POZNÁMKY

Poznámka: Jestliže se překlad provádí ze souboru, musí se jednat o soubor, uložený příkazem SAVE.

- podívejte se na direktivu .OPT, která řídí výstup listingu a přeloženého programu.

- Přeložený program bude mít formát složeného binárního souboru, který se v DOSu XL vytváří povelem SAVE, v DOSu 2.5 povelem K (Binary Save). Prostudujte formát souboru v příručce DOSu.

- Pro listing nebo přeložený program lze jako výstupní zařízení použít #C:. Kazetu však nelze použít pro zdrojový program, ani pro pracovní soubor křížových referencí (z toho implicitně vyplývá, že bez diskové jednotky nemůžeme dosáhnout křížových odkazů). Nedoporučujeme používat kazetu jako soubor pro přeložený program, protože může vzniknout příliš dlouhý zaváděcí tón, ztěžující načtení programu povelem BLOAD. Místo toho použijte raději překlad do paměti s následným uložením povelem BSAVE.

odstavec 2.2

povel editoru: BLOAD

účel: umožňuje uživateli zavést do paměti binární soubor (strojový program) z disku nebo kazety

použití: BLOAD #filespec

Povel BLOAD zavede do paměti soubor, který byl vytvořen povelem BSAVE, překladem do souboru, nebo povelem SAVE v OS/A+, nebo DOS XL, nebo funkcí K v DOSu 2.5

Příklad: BLOAD #D:OBJECT

Tento povel zavede do paměti binární soubor "OBJECT" od adresy, ze které byl dříve uložen nebo od které byl přeložen assemblerem.

Příklad: BLOAD #C:

Tento příklad slouží k načtení binárního souboru z kazety.

UPOZORNENI: Vřele se doporučuje zavádět povelem BLOAD pouze takové soubory, které byly MAC/65 přeloženy do volné paměti (kterou zjistíme povelem SIZE), nebo který se zavádí do nám známých bezpečných oblastí paměti.

odstavec 2.3
---

povel editoru: BSAVE

účel: Ukládá na definované zařízení určenou část paměti. Je ekvivalentní povelu SAVE OS/A+ a K v DOS 2.5.

použití: BSAVE #filespec < hxnum1 ,hxnum2

Povel BSAVE ukládá obsah paměti od adresy hxnum1 do adresy hxnum2 na specifikované zařízení. Vzniklý binární soubor je kompatibilní s formátem ukládacích povelů pro binární soubory DOSu 2.5 i OS/A+ a DOS XL.

Příklad: BSAVE #D:OBJECT<5000,5100

Tento povel uloží obsah paměti od $5000 do $5100 na disk do souboru "OBJECT".

Příklad: BSAVE #C: <5000,5100

Tentýž úsek paměti se uloží na kazetu.

odstavec 2.4
---

povel editoru: BYE

účel: přechod do selftestu

použití: BYE

Pomocí BYE se u modelů 800 XL a 130 XE a novějších dostaneme do selftestu. U modelů řady 400/800 se dostaneme do modu Blackboard.

odstavec 2.5
---

povel editoru: DDT

účel: přechod do ladicího programu DDT, který je součástí modulu MAC/65.

použití: DDT

Po zadání toho povelu přebírá řízení ladicí program DDT.

Při zachování určitých pravidel lze přejít zpět do MAC/65, aniž by tím byl dotčen zdrojový program v paměti.

Manuál DDT obsahuje všechny potřebné informace, ale jako

všeobecné pravidlo platí, že nesmíme používat adresy $80 až $AF v nulové stránce a paměťové oblasti ležící uvnitř intervalu, zobrazeného povelem SIZE.

Další podrobnosti jsou v manuálu programu DDT.

pozn.př.: V disketové verzi programu tento povel nelze použít. Pro odlaďování strojových programů je nutno použít některý z nezávislých ladicích programů (např. DEBUG+ Bryana Schapella), nebo použít DEBUG z EASMD, nebo Assembler/Editoru. U posledních dvou se ladicí program aktivuje povelem BUG.

odstavec 2.6

povel editoru: DEL

účel: vymazání řádky, nebo skupiny řádek ze zdroje/textu v paměti

použití: DEL lno1 [,lno2]

DEL vymaže z paměti zdrojové řádky. Je-li uvedeno pouze lno1, vymaže se pouze jedna řádka. Jsou-li zadána obě čísla, vymažou se řádky od lno1 do lno2 včetně.

Poznámka: Aby povel proběhl, musí být řádka lno1 v paměti obsažena.

Příklady:

DEL 100	vymaže řádku 100
DEL 200,1300	vymaže řádky 200 až 1300 včetně

odstavec 2.7

povel editoru: DOS [nebo ekvivalentně CP]

účel: přechod z MAC/65 do DOSu

použití: DOS
nebo
CP

DOS nebo CP nás vrátí do DOSu. Pokud pracujeme v DOS 2.5, objeví se nám DOS menu. Pracujeme-li v DOS XL, dostaneme se buď do DOS XL menu, nebo do CP (Command Processor) podle toho, odkud jsme se do MAC/65 dostali.

V úvodu tohoto manuálu jsou uvedeny informace o teplém i studeném startu MAC/65 a povelech DOSu.

odstavec 2.8

povel editoru: ENTER

účel: vstup textových ASCII (nebo ATASCII) souborů do paměti editoru MAC/65

použití: ENTER #filespec [(,M) (,A)]

ENTER přiměje editor k načtení ASCII textu ze specifikovaného zařízení. ENTER nejprve vyčistí textovou paměť. Proto je uživatelův program v okamžiku zadání povelu ENTER vymazán.

Parametr "M" (MERGE) způsobí, že MAC/65 NEvymaže před načítáním textu ze zařízení paměťovou oblast editoru. Text, načtený ze zařízení, se podle čísel řádek zatřídí do stávajícího textu. Pokud má řádka, přicházející ze zařízení, stejné číslo, jako řádka v paměti, přepíše vstupující řádka řádku stávající.

Parametr "A" umožní uživateli načíst ze zařízení nečíslovaný text. Editor začne číslovat číslem 10 s přírůstkem rovněž 10.

POZOR: Volba "A" vždy vymaže před načítáním textovou paměť. Proto se nesmí použít "M" a "A" současně!

odstavec 2.9

povel editoru: FIND

účel: vyhledání textového řetězce v textové paměti editoru

použití: FIND /string/ [lno1 [,lno2]] [,A]

Povel FIND vyhledá v paměti všechny řádky s hledaným textem v rozmezí lno1 až lno2. Hledaný text je umístěn mezi omezovači, kterými může být kterýkoliv znak kromě mezery. Pokud se text najde, vypíše se řádka na obrazovce.

Poznámka: Hledaný text se neuzavírá do úvozovek.

Příklad: FIND/LDX/

Tento příklad najde první výskyt "LDX".

Příklad: FIND|Label|25,80

Tento příklad najde první výskyt slova "Label" mezi řádky 25 a 80.

Je-li uveden parametr "A", vypíší se všechny řádky s hledaným textem v daném rozsahu řádek. Pokud čísla řádek neuvedeme, hledá se v celém programu.

odstavec 2.10
---

povel editoru: LIST

účel: výpis části, nebo celé textové paměti editoru MAC/65 ve formě ASCII (ATASCII) na dané zařízení.

použití: LIST [#filespec,] [lno1 [,lno2]]

LIST vypisuje zdrojový program na obrazovku, nebo na zařízení, pokud je uvedeno #filespec. Nejsou-li uvedena čísla řádek, vypisuje se celý program, uložený v paměti.

Je-li specifikováno jen první číslo řádky, vypíše se tato řádka, pokud je v paměti. Zadáme-li obě čísla řádek, vypisuje se od řádky s číslem větším, nebo stejným jako lno1, a skončí, je-li číslo řádky větší, než lno2.

Příklad: LIST #P:
vypíše aktuální obsah paměti editoru na tiskárnu.

Příklad: LIST #D2:TEMP, 1030, 1800
vypíše řádky, ležící v rozsahu 1030 až 1800 včetně, do diskového souboru "TEMP" na disku 2.

Poznámka: Druhý příklad ukazuje metodu přesunu nebo duplikování velkých částí textu pomocí dočasných diskových souborů. Lze jich dosáhnout přečíslováním textu v paměti před a po LISTu s následným použitím ENTER s volbou MERGE.

odstavec 2.11
---

povel editoru: LOAD

účel: načtení tokenizovaného zdrojového programu, uloženého povelem MAC/65 SAVE

použití: LOAD #filespec [,A]

LOAD zavede do paměti editoru tokenizovaný soubor, vytvořený povelem MAC/65 SAVE. LOAD před zaváděním vyčistí textovou paměť, pokud není zadán parametr "A".

Parametr "A" (APPEND) způsobí, že editor textovou paměť nevyčistí a připojí načítaný soubor k obsahu paměti.

Poznámka: Volba APPEND nepřečíslovává zdrojový program v paměti. Tím mohou vzniknout duplicitní čísla řádek. Duplicity lze odstranit povelem "REN".

odstavec 2.12
---

povel editoru: LOMEM

účel: změna dolní hranice paměti, používané editorem MAC/65

použití: LOMEM hxnum

LOMEM umožňuje určit hranici paměti, od které se ukládá zdrojový program.

POZOR! Povel LOMEM vymaže z paměti jakýkoliv zdrojový program, stejně, jako by byl zadán povel "NEW".

odstavec 2.13
---

povel editoru: NEW

účel: vymazání paměti editoru a nastavení modu syntaktické kontroly

použití: NEW

NEW vymaže z paměti veškerý zdrojový kód. Objeví se výzva "Edit", oznamující, že byl aktivován modus syntaktické kontroly. Pokud uživatel potřebuje syntaktickou kontrolu vyřadit, musí použít povel TEXT.

odstavec 2.14
---

povel editoru: NUM

účel: aktivuje modus automatického číslování řádek

použití: NUM [dcnum1 [,dcnum2]]

NUM způsobí, že editor začne automaticky číslovat text, zadávaný z klávesnice. Za číslem řádky se automaticky vypíše povinná mezera. Nejsou-li specifikována čísla dcnum 1 a 2, začíná se číslovat od 10-ti s přírůstkem 10. Při zadání dcnum1 se čísluje od první volné řádky s přírůstkem dcnum1. Není-li v paměti text, čísluje se od 10. Při zadání obou čísel se čísluje od dcnum1 s přírůstkem dcnum2.

Příklad: NUM 1000,20
způsobí, že editor vyzývá uživatele ke vložení řádky výzvou "1000", za kterou následuje mezera. Po vložení řádky následuje výzva "1020 " atd.

Modus NUM končí, když se číslo řádky, která má být další na řadě, vyskytuje v paměti.

NUM modus lze ukončit klávesou BREAK, nebo CONTROL-3. Také je možno použít CONTROL-C, následované <RETURN>em.

odstavec 2.15
---

povel editoru: PRINT

účel: tisk celého textu, nebo jeho části na zvolené zařízení.

použití: PRINT [#filespec,] [lno1 [,lno2]]

PRINT funguje přesně stejně jako LIST s tím rozdílem, že netiskne čísla řádek. Jestliže vytiskneme soubor na disk, můžeme jej načíst zpět povelem ENTER s parametrem "A".

odstavec 2.16
---

povel editoru: REN

účel: přečíslování všech řádek v paměti editoru

použití: REN [dcnum1 [,dcnum2]]

REN přečísluje zdrojové řádky v paměti. Není-li uvedeno dcnum1, ani dcnum2, přečíslování začíná od 10-ti s přírůstkem 10. REN dcnum1 začíná číslování od 10 s přírůstkem dcnum1. REN dcnum1,dcnum2 začíná číslovat od dcnum1 s přírůstkem dcnum2.

odstavec 2.17
---

povel editoru: REP

účel: nahrazuje zadaný text jiným zadaným textem

použití:
REP /starý text/nový text/ [lno1 [,lno2]] [(,A) (,Q)]

Povel REP vyhledá v rozsahu specifikovaných řádek "starý text".

Parametr "A" způsobí, že všechny výskyty "starého textu" se nahradí "novým textem". Parametr "Q" vypíše řádku s nalezeným "starým textem" a požaduje povolení změny ("Y" a <RETURN> znamená změnu, <RETURN> přeskočení tohoto výskytu). Není-li udáno ani "A", ani "Q", nahradí se "novým textem" jen první výskyt "starého textu". Po provedení změny je řádka vždy vypsána.

Příklad: REP/LDY/LDA/200,250,Q

V tomto příkladu se vyhledá "LDY" mezi řádky 200 až 250 včetně a při každém výskytu žádá povolení změny.

Poznámka: Stisknutí klávesy BREAK ukončí nahrazování a vrátí nás do editoru.

Poznámka: Způsobí-li změna syntaktickou chybu, ukončí se nahrazovací modus a řízení převezme editor. V textovém modu se ovšem syntaktická kontrola neprovádí, proto v něm k této situaci nemůže dojít.

odstavec 2.18

povel editoru: SAVE

účel: uložit tokenizovaný zdrojový program/text z paměti na specifikované zařízení

použití: SAVE #filespec

SAVE uloží tokenizovaný (kódovaný) zdrojový program/text uživatele na specifikované zařízení. Formát tokenizovaného souboru je následující:

hlavička souboru
dvoubytové číslo (dolní byte, horní byte) určuje velikost souboru v bytech.

struktura řádky v souboru:
dvoubytové číslo řádky (dolní-horní), následované jedním bytem délky řádky (vlastně ukazatelem na další řádku) následovaným tokenizovanou řádkou.

odstavec 2.19

povel editoru: SIZE

účel: zjistí a vypíše velikost různých paměťových oblastí, používaných editorem MAC/65

použití: SIZE

SIZE vypíše dolní a horní hranici paměti, obsazené zdrojovým programem, a nejvyšší adresu paměti, použitelnou editorem, v uvedeném pořadí. Čísla jsou uvedena hexadecimálně.

Tyto adresy jsou důležité při stanovení oblastí paměti, které se nesmí použít při kompilaci přímo do paměti. Pamatujme, že MAC/65 potřebuje určitý úsek paměti nad středním číslem pro tabulku symbolů (při kompilaci). Podrobnosti o použití paměti jsou uvedeny v manuálu DDT. Obecně se nedoporučuje používat při kompilaci přímo do paměti dolní polovinu stránky 6.

odstavec 2.20

povel editoru: TEXT

účel: umožňuje vstup libovolného ASCII (ATASCII) textu bez syntaktické kontroly

použití: TEXT

TEXT vymaže paměť editoru a přepne jej do textového modu. Po tomto povelu vyzývá editor uživatele k zadání povelu nebo textu výzvou "Text:" (místo "Edit"), což indikuje, že se neprovádí syntaktická kontrola.

TEXTMODE lze ukončit povelem NEW.
Pozor: Neexistuje způsob, jak přejít z modu syntaxe (Edit) do textového modu nebo zpět bez ztráty obsahu paměti editoru.

odstavec 2.21

povel editoru: ?

účel: konverze dekadických/hexadecimálních čísel

použití: ? ($hxnum) (dcnum)

? je rezidentní dekadicko-hexadecimální a hexadecimálně-dekadický konvertor. Převádět lze čísla v rozsahu 0 - 65535 dekadicky ($0000 až $FFFF hexa).

Příklad: ? $1200 napíše =4608
? 8190 napíše=$1FFE

KAPITOLA 3: MAKROASSEMBLER

Překladač assembleru se v MAC/65 aktivuje povelem ASM. Syntaxe tohoto povelu je uvedena v odstavci 2.1 (mezi povely editoru). Překlad lze ukončit klávesou BREAK. MAC/65 zavře před ukončením překladu korektně všechny soubory a "uklidí" po sobě.

3.1 Vstup do překladače

Překladač assembleru čte ze specifikovaného vstupního zařízení nebo z paměti editoru řádku po řádce. Pokud čte ze zařízení, musí být čtený soubor vytvořen povelem editoru SAVE (čili musí jít o tokenizovaný soubor). Zdrojové řádky a jejich syntaxe jsou vysvětleny v kapitole o editoru. Tokenizovaná forma je všeobecně vysvětlena v odstavci 2.19 u povelu SAVE.

Zdrojové řádky mají následující formu:

číslo řádky + povinná mezera + zdrojový příkaz

Zdrojový příkaz může mít jednu z následujících forem:

[návěští] [(instrukce 6502) (direktiva)] [komentář]

Toto je ukázka platných zdrojových řádek (bez nároku na smysluplnost jejich významu):

```
100 LABEL
120 ;Radka komentare
140  LDA #5      a dalsi komentar
150  DEY
160  ASL A       zdvojeni cisla v akumulatoru
170 GETNUM LDA (ADDRESS),Y
180  .PAGE "direktivy jsou take povoleny"
```

Obecně je formát stejný jako formát doporučený v programovací příručce výrobce procesoru (MOS Technology 6502 Programming Manual). Uživateli, který neovládá programování procesoru 6502 lze doporučit z anglické literatury:

"Machine Language for Beginners" od R. Mansfielda
nebo
"Programming the 6502" od Rodney Zaksse.

pozn.př.: z u nás dostupných příruček existuje popis instrukcí 6502, přeložený P. Dočekalem, nebo lepší "Programování v Assembleru pro počítače Atari" od Marka Chasina, kterou velmi dobře přeložil Karel Šmuk.

Zvláštní poznámka: Překladač MAC/65 bere v úvahu jen návěští, mnemoniky apod., psané velkými písmeny. Protože však editor konvertuje (viz odstavec 1.3) všechna malá písmena na velká (kromě komentářů a textů v uvozovkách), může uživatel psát podle svého gusta jak malými, tak velkými písmeny.

3.2 Formát instrukcí

A) Mnemoniky instrukcí jsou převzaty z programovací příručky MOS Technology Programming Manual.

B) Přímé operandy začínají znakem "#".

C) "(operand,X)" a "(operand),Y" označují indexovaně nepřímé a nepřímo indexované adresování.

D) "operand,X" a "operand,Y" označují indexované adresování

E) Odkazy na adresy v nulové stránce nemohou předcházet jejich definici. Výsledkem toho bývá chyba "PHASE ERROR".

F) Dopředná přiřazení se vyhodnocují v rámci dvouprůchodového překladu.

G) "*" označuje čítač momentální adresy

H) Řádky komentáře mohou začínat ";", nebo "*".

I) Středník v kterékoliv pozici řádky označuje začátek pole komentáře

J) Hexadecimální konstanty začínají vždy znakem "$".

K) Operand "A" je vyhrazen pro adresování akumulátoru.

L) Přípustné adresovací formáty jsou rozšířeny o nové způsoby adresování, které umožňuje nový mikroprocesor 65C02 firmy NCR. Nové instrukce, které nemá procesor 6502 jsou uvedeny v kapitole 7. Rozšíření obsahuje:

1. "(operand)", znamenající nepřímé adresování, které je nyní povoleno u ADC, AND, CMP, EOR, LDA, ORA, SBC a STA. Operand musí být v nulové stránce.

2. "(operand,X)" je nyní povoleno u JMP. Operandem zde může být libovolná absolutní adresa.

3. U instrukce BIT je nyní povolen adresovací modus "operand,X". Operandem může být jak absolutní adresa, tak adresa v nulové stránce.

4. Jsou povoleny nové mnemoniky BRA, DEA, INA, PHX, PHY, PLX, PLY, STZ, TRB a TSB.

pozn.př.: Nepleťte si procesor 6502C, kterým jsou vybavovány počítače ATARI, s procesorem 65C02. 6502C je programově stejný, jako 6502, zatímco 65C02 je rozšířen o nové možnosti.

3.3 Návěští

Návěští musí začít písmenem nebo znaky "@" nebo "?". Na dalších pozicích mohou být navíc číslice 0 až 9, nebo ".". Písmena musí být velká (což nám zajistí editor) a návěští nesmí být přerušeno mezerou. Maximální počet znaků v návěští je 127 a VŠECHNY jsou platné.

Návěští, začínající otazníkem ("?") se pokládají za lokální a jsou přístupná pouze kódu uvnitř příslušné lokální oblasti. Lokální oblasti jsou omezeny postupným uvedením direktivy .LOCAL s tím, že první lokální oblast začíná na začátku programu, ať již byla direktiva .LOCAL uvedena, nebo ne. V jednom programu může být maximálně 62 lokálních oblastí. Pokud se v programu direktiva .LOCAL neuvede, jsou všechna návěští dostupná odevšud. Návěští, začínající otazníkem, se v žádném případě neuvádějí v tabulce symbolů.

Následují příklady platných návěští:

```
TEST1  @.INC  LOCATION  LOC22A WHAT?
ADDRESS1.1  EXP..  SINE45TAB.
```

3.4 Operandy

Operandem může být návěští, parametr makroinstrukce, číselná konstanta, čítač momentální adresy (program counter "*"), "A" pro adresování akumulátoru, výraz nebo ASCII znak, před nímž musí být apostrof. Následují příklady různých typů operandů:

```
10      LDA     #VALUE      ;návěští
20      ROR     A           ;adresování akumulátoru
30      .BYTE   123,$45     ;číselné konstanty
40      .IF     %0          ;parametr makra
45      CMP     #'A         ;znak ASCII
50 THISLOC  =   *           ;programový čítač
55      .WORD   PMBASE+[PLNO+4]*256   ;výraz
```

3.5 Operátory

MAC/65 povoluje použití následujících operátorů:

```
[ ]     pseudozávorky
+       sčítání
-       odčítání
/       dělení
\       modulo (zbytek po celočíselném dělení)
*       násobení
&       binární AND
!       binární OR
^       binární EOR

=       logická rovnost
>       logické větší než
<       logické menší než
<>      logická nerovnost
>=      logické větší, nebo rovno
<=      logické menší, nebo rovno
.OR     logické OR
.AND    logické AND

-       unární minus
.NOT        unární, logická negace. Výsledkem je "pravda" (1), je-li
výraz
        nulový. Je-li výsledek výrazu nenulový, vrací nepravdu (0).
.DEF    unární logický operátor, definice návěští. Vrací hodnotu
        "pravda", je-li návěští definováno.
.REF    unární logický operátor. Vrací "pravda", je-li v programu
        odkaz na návěští.
>       unární operátor. Vrací horní byte 16-ti bitového výrazu.
<       unární operátor. Vrací dolní byte 16-ti bitového výrazu.
```

Logické operátory vrací vždy hodnotu "pravda" (TRUE, 1), nebo "nepravda" (FALSE, 0). Za pravdivou se ale považuje jakákoliv nenulová hodnota. Nedefinovaná návěští dostávají tedy hodnotu 0 (nepravda, FALSE).

Některé z uvedených operátorů potřebují možná bližší vysvětlení ohledně účelu a použití. Proto jsou jednotlivé skupiny operátorů blíže vysvětleny v následujících pododstavcích.

3.5.1 Operátory: + - / \

Tyto aritmetické operátory jsou všeobecně známé, ačkoliv "\" může pro Vás nový být. Mějte na paměti, že tyto operace pracují s šestnácti bitovými čísly se znaménkem a ignorují všechna přetečení. Proto je například $FF00 + 4096 rovno $0F00 a žádná chyba se negeneruje.

```
komentář:  "op1 \ op2" je přesně rovno
           "op1 - [ op2 * [ op1 / op2 ] ]"
           a rovná se zbytku po celočíselném dělení.
           Příklad: 11\4 je 3.
```

3.5.2 Operátory: & ! ^

Toto jsou binární, neboli "bitové" operátory. Pracují s 16-ti bitovými hodnotami. Provádějí operace AND, OR a EOR bit po bitu. Jsou to 16-tibitové ekvivalenty operací AND, ORA a EOR procesoru 6502.

Příklady:
```
$FF00 & $00FF = $0000
$03 ! $0A     = $000B
$003F ^ $0120 = $011F
```

3.5.3 Operátory: = > < <> >= <=

Toto jsou známé porovnávací operátory. Porovnávají 16-tibitové operandy a vracejí hodnotu "pravda" (TRUE, 1), nebo "nepravda" (FALSE, 0).

Příklady:
```
3 < 5     vrací 1
5 < 5     vrací 0
5 <= 5    vrací 1
```

Upozornění: Tyto operátory pracují vždycky s dvojicí operandů. Znaménka "<" a ">" mají zcela jiný význam, pokud jsou použita jako unární operátory.

3.5.4 Operátory: .OR .AND .NOT

Tyto operátory také provádějí logické operace a nesmí se splést s jejich bitovými protějšky. Pamatujme, že tyto operátory vrací vždy pouze hodnotu "pravda" nebo "nepravda".

Příklady:
```
3 .OR 0      vrací 1
3 .AND 2     vrací 1
6 .AND 0     vrací 0
.NOT 7       vrací 0
```

3.5.5 Operátor: - (unární)

Znaménko mínus lze použít jako unární operátor. Jeho účinek je stejný, jako by bylo použito v odčítání, v němž je menšenec nula.

Příklad: -2 je $FFFE (totéž, co 0 - 2)

3.5.6 Operátory: < > (unární)

Tyto unární operátory jsou nesmírně užitečné, když je potřeba oddělit horní a dolní byte adresy návěští, nebo hodnoty výrazu. Nejčastěji se používají v instrukcích s přímým operandem, jako např. "LDA #" apod.

Příklad:
```
FLEEP = $3456
  LDA #<FLEEP (totéž, co LDA #$56)
  LDA #>FLEEP (totéž, co LDA #$34)
```

pozn.př.: Tento operátor nefunguje u starších assemblerů. U všech, a tedy i u MAC/65 funguje toto vyjádření:

```
LDA #FLEEP%255
LDA #FLEEP/256
```

3.5.7 Operátor: .DEF

Tento unární operátor testuje, zda již bylo definováno návěští, uvedené jako operand, a vrací hodnotu "pravda", nebo "nepravda" podle skutečného stavu.

Upozornění: Definovat návěští ZA operátorem .DIF, který testuje jeho existenci, může být nebezpečné, zvláště je-li .DEF v direktivě .IF.

Příklad:

```
        .IF .DEF ZILK
        .BYTE "generuj nějaké byty"
        .ENDIF
ZILK = $3000
```

V tomto příkladu se v prvním průchodu kompilace NEBUDE generovat textový řetězec v direktivě .BYTE. Ve druhém průchodu se ale text generovat BUDE. Tím se s největší pravděpodobností objeví při pokračování překladu chyba "PHASE ERROR".

3.5.8 Operátor: .REF

Tento unární operátor testuje, zda již bylo návěští v operandu uvedeno v některé instrukci nebo direktivě. Vrací hodnotu "pravda", nebo "nepravda", což lze použít ve spojení s direktivou .IF.

Pochopitelně zde platí totéž varování jako u .DEF při testování před výskytem návěští, ale zde můžeme vytěžit smysluplný efekt.

Příklad:

```
        .IF .REF PRINTMSG
PRINTMSG
        ... (kód, implementující rutinu
          PRINTMSG)
        .ENDIF
```

V tomto příkladu se kód, implementující rutinu PRINTMSG přeloží JEN tehdy, když se někde v předcházejícím kódu objevilo její jméno! Toto je výhodný způsob budování knihovny procedur assembleru, kdy se překládají jen procedury, které jsou v daném programu potřeba. Samozřejmě to předpokládá, že bude knihovna začleněna na konec programu direktivou .INCLUDE, což však není velké omezení. Tato technika byla použita firmou OSS při psaní knihoven pro kompilátor C/65 jazyka C.

Upozornění: Všimněte si, že v popisu bylo předpokládáno použití .REF ve spojení s .IF. Jiné použití není teoreticky vyloučeno, ale pokus použít .REF s jiným operátorem může dávat bizarní výsledky. Proto nelze .REF efektivně použít jinak, než ve spojení s .IF, takže například:

```
.IF .REF ZAM .OR .REF BLOOP  je zakázáno!
                             ----------
```

Jediný operátor, se kterým je možné legálně kombinovat .REF je .NOT, jako ve výrazu .IF .NOT .REF LABEL.

Všimněme si, že ilegální řádka, uvedená výše, může být legálně simulována takto:

Příklad:

```
DOIT .= 0
  .IF .REF ZAM
DOIT .= 1
  .ENDIF
  .IF .REF BLOOP
DOIT .= 1
  .ENDIF
  .IF DOIT
  ...
```

3.5.9 Operátor: [ ]

MAC/65 používá hranaté závorky jako "pseudozávorky". Normální kulaté závorky se ve výrazech nesmí použít, protože mají zvláštní význam u některých adresovacích modů. Proto se místo nich používají hranaté závorky, které mohou být použity v jakýchkoliv výrazech ke změně pořadí provádění operací, nebo k jeho potvrzení.

Příklady:

```
LDA GEORGE+5*3      ;toto je platné, ale vynásobí
                    se nejprve 3*5 a hodnota 15 se
                    potom přičte k návěští GEORGE
                    ...   to jste pravděpodobně
                    nechtěli.
LDA (GEORGE+5)*3    ;chyba syntaxe!!!!
LDA [GEORGE+5]*3    ;OK, sčítání se provede
                    před násobením
LDA ([GEORGE+5]*3),Y ;vidíte nutnost použít
                    obou druhů závorek?
```

Pamatujte: Pro operátory ve výrazech MAC/65 platí pravidla priority. Hranaté závorky lze použít ke změně těchto pravidel.

3.6. Výrazy v assembleru

Výrazem rozumíme jakoukoliv platnou kombinaci operandů a operátorů, kterou překladač vyčíslí jako 16-ti bitové číslo bez znaménka s ignorováním přetečení. Následující ukázky jsou příklady platných výrazů:

```
10      .WORD    TABLEBASE+LINE*COLUMN
55      .IF .DEF INTEGER .AND [ VER=1 .OR VER >= 3 ]
200     .BYTE    >EXPLOT-1, >EXDRAW-1, >EXFILL-1
300     LDA      #< [ < ADDRESS ^ - 1 ] + 1
305     CMP      # -1
400     CPX      # 'A
440     INC      %1+1
```

3.7 Priorita operátorů

Tabulka ukazuje prioritu operátorů (od nejvyšší do nejnižší) při vyčíslování výrazů:

```
[ ]   (pseudozávorky)
>   (horní byte),   <  (dolní byte),   .DEF,   .REF,   -  (unární)
.NOT
*,  /,  \
+,  -
&,  !,  ^
=,  >,  <,  <=,  >=,  <>      (porovnávací operátory)
.AND
.OR
```

Operátory, umístěné na stejné řádce mají stejnou prioritu a vyčíslují se zleva doprava, pokud se mezi nimi neobjeví operátor s vyšší prioritou.

Obecně řečeno, priorita operátorů odpovídá běžným matematickým zvyklostem. Musíme ale dávat pozor na unární operátory "<" a ">".

Příklad:

```
TABLE = $45FE
        LDA # > TABLE + 3 ;do A se dostane $48
        LDA # > [TABLE+3] ;do A se dostane $46
```

3.8 Numerické konstanty

MAC/65 zná tři druhy konstant: dekadické, hexadecimální a znakové.

Dekadická konstanta je dekadické číslo v rozsahu 0 až 65535; pokus použít čísla mimo tento rozsah může a nemusí fungovat, ale každopádně jeho výsledky nelze předvídat.

```
Příklady:           1  234  65200  32767
(při použití:)      .BYTE 2,4,8,16,32,64
                    LDA #1
```

Hexadecimální konstanta se skládá ze znaku "$" a jedné až čtyř hexadecimálních číslic (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). Použití více číslic než 4 přináší neurčitelné výsledky.

```
Příklady:           $1, $EA  $FF00  $7FFF
(při použití:)      .WORD $100, $200, $400, $800
                     AND #$7F
```

Znaková konstanta je definována jako apostrof, následovaný jakýmkoliv tištitelným, nebo zobrazitelným znakem. Hodnota znakové konstanty se rovná ASCII (nebo ATASCII) kódu znaku za apostrofem.

```
Příklady:           'A   '*   '"   '#
(při použití:)      CMP #'#
                    CMP #'Z+1 ;totéž, co #$5B
                    CMP #'J+3 ;totéž, co #'M
```

3.9 Stringy (textové řetězce)

Stringy mohou být dvou typů. Stringové (textové) konstanty, a stringové proměnné pro makra (příklad %$5).

```
Příklad:    10    .BYTE  "Retezec znaku"

Příklad:    20    .SBYTE %$1
```

Všimněme si, že v MAC/65 existuje jen 6 míst, kde jsou stringy povoleny: jako parametr při volání makra nebo jako operand direktiv .BYTE, .CBYTE, .SBYTE, .TITLE a .PAGE.

KAPITOLA 4: DIREKTIVY

Jak již bylo uvedeno v odstavci 3.1, pole instrukce v řádce zdrojového programu může obsahovat direktivu neboli pseudoinstrukci (místo platné instrukce 6502). Tato kapitola popisuje zhruba v abecedním pořadí všechny platné direktivy MAC/65 kromě direktiv, spojených s makroinstrukcemi. Ty jsou popsány zvlášť v kapitole 5.

Direktivy můžeme rozdělit do tří typů: 1) ty, které produkují kód pro použití přeloženým programem (t.j. .BYTE, .WORD atd.); 2) ty, které přimějí překladač, aby vykonal nějakou akci, jako třeba změnil umístění strojového kódu v paměti nebo přiřadil návěští nějakou hodnotu (t.j. např. *=, =, atd.); a 3) ty, které dávají programátorovi možnost ovlivňovat formát výpisu programu při překladu, umístění zdrojového textu a podobně (např. .TITLE, .OPT, .INCLUDE).

Samozřejmě, že bychom teoreticky mohli pracovat i bez direktiv třetího typu; ale až budete číst jejich popis, uvidíte, že jsou v praxi jedny z nejužitečnějších. Mimochodem, všechny direktivy spojené s makroinstrukcemi mohou být zařazeny do třetí skupiny.

Tři z direktiv, které budou následovat, (.PAGE, .TITLE a .ERROR) umožňují uvést text (v uvozovkách), který se vytiskne. U těchto třech direktiv je uživatel omezen maximální délkou textu 70 znaků. Přebytečné znaky se ignorují.

odstavec 4.1

direktiva: *= a .ORG

účel: změnit hodnotu čítače adresy assembleru

použití:
```
[ návěští ] *= výraz
[ návěští ] .ORG výraz
```

= (nebo ekvivalentně .ORG) přiřadí čítači adresy hodnotu výrazu. Výraz musí být předem definován. (= se musí psát dohromady bez mezery.)

Příklad:
```
50      *= $1234    ;nastaví čítač adresy na $1234
135     .ORG $1234  ; totéž
```

Jiným obvyklým použitím *= je vyhrazení místa pro data, která se naplní a používají v době běhu programu. Protože samotný znak "*" se chápe jako návěští (resp. proměnná) s hodnotou čítače adresy, výraz "*= *+exp" je nejobecnější metodou vyhrazení "exp" bytů pro pozdější použití.

Příklad:
```
70 LOC *= *+1  ; přiřadí návěští LOC momentální
                 hodnotu čítače adresy, načež zvýší
                 hodnotu čítače o 1.
70 LOC .ORG *+1  ; dtto
```

Upozornění: Protože návěští, spojené s touto direktivou, nabývá hodnoty čítače PŘED provedením direktivy, nemá smysl dávat k "*=", nebo ".ORG" návěští, pokud není použito k vyhrazení paměti, jako v druhém příkladu.

Poznámka: Některé assemblery chápou návěští direktivy "ORG", nebo ".ORG" odlišně, to jest, že přiřazují danému návěští hodnotu čítače adresy až po její změně direktivou. Proto je třeba při převodu z jiných assemblerů do MACu dávat na tuto věc pozor. Pokud máte pochybnosti, přesuňte návěští na řádku předchozí, nebo následující.

pozn.př.: Assemblery, které jsou u nás dostupné, t.j. Assembler/Editor a AMAC firmy ATARI a EASMD firmy OSS chápou direktivu "ORG", resp ".ORG" nebo "*=" stejně, jako MAC/65. U různých jiných assemblerů bude třeba dát pozor.

Zvláštní poznámka: I když je forma "návěští *= *+výraz" u 6502 standardní, můžete použít i direktivu MAC/65 ".DS" (viz odstavec 4.7), která je jednodušší na pochopení a v programu čitelnější.

odstavec 4.2
---

direktiva: = a .EQU

účel: přiřadit návěští hodnotu

použití: návěští = výraz
návěští .EQU výraz

Direktiva "=" přiřadí návěští hodnotu výrazu. Návěští může nabýt v programu hodnotu pomocí "=" jen jednou.

Příklad:
```
10 PLAYER0  =  PMBASE + $200
20 PLAYER1 .EQU PMBASE + $280
```

Poznámka: Je-li návěští přiřazována hodnota více, než jednou, bude "návěští" obsahovat naposledy přiřazenou hodnotu. Výsledkem tohoto procesu však bude chyba překladu.

odstavec 4.3
---

direktiva: .=

účel: nastavit návěští na změnitelnou hodnotu

použití: návěští .= výraz

Direktiva ".=" nastaví návěští na hodnotu výrazu. Pomocí této direktivy lze návěští nastavit v tomtéž programu na jednu nebo více hodnot tolikrát, kolikrát je třeba.

Příklad:
```
10 LBL .= 5
20     LDA #LBL      ;totéž, jako LDA #5
30 LBL .= 3+'A
40     LDA #LBL      ;totéž, jako LDA #68
```

Výstraha: Návěští, kterému byla přiřazena hodnota pomocí "=" nebo uvedením před instrukcí, nesmí být nastaveno na jinou hodnotu pomocí ".=".

odstavec 4.4

direktiva: .BYTE [a .SBYTE]

účel: určit obsah jednotlivých bytů ve výstupním kódu.

použití:
[návěští] .BYTE [+exp,] (exp) (strvar) [,(exp) (strvar) ...]
[návěští] .SBYTE [+exp,] (exp) (strvar) [,(exp) (strvar) ...]

Direktivy .BYTE a .SBYTE umožňují uživateli generovat ve výstupním strojovém kódu jednotlivé byty. Výsledkem výrazů musí být osmibitové čísla. Strvar generuje tolik bytů, kolik je délka řetězce. .BYTE generuje byty tak jak jsou napsány, .SBYTE je konvertuje do obrazovkového (interního) kódu počítačů ATARI.

Příklad: 100 .BYTE "ABC" , 3 , -1

V tomto příkladu se vytvoří ve výstupním kódu tyto byty:
41 42 43 03 FF
Všimněme si, že záporný výraz byl zkrácen na jednobytovou hodnotu.

Příklad: 50 .SBYTE "Hello!"

Tato řádka vygeneruje následující obrazovkové kódy:
28 65 6C 6C 6F 01.

Zvláštní poznámka: Obě direktivy, .BYTE i .SBYTE dovolují použít aditivního modifikátoru. Modifikátor je výraz, jehož hodnota se přičte ke každému generovanému bytu. Překladač rozpoznává modifikátor podle znaménka "+". Modifikátor sám není součástí výstupního kódu.

Příklad: 5 .BYTE +$80 , "ABC" , -1

Tato řádka vygeneruje následující kód:
C1 C2 C3 7F.

Příklad: 100 .BYTE +$80 , "DEF" ,'G+$80

Tato řádka vygeneruje následující kód:
C4 C5 C6 47.

(Všimněte si zvláště efektu přičtení $80 působením modifikátoru a dalším přičtením $80 k jednotlivému bytu. Výsledkem je nezměněný byte, protože jsme dohromady přičetli 256 ($100), což nezmění dolní byte 16-ti bitového výsledku.)

Příklad: 55 .SBYTE +$40 , "A12"

Tato řádka vygeneruje: 61 51 52

Příklad: 80 .SBYTE +$C0 , 'G-$C0,"REEN"

Tato řádka vygeneruje: 27 F2 E5 E5 EE

Poznámka: .SBYTE provádí konverzi všech výrazů, hodnot a stringů v

operandu podle algoritmu převodu ATASCII na kód displeje. Nevyjímá z něj ani zvláštní CTRL-znaky, jako je BELL, TAB a podobně. Tyto znaky JSOU konvertovány také.

odstavec 4.5
---

direktiva: .CBYTE

účel: totéž, co .BYTE s tím, že nejvyšší bit posledního byte textového argumentu je invertován.

použití:
[návěští] .CBYTE [+exp,] (exp) (strvar) [,(exp) (strvar) ...]

Direktiva .CBYTE může být často výhodně použita při výstavbě tabulek stringů apod., kde je nutné označit konec stringu jiným způsobem než např. nulovým bytem. Díky inverzi znaménka posledního bytu může rutina, čtoucí string, rozpoznat konec jednoduše instrukcí BPL, nebo BMI.

Příklad: ERRORS .CBYTE 1,"SYSTEM"

Tato řádka vygeneruje následující strojový kód:
01 53 59 53 54 45 CE

Podprogram může tyto byty zpracovávat následovně:

```
     LDY #1
LOOP LDA ERRORS,Y
     BMI ENDOFSTRING
     INY
     BNE LOOP
     ...
ENDOFSTRING
     ...
```

odstavec 4.6
---

direktiva: .DBYTE [viz též .WORD]

účel: specifikuje dvoubytovou hodnotu ve výstupním kódu s uložením v pořadí horní - dolní byte

použití: [návěští] .DBYTE exp [,exp ...]

Obě direktivy, .DBYTE i .WORD umisťují do výsledného strojového kódu hodnotu každého výrazu ve dvou bytech. Rozdíl je v tom, že .WORD umisťuje oba byty v adresovém uspořádání 6502, t.j. nejprve dolní, potom horní byte, .DBYTE umístí byty obráceně, t.j. horní - dolní.

.DBYTE má ve spojení s 6502 omezené použití a mohlo by být využíváno při tvorbě tabulek apod., kde by se mohlo využít obrácené pořadí bytů (oproti běžnému uspořádání).

Příklad: .DBYTE $1234,1,-1
produkuje: 12 34 00 01 FF FF
.WORD $1234,1,-1
produkuje: 34 12 01 00 FF FF

odstavec 4.7
---
direktiva: .DS

účel: vyhrazuje místo pro data, aniž by mu byla přiřazována nějaká hodnota.

použití: [návěští] .DS výraz

Použití ".DS výraz" je ekvivalentní konstrukci " *= *+výraz ". To znamená, že návěští, pokud je uvedeno, dostane momentální hodnotu čítače adresy. Potom se hodnoda čítače zvýší o hodnotu výrazu.

Příklad:
```
BUFFERLEN .DS 1   ; vyhrazení 1 byte
BUFFER    .DS 256 ; vyhrazení 256 bytů
```

odstavec 4.8
---
direktiva: .ELSE

účel: účel a použití je popsán u .IF.

odstavec 4.9
---
direktiva: .END

účel: ukončení překladu z paměti

použití: [návěští] .END

Direktiva .END ukončí překlad JEN v případě, že se zdroj čte z paměti. Jinak nemá .END žádný význam a překlad neovlivní.

Toto "neovlivnění" je šikovné, protože můžete včleňovat soubory pomocí .INCLUDE, aniž by bylo nutno z nich odstraňovat .ENDy, které by mohly eventuelně obsahovat. Ve skutečnosti není u MAC/65 direktiva .END vůbec potřeba.

odstavec 4.10
---
direktiva: .ENDIF

účel: ukončuje blok podmíněného překladu

Viz popis .IF, kde je uvedeno použití a další podrobnosti

odstavec 4.11
---
direktiva: .ERROR

účel: vynucení chyby překladu s výpisem zprávy

použití: [návěští] .ERROR [string]

Direktiva .ERROR umožňuje uživateli vygenerovat pseudochybu. String, specifikovaný direktivou se vypíše, jakoby šlo o chybu, generovanou překladačem. Chyba se započítá do počtu chyb, uváděného na konci překladu.

Příklad: 100 .ERROR "Chybí parametr!"

odstavec 4.12

direktiva: .FLOAT

účel: určení konstanty v pohyblivé řádové čárce k umístění ve výstupním kódu.

použití:
[návěští] .FLOAT floating-constant [floating-constant ...]

,kde floating-constant je konstanta v pohyblivé řádové čárce.

Tuto direktivu použije jen programátor, který chce používat vestavěnou matematickou ROM.

Každá konstanta v pohyblivé čárce, následující za direktivou .FLOAT, způsobí vygenerování šesti bytů do výstupního kódu. Tyto byty jsou ve formátu, používaném výše zmíněnou matematickou ROM. Všeobecně vzato, první byte obsahuje exponenciální část čísla v notaci excess-64, reprezentující mocniny 100. (pozn.př.: podrobnosti lze najít v "OPERATING SYSTEM USER'S MANUAL, ATARI, INC. 1982.) Horní bit exponentového bytu určuje znaménko mantisy. Následujících 5 byte je mantisa v pakovaném BCD tvaru, normalizovaná na hranici byte (což odpovídá mocninám exponentu 100).

Příklady:
.FLOAT 3.14146295, -2.718281828

Výše uvedený příklad vyprodukuje ve výstupním kódu následující byty:
40 02 14 15 62 95
C0 27 18 28 18 28

Poznámka: U direktivy .FLOAT jsou povoleny jen konstanty, NIKOLIV výrazy. Není ovšem problém, aby si programátor spočítal hodnotu výrazu na kalkulačce, nebo v BASICu a potom napsal výsledek do programu MAC/65.

odstavec 4.13

direktiva: .IF

účel: určení, zda se určitá část programu bude nebo nebude překládat podle výsledku vyčíslení výrazu "exp".

použití:
.IF exp
[.ELSE]
.ENDIF

poznámka k použití: mezi .IF a .ELSE, nebo .ENDIF může být libovolný počet řádek zdrojového programu, stejně jako mezi .ELSE a .ENDIF.

Direktivy .IF, .ELSE a .ENDIF řídí podmíněný překlad.

Jestliže překladač narazí na .IF, vyčíslí výraz, následující za ním.

Je-li výsledek nenulový (pravda), překládají se zdrojové řádky, dokud se nenarazí na .ENDIF, nebo .ELSE. Jestliže se narazí na .ENDIF dříve, než na .ELSE, řádky mezi .ELSE a odpovídajícím .ENDIF se nepřekládají. Je-li výsledkem operace nula (nepravda), řádky za .IF se nepřekládají. Překlad opět začne po odpovídajícím .ELSE nebo .ENDIF.

Konstrukce .IF-.ENDIF a .IF-.ELSE-.ENDIF mohou být do sebe vnořovány až do hloubky 14-ti úrovní. Při hledání odpovídajícího .ELSE a .ENDIF se přeskakují vnořené konstrukce nižší úrovně.

Příklady:

```
10      .IF 1                   ;nenulové, proto pravda
20      LDA # '?                ;tyto dvě řádky
30      JSR CHAROUT             ;budou přeloženy
40      .ENDIF

10      .IF  0                  ;výraz je nepravda
20      LDX  # >ADDRESS         ;tyto dvě řádky
30      LDY  # <ADDRESS         ;se nebudou překládat
40      .IF  1
50      .ERROR "sem se nelze dostat"
60 ;zde může být cokoliv, protože tato konstrukce
70 ;je vnořena do úseku, který se nepřekládá
80      .ENDIF
90      .ELSE
100 ;
110     LDX  # <ADDRESS         ;tyto řádky
120     LDY  # >ADDRESS         ;budou přeloženy
130     .ENDIF
140     JSR  PRINTSTRING        ;pokračujeme v programu
```

Poznámka: Překladač inicializuje podmínkový zásobník na začátku každého průchodu. Chybějící .ENDIFy se NEindikují.

odstavec 4.14
---

direktiva: .INCLUDE

účel: umožňuje jednomu zdrojovému programu, aby požadoval včlenění jiného programu a společný překlad obou.

použití: .INCLUDE #filespec

poznámka k použití: tato direktiva nesmí mít návěští.

Direktiva .INCLUDE přiměje překladač ke čtení zdrojového programu ze specifikovaného zařízení. Je-li dosažen konec souboru, pokračuje překladač ve čtení původního souboru (nebo paměti).

Výstraha: Soubor, včleňovaný pomocí .INCLUDE musí být tokenizovaný program MAC/65, správně uložený povelem SAVE v editoru. Nemůže to být textový ASCII soubor.

Poznámka: Včleňovaný soubor nesmí sám obsahovat direktivu .INCLUDE.

Příklad: .INCLUDE #D:SYSEQU.M65

Tento příklad včlení do programu soubor systémových přiřazení firmy OSS.

odstavec 4.15

direktiva: .LOCAL

účel: ukončuje oblast lokálních návěští

použití: .LOCAL

poznámka k použití: tato direktiva nesmí mít návěští.

Tato direktiva ukončí přecházející oblast platnosti lokálních návěští a začne oblast novou. Předpokládá se, že první lokální oblast začíná na začátku programu a poslední oblast končí na konci programu.

Jakékoliv návěští, začínající dvojtečkou ":", nebo otazníkem "?", se považuje v dané lokální oblasti za lokální. Jako takové je neviditelné a nepřístupné kódu, přiřazením, odkazům atd. mimo vlastní lokální oblast.

Tento rys je zvláště výhodný při používání automatického generování kódu nebo když pracuje na jednom projektu více lidí. V obou případech může programátor používat návěští, začínající ":" nebo "?" s jistotou, že nemůže vzniknout duplicita.

Příklad:

```
10  *= $4000
11  LDX #3      ;nastavení čítače
12 ?LOOP
13  LDA FROM,X ;načtení byte
14  STA TO,X   ;zápis byte
15  DEX        ;je ještě co dělat?
16  BPL ?LOOP  ;jde na řádku 12
17 ;
18  .LOCAL     ;jiná lokální oblast
19 ;
20 ?LOOP = 6
21 ;
22  LDY #?LOOP ;totéž, co LDY #6
23   (atd.)
```

odstavec 4.16

direktiva: .OPT

účel: vybírá různé volby řízení překladu

použití: .OPT volba [, [NO] volba ...]
nebo
.OPT [NO] volba [, [NO] volba ...]

poznámka k použití: platné volby jsou následující.

LIST	ERR	EJECT	OBJ
MLIST	CLIST	NUM	XREF

Direktiva .OPT umožňuje uživateli řídit určité funkce překladu. Uvedení ".OPT volba" zapíná funkci, nebo volbu, zatímco ".OPT NO

volba" ji vypíná.

Na jedné řádce lze použít libovolný počet voleb. Lze například napsat toto:

```
.OPT NO LIST, NO XREF, OBJ, ERR
```

Následuje popis jednotlivých možností:

LIST řídí výpis programu při překladu
NO LIST vypíná výpis všeho kromě chybových řádek.

ERR určí, zda se budou vypisovat chyby.
NO ERR potlačuje výpis chyb a je tím pádem nebezpečné.

EJECT řídí stránkování a výpisy titulů a nadpisů stránek
NO EJECT vypíná automatické stránkování, což však nemá vliv na direktivu .PAGE

OBJ určí, zda se bude do paměti nebo na zařízení zapisovat výstupní kód.
NO OBJ je užitečné při zkušebních překladech.
OBJ je nutno zadat, má-li se překládat přímo do paměti.

NUM automaticky čísluje výpis zdroje při překladu, místo původních čísel řádek. NUM začíná od 100 s přírůstkem 1.
NUM nemá obvykle mnoho použití, snad kromě konečného "pěkného" výpisu programu.

MLIST řídí výpis maker.
NO MLIST vypisuje jen ty řádky makroinstrukcí, které produkují výstupní kód.
MLIST vypisuje makra celá.

Poznamenejme, že NO MLIST je zvláště užitečné pro vznik čitelných listingů.

CLIST řídí výpis úseků podmíněného překladu.
NO CLIST nevypisuje řádky, které se nepřekládají.
CLIST vypisuje všechny řádky podmíněných konstrukcí.

XREF dovoluje uživateli určit, které části programu budou zahrnuty do křížových referencí, pokud je při povelu ASM definováno zařízení #file4.

Řádky mezi .OPT NO XREF a .OPT XREF se v křížových referencích neuplatní.

Kombinací NO XREF a NO LIST můžete vypsat a křížově analyzovat i velmi rozsáhlé programy po částech. Nebo můžete použít NO XREF k potlačení křížových referencí ve včleňovaných souborech (.INCLUDE). XREF a NO XREF jsou neúčinné, pokud není v povelu ASM specifikován #file4 (ale chyba se negeneruje).

Poznámka: Pokud uživatel neurčí jinak, platí pro všechny volby standardní nastavení. Standardy jsou následující:

LIST	listing se generuje
ERR	chyby ve vypisují
EJECT	stránkuje se a stránky se číslují
NO NUM	používají se čísla řádek ze zdroje
MLIST	vypisují se všechny řádky maker
CLIST	vypisují se i nekompilované řádky
XREF	křížové reference z celého programu
NO OBJ	VIZ UPOZORNENÍ!!!

Upozornění: Volba OBJ je řízena dvojím způsobem:
Při kompilaci do paměti je standard .OPT NO OBJ.
Při kompilaci do souboru je standard .OPT OBJ.

Poznámka: Je-li uvedeno .OPT NO NUM, vypisují se řádky makroinstrukcí bez čísel řádek.

odstavec 4.17

direktiva: .PAGE

účel: generuje hlavičku stránky a/nebo přechází na novou stránku

použití: .PAGE [string]

poznámka k použití: direktiva .PAGE nesmí mít návěští.

Direktiva .PAGE umožňuje uživateli specifikovat záhlaví stránky, které se vytiskne pod číslo stránky a titulní hlavičku.

.PAGE odstránkuje a vytiskne poslední platnou titulní hlavičku a hlavičku stránky.

Příklad: 300 .PAGE "Hledani navesti"

Poznámka: Assembler automaticky odstránkuje a vytiskne hlavičky po vypsání 61 řádek (viz ale direktivu .SET, odst. 4.19).

odstavec 4.18

direktiva: .SBYTE

účel: generuje "obrazovkové" byty do výstupního kódu

použití: viz popis .BYTE, odstavec 4.4

odstavec 4.19

direktiva: .SET

účel: řídí různé funkce překladače

použití: .SET dcnum1, dcnum2

Direktiva .SET umožňuje uživateli měnit různé vnitřní parametry překladače assembleru. Dcnum1 určuje parametr, který se bude měnit, dcnum2 je nová hodnota. Následující tabulka soustřeďuje všechny

parametry .SET. Standardní hodnoty parametrů jsou v závorkách, za nimi je uveden povolený rozsah hodnot.

dcnum1	dcnum2	funkce
0	(4) 1-4	řídí formát výpisu .BYTE a .SBYTE Na řádce se může vytisknout v poli strojového kódu 1 až 4 byty.
1	(0) 0-31	nastavuje levý okraj výpisu zdroje překladu. Číslo udává počet mezer, které se vytisknou před opisem překládané zdrojové řádky
2	(80) 40-132	nastavuje šířku výpisu, přizpůsobení tiskárně
3	(12) 0,12	výběr druhu odstránkování. 0 znamená, že tiskárna nemá funkci Form Feed a stránkování se provádí opakovaným řádkováním. Cokoliv jiného se použije jako kód pro Form Feed.
4	(66) 20-255	počet řádek na stránku
5	(0) 0-255	počet mezer mezi středníkem v poli komentáře a zbytkem textu komentáře.
6	(0) 0-$FFFF	konstanta, která se přičítá k čítači adresy v okamžiku zápisu bytu do souboru. Tak se dá vytvořit kód pro určitou adresu, zaváděný od jiné adresy.

Zvláštní poznámka: Podrobný popis možností .SET 6 je v kapitole 8.

pozn.př.: U starší disketové verze MAC/65 nefunguje direktiva .SET 6. Pro správný výpis programů na evropský papír je třeba nastavit .SET 4,72 a pro Atari 1029 ještě .SET 3,0.

odstavec 4.20
---

direktiva: .TAB

účel: nastavuje tabelátorové zarážky pro zlepšení čitelnosti výpisu

použití: .TAB dcnum1, dcnum2, dcnum3

Direktiva .TAB umožňuje určit počáteční sloupec výpisu instrukčního pole, pole operandů a pole komentáře. Standardní hodnoty jsou 8, 12, 20.

```
Příklad:    200     .TAB 16,32,50
            ...
            1200    .TAB 8,12,20 ; návrat ke standardu
```

odstavec 4.21

direktiva: .TITLE

účel: určuje hlavičku výpisu překladu

použití: .TITLE string

Direktiva .TITLE umožňuje uživateli stanovit záhlaví výpisu programu při překladu. Titulní text se vypisuje v záhlaví každé stránky, následovaný číslem stránky.

odstavec 4.22

direktiva: .WORD [viz též .DBYTE]

účel: umístí ve výstupním kódu 16-ti bitové slovo

použití: [návěští] .WORD exp [,exp ...]

Obě direktivy, .WORD i .DBYTE umisťují do výsledného strojového kódu hodnotu každého výrazu ve dvou bytech. Rozdíl je v tom, že .WORD umisťuje oba byty v adresovém uspořádání 6502, t.j. nejprve dolní, potom horní byte, .DBYTE umístí byty obráceně, t.j. horní - dolní.

.WORD je v programech pro 6502 mnohem užitečnější a je slučitelné s kódem, který vniká překladem instrukcí 6502.

```
Příklad:    .DBYTE $1234,1,-1
                produkuje: 12 34 00 01 FF FF
            .WORD  $1234,1,-1
                produkuje: 34 12 01 00 FF FF
```

KAPITOLA 5: MAKROINSTRUKCE

Definice makra (makroinstrukce) je seskupení zdrojových řádek, kterému je přiřazeno jméno a je uloženo v paměti. Když pak překladač najde v poli instrukce místo direktivy, nebo mnemoniky instrukce jméno makra, nahradí jméno makra řádkami, uloženými pod tímto jménem v paměti, a přeloží je. Prakticky to znamená, že si uživatel může definovat nové instrukce pro assembler. Podle toho, jaký kód je definován, může makro tvořit buď zvláštní direktivu nebo novou instrukci.

Proces vyhledání makra v tabulce a překlad jím definovaného kódu, který definuje, se nazývá volání (nebo použití) makra. Jedinečnou vlastností makroinstrukcí je možnost předávat jim parametry. Tyto skutečné parametry nahradí během volání makra formální parametry, které jsou použity v definici makra.

Použití (volání) makra vyžaduje, aby bylo makro nejprve definováno. Proto musíme k souboru direktiv, popsaných v kapitole 4, přidat ještě dvě direktivy, definující nové makro:

```
.MACRO
.ENDM
```

Tato kapitola nejprve vysvětlí obě uvedené direktivy, ukáže, jak použít (vyvolat) makro a vysvětlí použití formálních a skutečných parametrů včetně parametrů stringových.

odstavec 5.1

direktiva: .ENDM

účel: ukončit definici makra.

použití: .ENDM

poznámka k použití: tato direktiva nesmí mít návěští.

Tato direktiva se používá k ukončení definice makra. Nepoužívá se při volání makra. Koncept maker vyžaduje, aby všechny řádky mezi direktivami .MACRO a .ENDM byly uloženy ve zvláštní části paměti (tabulce maker). Proto je nalezení nesprávně párované direktivy .ENDM považováno za závažnou chybu. Další informace jsou u popisu .MACRO.

odstavec 5.2

direktiva: .MACRO

účel: definovat makro

použití: .MACRO jméno makra

poznámka k použití: "jméno makra" může být jakékoliv jméno podle konvencí MAC/65 pro návěští. Může to být i jméno stejné, jako jméno některého návěští v programu, aniž by vznikl konflikt.

Direktiva .MACRO způsobí, že řádky, které ho následují, se načtou a pod jménem makra uloží. Definice končí direktivou .ENDM.

Ve zdrojových řádkách makroinstrukce může být cokoliv kromě direktivy .MACRO. Makrodefinice NESMÍ obsahovat definici jiného makra.

Jednoduchý příklad makrodefinice:

```
10   .MACRO PUSHXY ; Jméno makra je "PUSHXY"
11 ; Je-li makro použito (vyvoláno), následující instrukce
12 ; nahradí "PUSHXY" a přeloží se.
13  TXA
14  PHA
15  TYA
16  PHA
17  .ENDM           ; ukončení "PUSHXY"
```

Zvláštní poznámka: Všechna návěští uvnitř makra se pokládají v daném makru za lokální. MAC/65 to realizuje "třetím průchodem" překladu během volání makra. Tím je návěští v makru přístupné kódu, který za makrem následuje; jiné volání makra však hodnotu návěští změní. Je to podobné, jako u direktivy ".=" s tím, že dopředné odkazy na návěští v makru jsou POVOLENY.

pozn.př.: Přestože se lze odkazovat na návěští v makroinstrukci, z důvodu čitelnosti a srozumitelnosti programu bych doporučoval se této metodě vyhnout.

Příklad:

```
20  .MACRO MOVE6
21  LDX #5
22 LOOP
23  LDA FROM,X
24  STA TO,X
25  DEX
26  BPL LOOP
27  .ENDM
```

Návěští "LOOP" je pro použití makra lokální. Kromě toho se na něj můžeme odkazovat i mimo makro (ale ne v jiném makru). (Všimněme si, že pokud bylo návěští v programu definováno vyvoláním makra jen jednou, má to stejný efekt, jako by bylo návěští definováno kdekoliv mimo makro.) Ačkoliv lze direktivu .LOCAL použít i v oblasti makroinstrukcí, je uživatel omezen na maximálně 62 lokálních oblastí. Pro počet lokálních použití návěští v makrech neexistuje žádné omezení.

5.3 VOLÁNÍ MAKRA, část 1

Jak již bylo vysvětleno, makro při svém volání "expanduje", čili jeho jméno je nahrazeno kódem, který je jím reprezentován. Použití makra je jednoduchost sama.

Vyvolat (použít) makro znamená napsat jeho jméno do sloupce operace ve zdrojové řádce. Pamatujme ale, že makro musí být před použitím definováno.

Jako příklad použijeme makra, definovaná v předchozím odstavci 5.2. Pro praktické vyzkoušení je možné všechny tři příklady "naklofat" a přeložit:

Příklad:

```
2000 ALABEL PUSHXY
2010 ; makro PUSHXY generuje následující kód:
2020 ;  TXA   PHA   TYA   PHA
2030 ;
2040  MOVE6
2050 ;  podobně použité MOVE6
2060  JMP LOOP
2070 ; LOOP je návěští, definované
2080 ; uvnitř makra MOVE6
...
```

Poznamenejme, že použití návěští u volání makra je nepovinné. Návěští, pokud je uvedeno, dostane hodnotu čítače adresy bez ohledu na obsah volaného makra.

S použitím maker je spojeno mnoho "triků" a možností, ale budeme se jimi zabývat po vysvětlení používání parametrů v definicích makra.

5.4 PARAMETRY MAKER

Parametry maker mohou být dvou druhů: výrazy (které se vyčíslí jako 16-ti bitová hodnota), nebo stringy. Parametry jsou předávány při volání (použití, expanzi) makra a jsou v paměti uskladněny v pořadí výskytu. Celkově může být uskladněno 63 parametrů, včetně parametrů pro volání maker uvnitř jiných maker.

Samozřejmě, že před napsáním parametru ve volání makra musí být cesta k jeho zpřístupnění v DEFINICI MAKRA. Parametry jsou v definici makra označovány znakem "%" pro výrazy a "%$" pro stringy. Hodnota za tímto znakem určuje pořadové číslo skutečného parametru.

Zvláštní poznámka: Číslo parametru může být reprezentováno jak dekadickým číslem (např. %2), tak návěštím, uzavřeným v závorkách (např. %(LABEL)). Totéž platí pro stringové parametry (%$1, nebo %$(INDEX)).

Příklady:

```
10      LDA     # >%1    ; horní byte parametru 1
15      CMP     (%23,X) ; parametr číslo 23
20      .BYTE   %2-1    ; hodnota parametru 2, zmenšená o 1
```

Poznámka: Příklad v řádce 20 NENÍ ekvivalentní použití parametru %1. Nahrazení parametru má vyšší prioritu!

```
25 SYMBOL .= SYMBOL + 1
30      LDX     # -%(SYMBOL)  ;vidíte ty možnosti?
40      .BYTE   %$1,%$2,0  ; stringové parametry, ukončené nulou
```

Pamatujte, že parametry mohou být teoreticky číslovány od 1 do 63. Ve skutečnosti nesmí CELKOVÝ počet parametrů všech současně aktivních (do sebe vnořených) maker překročit 63. To neznamená, že byste ve svých makrodefinicích neměli použít více, než 63 parametrů. Omezení se vztahuje jen na okamžik volání makra, nebo soustavy vnořených maker. Vysvětlím to na příkladu. Mějme soustavu různých maker. Každé z nich má 10 parametrů. Předpokládejme, že makro 1 volá makro 2, to zas volá makro 3 atd. Takováto soustava může mít nejvýše 6 stupňů. Pokud by

makro 6 volalo ještě makro 7, byl by již celkový počet parametrů v soustavě 70, což není přípustné.

Zvláštní poznámka: Kromě "normálních" parametrů, označených pořadovým číslem, má u MAC/65 zvláštní význam parametr nula (%0). Tento parametr oznámí uživateli, kolik parametrů bylo ve volání makra uvedeno.

Tento parametr umožňuje nastavit uvnitř makra parametry na standardní hodnoty nebo testovat správný počet parametrů a podobně. Následující příklad ilustruje použití %0 a ukazuje správné používání parametrů.

Příklad:

```
10  .MACRO BUMP
11 ;
12 ; Toto makro zvýší hodnotu specifikovaného slova
13 ;
14 ; formát vyvolání je následující:
15 ;              BUMP adresa [ ,inkrement ]
16 ; není-li inkrement uveden, předpokládá se 1
17 ;
18  .IF %0=0 OR %0>2
19    .ERROR "BUMP: chybny pocet parametru"
20  .ELSE
21 ;
22 ; toto se provádí jen pro 1 nebo 2 parametry
23 ;
24    .IF %0>1      ; uvedl uživatel inkrement?
25 ;toto se překládá, když uvedl uživatel 2 parametry
26     LDA %1       ;přičti inkrement k obsahu adresy
27     CLC
28     ADC #<%2     ;dolní byte přírůstku
29     STA %1       ;dolní byte výsledku
30     LDA %1+1     ;horní byte slova
31     ADC #>%2     ;přičti horní byte inkrementu
32     STA %1+1     ;ulož do zbytku výsledku
33 ;
34    .ELSE
35 ;toto se překládá, je-li zadán jen jeden parametr
36     INC %1       ;zvyš o 1
37     BNE SKIPHI   ;implicitně lokální návěští
38     INC %1+1     ;musíme zvýšit i horní byte
39 SKIPHI
40    .ENDIF        ;patří k .IF na řádce 24 (.IF %0>1)
41  .ENDIF          ;patří k .IF na řádce 18
42  .ENDM           ;terminátor
```

5.5 VOLÁNÍ MAKRA, část 2

Již jsme si ukázali, že makrodefinice mohou obsahovat specifikace různých parametrů (tyto specifikace se nazývají formální parametry). Tento odstavec nám ukáže, jak definici předat skutečné parametry (hodnoty parametrů, nebo také parametry volání).

Metoda je jednoduchá. Na řádce s voláním makra může uživatel za jménem makra uvést výrazy (nebo stringy, viz 5.6). MAC/65 přiřadí jejich hodnotám čísla od jedné do 63 a během expanze makra nahradí formální parametry (%1, %2, %(label), atd.) odpovídajícími hodnotami.

Zní to příliš komplikovaně? Interně to komplikované je. Externě je to však takhle jednoduché:

Příklad:

Předpokládejme, že je definováno makro BUMP z předchozího příkladu (odst. 5.4). Uživatel jej může volat dle potřeby takto:

```
100 ALABEL BUMP A.LOCATION
110 INCR .= 7
120  BUMP A.LOCATION,3
130  BUMP A.LOCATION,INCR-2
140  BUMP
150  BUMP A.LOCATION,INCR,7
160 A.LOCATION .WORD 0
     pozn: řádky 140 a 150 vybudí chybu a výpis:
     "BUMP: chybný počet parametrů!"
```

Při volání maker nesmíme přehodit pořadí parametrů, protože makro nemůže obvykle záměnu odhalit. Další příklady ukazují, jak lze dosáhnout výsledků, které jsme nechtěli:

```
170  BUMP INCR,A.LOCATION
          pokus zvýšit adresu 7 o cosi
180  BUMP PORT5
          je-li PORT5 hardwarový registr,
          mohou se dít neuvěřitelné věci
```

5.6 Stringové parametry

Stringové parametry jsou v definici makra reprezentovány znakem "%$". Všechny numerické parametry mají své stringové protějšky, ale ne všechny z nich jsou k něčemu. Všechny stringové parametry mají své numerické protějšky (délku).

Jako zvláštní případ, %$0 vrací vždy jméno makra.

Následující tabulka ukazuje různé stringy a numerické hodnoty pro daný parametr:

Co je uvedeno ve volání makra:	vrácený string (v uvozovkách):	vrácená numerická hodnota:
"A String 1 2 3"	"A String 1 2 3"	délka stringu
NUMERICSYMBOL	"NUMERICSYMBOL"	hodnota návěští
SYMBOL+1	"SYMBOL"	hodnota výrazy
%$4	string parametru 4	hodnota originálu
(tento způsob se používá v makrech, volajících jiné makro)		
-LABEL	"LABEL"	hodnota výrazu
GEORGE*HARRY+PETE	nedefinováno	hodnota výrazu
.DEF CIO	"CIO"	hodnota výrazu
2 + 2 * 65	nedefinováno	hodnota výrazu

Příklad stringového parametru:

```
10  .MACRO PRINT
11 ;
12 ; Toto makro vytiskne zadaný stringový
13 ; parametr 1. Pokud není žádný zadán, vytiskne EOL.
15 ;
16 ; forma vyvolání:   PRINT  [ string ]
17 ;
18  .IF %0 = 1              ;je string na vytištění?
19   JMP PASTSTR            ;ano, přeskočíme místo umístění
20 STRING .BYTE %$1,EOL     ;dáme string sem
21 ;
22 PASTSTR
23   LDX #>STRING           ;adresu stringu do X a Y
24   LDY #<STRING           ;pro JSR na "tisk stringu"
25   JSR STRINGOUT
26  .ELSE
27 ; není string, tiskne EOL
28   LDA #EOL
29   JSR CHAROUT
30 ;
31  .ENDIF
32  .ENDM                   ;konec
```

Toto makro lze volat následujícími způsoby:

```
100     PRINT     "tohle je string"
110     PRINT
120     PRINT     ZPRAVA
```

Řádka 120 je podivná: Makroinstrukce předpokládá, že "ZPRAVA" je string (kvůli jeho použití), a vytiskne jej přesně tak, jako by byl umístěn v uvozovkách. Pokud ale nebylo návěští ZPRAVA v programu definováno, způsobí tato řádka navíc chybu "Undefined Label". Proto nelze používání tohoto způsobu doporučit. Používejte vždy texty v uvozovkách.

5.7 DOPORUČENÍ PRO MAKRA

Každý si určitě brzy vytvoří vlastní styl psaní makroinstrukcí, ale jsou tu obecná pravidla, která se vyplatí dodržovat.

A. Definované makro musí být celé uloženo v paměti (v tabulce maker). Protože velikost paměti je omezena, je dobré udržovat makra co nejmenší. Jedním ze způsobů, jak toho dosáhnout, je nepoužívat v makrech komentář. Pokud chcete svá makra dokumentovat (což bezpochyby chcete), umístěte komentář před direktivu .MACRO. Překladač pak nespotřebuje paměťové místo na komentáře.

B. Nepoužívejte parametr volání, pokud si nejste jisti, že byl při vyvolání makra uveden. Použití parametru, který ve volání uveden nebyl, způsobí chybu "MACRO PARAMETER error". Záleží na definici makra, zda může nebo nemůže produkovat nepředvídatelné výsledky. Příklad nezabezpečeného makra:

```
.IF %0>1 .OR %2=0
 .WORD %1
.ENDIF
```

Nebezpečí zde vznikne, když se při volání neuvede druhý parametr. Protože %2 neexistuje (a je tím pádem nedefinován), podvýraz "%2=0" je vždy pravda a efekt "%0>1" je potlačen. Absence parametru 2 sice způsobí "PARAMETER ERROR", to však již bývá pozdě. Lépe je výše uvedený příklad napsat takto:

```
.IF %0>1
 .IF %2<>0
  .WORD %1
 .ENDIF
.ENDIF
```

C. I když jsou návěští v makrech v každém volání lokální, zůstávají "viditelná" i mimo makra. Proto je dobré používat pro návěští v makrodefinicích zvláštní formu, která se nepoužívá nikde jinde mimo makra. Knihovna maker, dodávaná s MAC/65, používá jako lokální návěští maker návěští, začínající znakem "@".

Výstraha: V makru nesmíte definovat návěští, začínající otazníkem, nebo dvojtečkou. Stejně tak se uvnitř makra nesmí použít direktiva .LOCAL. (Lokální návěští, začínající otazníkem lze v makru použít, pokud jsou definována mimo něj.)

5.8 KOMPLEXNÍ PŘÍKLAD MAKER

Následující sada makroinstrukcí je navržena jako ukázka dodržování některých bodů, uvedených v předchozích kapitolách. Kromě toho je to sada dobrých a užitečných maker. Dobře si je prostudujte. (Čísla řádek byla pro lepší přehlednost vynechána.)

```
;
; První macro, "@CH" je navrženo pro vložení ukazatele IOCB do
; registru X. Je-li mu předána hodnota 0 až 7, předpokládá se, že
; jde o číslo kanálu (přímý operand). Je-li předána jiná hodnota,
; předpokládá se, že jde o adresu v paměti, která obsahuje
; číslo kanálu.
;
; Všimněte si, že veškeré komentáře jsou mimo tělo makra,
; čímž se šetří paměť.
;
        .MACRO @CH
          .IF %1>7          ;kde je číslo kanálu?
          LDA %1            ;číslo kanálu je v paměti
          ASL A             ;natáhneme ho a
          ASL A             ;vynásobíme
          ASL A             ;16-ti pomocí
          ASL A             ;těchto posunů
          TAX               ;a přesuneme do registru X
          .ELSE
          LDX #%1*16        ;číslo kanálu *16 jde do X
          .ENDIF
        .ENDM
```

```
; Další makro, "@CV", slouží k naplnění akumulátoru konstantou, nebo
; hodnotou. Je-li hodnota v rozmezí 0 až 255, pokládá se za hodnotu
; (přímý operand) konstanty. Jiná hodnota se pokládá za adresu.
; (mimo základní stránku).
;
        .MACRO @CV
          .IF %1<256          ;je to konstanta?
          LDA #%1             ;ano, natáhneme ji
          .ELSE
          LDA %1              ;ne, vezmeme si ji z paměti
          .ENDIF
        .ENDM
;
; Třetí makro je "@FL", které určí specifikaci souboru.
; Jestliže je předána textová konstanta, generuje @FL string
; do řádky, přeskočí ji a umístí její adresu v IOCB, na který ukazuje
; registr X. Pokud je předáno návěští mimo základní stránku, makro @FL
; předpokládá, že jde o adresu stringu s platnou specifikací souboru
; a použije ji místo adresy vlastní řádky
;
        .MACRO @FL
          .IF %1<256          ;je to stringová konstanta?
          JMP *+%1+4          ;ano, přeskoč string
@F        .BYTE %$1,0         ;... a ulož ho sem
          LDA # <@F           ;natáhni si adresu stringu
          STA ICBADR,X        ; a ulož ji do pole adresy bufferu IOCB
          LDA # >@F           ;totéž pro horní byte
          STA ICBADR+1,X
          .ELSE
          LDA # <%1           ;není konstanta
          STA ICBADR,X        ;takže jenom předáme adresu
          LDA # >%1           ; (oba byty)
          STA ICBADR+1,X      ;do IOCB
          .ENDIF
        .ENDM
```

```
;
; Hlavní makro v tomto příkladu je makro "XIO", které simuluje
; stejnojmenný povel jazyka BASIC.
; Syntaxe použití tohoto makra je následující:
;   XIO povel,kanál[,aux1,aux2][,filespec]
;
; kde kanál  je konstanta od 0 do 7, nebo adresa v paměti,
;
; povel, aux1, aux2 mohou být konstanty od 0 do 255, nebo adresy mimo
;      nulovou stránku,
;
; filespec může být buď textová konstanta, nebo adresa mimo
;      základní stránku
; Jsou-li vynechány aux1 a aux2, předpokládá se, že jsou nulové
; Je-li vynechána specifikace souboru, předpokládá se "S:".
;
        .MACRO XIO
          .IF %0<2 .OR %0>5  ;kontroluje počet parametrů
          .ERROR "XIO: chybný počet parametrů"
          .ELSE
           @CH  %2          ;zpracujeme číslo kanálu
           @CV  %1          ;a číslo povelu XIO
          STA ICCOM,X       ;... a dáme ho do IOCB
            .IF %0>=4       ;4 nebo 5 parametrů?
             @CV  %3        ;ano, zpracujeme je
            STA ICAUX1,X    ;aux1
             @CV  %4
            STA ICAUX2,X    ;aux2
            .ELSE           ;2 nebo 3 parametry
            LDA #0          ;předpokládáme nulovou hodnotu
            STA ICAUX1,X    ;pro aux1
            STA ICAUX2,X    ;i aux2
            .ENDIF
            .IF %0=2 .OR %0=4 ;bylo předáno jméno souboru?
             @FL  "S:"      ;ne...předpokládáme "S:"
            .ELSE           ;ale jestli ano,
@@IO        .=  %0          ;vezmeme číslo parametru jména
             @FL  %$(@@IO)  ;a zpracujeme jej
            .ENDIF
          JSR CIO           ;voláme OS
          .ENDIF
        .ENDM
```

Stačili jste to všechno sledovat? Trik je ve způsobu definice XIO. Můžeme mu legálně předat 2, 3, 4, nebo 5 parametrů; ale každé z těchto čísel reprezentuje jednoznačnou kombinaci parametrů:

```
XIO     povel, kanál
XIO     povel, kanál, soubor
XIO     povel, kanál, aux1, aux2
XIO     povel, kanál, aux1, aux2, soubor
```

Není to jednoduchý příklad. Možná, že nebudete mít nikdy příležitost psát něco tak složitého. MAC/65 ale nabízí nástroje i pro ty nejsložitější operace, pokud jsou třeba.

Zvláštní poznámka: Příloha B obsahuje kompletní sadu maker, které si můžete natukat a používat.

Nebo si také můžete koupit (v USA) MAC/65 TOOL KIT, obsahující tuto sadu maker a mnoho dalších pomůcek.

pozn.př.: Na disketě DOS XL, která koluje mezi našimi uživateli, je sada maker nahrána pod názvem IOMAC.LIB a dále uvedený soubor systémových definic se jmenuje SYSEQU.M65, nebo SYSEQU.ASM

KAPITOLA 6: KOMPATIBILITA

Ve světě je pro 6502 mnoho různých assemblerů, a zdá se, že každý z nich má nějaké mouchy, chyby, a zvláštnosti, (které jsou jejich promotéry nazývány charakteristikami). Ani MAC/65 není výjimkou.

Tato kapitola se zabývá věcmi, na které je třeba dát pozor při převodu programů z jiných assemblerů do MAC/65. Budeme se zabývat jen direktivami a operátory. Nebudeme se zabývat převodem mnemonik a instrukcí 6502 (předpokládáme, že každý dobrý assembler používá mnemoniky, doporučené výrobcem procesoru 6502).

Příklad: Některé antikvární assemblery používaly pro adresovací mody zvláštní mnemoniky. Konvenční LDA #3 se změnilo na LDAIMM 3 a LDA (ZIP),Y bylo LDAIY ZIP. Pro tyto zastaralé záležitosti nebyla žádná norma a proto se jimi nebudeme zabývat.

MAC/65 je přímým následníkem modulu Assembler/Editor firmy Atari (přes EASMD). Snažili jsme se dodržet kompatibilitu s A/E modulem, jak to jen bylo možné. Bohužel, v zájmu zlepšení MAC/65 bylo nutno přece jen některé věci změnit. Následující odstavec této kapitoly shrnuje tyto změny.

6.1 MODUL ASSEMBLER/EDITOR

Tento odstavec vypisuje všechny známé rozdíly mezi modulem Atari a MAC/65. MAC/65 má samozřejmě mnoho možností navíc, ale ty neovlivňují převod kódu, navrženého pro modul Atari (nebo také pro EASMD).

6.1.1 .OPT OBJ / NOOBJ

Modul Atari standardně produkuje strojový kód, i když se překládá přímo do paměti. Je to poněkud nebezpečné: snadno uděláme chybu a přepíšeme DOS, zdrojový kód v paměti, obrazovou paměť nebo i hardwarové registry.

MAC/65 dělá rozdíl mezi překladem do paměti a do souboru. Při překladu do paměti se výstupní kód generuje jen tehdy, když o to výslovně požádáme. Pokud neuvedeme direktivu ".OPT OBJ", nezapisuje se do paměti žádný kód.

6.1.2 Priorita operátorů

Modul Atari nemá určenou prioritu operátorů. MAC/65 používá prioritu operátorů podobnou, jako BASIC. Většinou to nedělá problémy, ale pozor na smíšené výrazy:

```
Příklad:                LDA # LABEL-3/256
  vyhodnotí A/E jako:   LDA # [ LABEL - 3 ] / 256
        MAC/65 jako:    LDA # LABEL - [ 3 / 256 ]
```

6.1.3 Direktiva .IF

Implementace direktivy .IF je u modulu Atari podivná a nepoužitelná. MAC/65 má tuto direktivu implementovanou podle běžných zvyklostí a podstatně použitelnější. Nejlépe je porovnat popis této direktivy v obou manuálech.

6.1.4 Dopředné odkazy na nulovou stránku

MAC/65 nemůže správně přeložit dopředné odkazy na návěští v nulové stránce (obvykle se objeví PHASE ERROR). Modul Atari tyto případy většinou přeloží, ale má jiná omezení adresových modů, která u MAC/65 nejsou.

Chybu PHASE ERROR lze obvykle odstranit přemístěním přiřazení hodnot návěštím v nulové stránce na začátek programu.

KAPITOLA 7: NOVÉ INSTRUKCE 65C02

MAC/65 tak, jak byl původně navržen, podporoval "standardní" instrukce 6502, stejně jako direktivy a adresové módy doporučené MOS Technology (výrobcem čipu 6502).

Poslední verze MAC/65 (v modulu z roku 1984) podporuje všechny původní funkce. Navíc má zabudovanou podporu pro nové funkce nejpopulárnější inovace čipu 6502. MAC/65 v nejnovější verzi podporuje všechny nové funkce a adresové módy čipu 65C02, vyráběného firmou NCR.

Popíšeme zde nejprve nové adresovací módy, potom instrukce s přidanými novými funkcemi a nakonec zcela nové instrukce.

Než ale začneme, je třeba poznamenat, že tyto instrukce fungují JEN na počítačích, ve kterých je místo 6502 (resp. 6502C) nainstalován procesor NCR 65C02. Také musíme pamatovat na to, že takovýto program může pracovat velkolepě na Vašem upraveném stroji, ale na jiném si ani neškrtne.

7.1 Hlavní přidané adresové módy

Standardní čip 6502 má dvě formy nepřímého adresování, které se v textu vyskytují takto:

```
LDA (adresa,X)
LDA (adresa),Y
```

Druhý modus, nazývaný často "nepřímý - Y", je jeden z nejužitečnějších a nejpoužívanějších adresových módů. Má ovšem jednu nevýhodu, používá dva registry, A a Y. Co je důležité, okolo 50% těchto operací má registr Y naplněno nulou.

Soubor instrukcí NCR 65C02 zde nabízí pomoc: lze použít instrukce, dovolující adresování ve stylu nepřímé-Y také v nepřímém módu, jehož formát je jednoduchý:

```
LDA (ADRESA)
```

kde adresa musí být v základní stránce, stejně jako u nepřímého indexovaného adresování. Účinek této instrukce nahradí dvojici:

```
LDY #0
LDA (adresa),Y
```

s tou výjimkou, že registr Y zůstává nezměněn a může být použit k něčemu jinému.

Následující tabulka ukazuje všechny instrukce, při nichž lze použít u 65C02 nepřímého adresování:

```
ADC (adresa)                ;součet + Carry
AND (adresa)                ;bitové AND
CMP (adresa)                ;srovnání s A-reg.
EOR (adresa)                ;vylučující OR
LDA (adresa)                ;natažení do A-reg.
ORA (adresa)                ;normální OR
SBC (adresa)                ;odečítání s Carry
STA (adresa)                ;uložení A-reg.
```

Připomínka: i když může být "adresa" kdekoliv v nulové stránce, omezíte se pravděpodobně na bezpečnou oblast s ohledem na operační systém a používaný ladicí program.

7.2 Drobné změny v instrukcích 6502

Instrukce "BIT" má přidány dva nové adresovací mody a "JMP" má přidaný jeden. Zde je popis:

Původně jsou u instrukce 6502 "BIT" povoleny tyto formy:

```
BIT absolutní
BIT stránka nula
```

nové formy 65C02 jsou navíc tyto:

```
BIT absolutní,X
BIT stránka nula,X
```

Možnost použít s instrukcí BIT registr X jako index, značně zvyšuje její použitelnost pro testování tabulek apod. Adresové mody "indexované-X" fungují stejně, jako u jiných instrukcí 6502 (LDA, CMP)

Původní formy instrukce JMP u 6502 jsou tyto:

```
JMP absolutně
JMP (nepřímo)
```

Nová forma u 65C02 je:

```
JMP (nepřímo,X)
```

Všimněte si, že instrukce JMP jako jediná používá pro nepřímé adresování absolutní (16-ti bitovou) adresu. Nová forma "nepřímo-X" není odlišná. Nepřímá adresa může být kdekoli v paměti.

Tento adresový modus je nový a unikátní. Jeho účinek je tento: Obsah X registru se přičte k ADRESE (ne k obsahu), specifikované zadanou nepřímou adresou; výsledek se použije jako skutečný operand instrukce JMP, skočí se na adresu na hranici slova, určenou skutečným operandem.

Příklad:

```
TABLE   .WORD   SUB1,SUB2,SUB3
        ...
        LDA     hodnota         ;předpokládáme, že hodnota
                                ;je 0, 1, nebo 2
        ASL A                   ;hodnota * 2
        TAX                     ;do X registru
        JMP   (TABLE,X)         ;skočí na SUB1, SUB2, nebo
                                ;SUB3 podle hodnoty
```

7.3 ZCELA NOVÉ INSTRUKCE 65C02

Zde uvádíme v logickém seskupení instrukce 65C02, které jsou ve světě 6502 opravdu nové:

7.3.1 BRA

mnemonika: BRA

čti jako: BRAnch

formát: BRA addr
kde addr musí být v rozsahu *-126 až *+129 (* je hodnota čítače adresy)

Poznámky: BRA patří do rodiny instrukcí Branch (BNE, BEQ, BMI, apod.) a přidává užitečnou možnost nepodmíněného skoku. Je ekvivalentní instrukci JMP, ale obsazuje jen 2 byty a je relokatibilní. Její adresový rozsah je omezen stejně jako u ostatních instrukcí této skupiny.

Status procesoru, časování apod. jsou velmi podobné instrukcím INX/INY/DEX/DEY.

7.3.3 PHX, PHY, PLX A PLY

mnemonika:
PHX
PHY
PLX
PLY

čti jako:
PusH X onto CPU stack
PusH Y onto CPU stack
PulL X from CPU stack
PulL Y from CPU stack

formát:
PHX
PHY
PLX
PLY

Poznámky: Tyto instrukce jsou navrženy pro zkrácení sekvencí, nutných u 6502. Jako příklad, PHX nahrazuje sekvenci:

```
STA  temp
TXA
PHA
LDA  temp
```

Díky přímému přístupu do zásobníku z registrů X a Y je možno vytvářet kompaktnější a snadněji přemistitelný kód. Status procesoru, nastavení příznaků a časování jsou stejné jako u podobných instrukcí PHA a PLA.

7.3.4 STZ

mnemonika: STZ

čti jako: STore Zero

formát:
```
STZ absolutní
STZ absolutní,X
STZ nulová stránka
STZ nulová stránka,X
```

Poznámky: Opět zkrácení standardních postupů. Nahrazuje sekvenci:
```
      LDA #0
      STA adresa
```
s tím rozdílem, že obsah A-reg. není dotčen. Přesná simulace této instrukce na 6502 je:
```
      PHA
      LDA #0
      STA adresa
      PLA
```

7.3.5 TRB a TSB

mnemonika:
```
TRB
TSB
```

čti jako:
```
Test and Reset Bits
Test and Set Bits
```

formát:
```
TRB absolutní
TRB nulová stránka
TSB absolutní
TSB nulová stránka
```

Poznámky: Tyto instrukce mají mnoho použití, například při synchronizaci procesů v hlavním programu a v pozadí (rutiny řízené přerušením). V booleovském vyjádření pracují instrukce takto:

```
TRB:    paměť := (Not A) and paměť
TSB:    paměť := A or paměť
```

Slovy můžeme popsat průběh operací takto:
Pro TRB: S doplňkem obsahu akumulátoru se provede bit po bitu logický součin s obsahem paměťové buňky, adresované operandem instrukce. Výsledek logického součinu se uloží do adresované buňky.

Pro TSB: Obsah akumulátoru se bit po bitu logicky sečte s obsahem paměťové buňky, jejíž adresa je v operandu instrukce. Výsledek se uloží zpět do adresované buňky.

Je-li výsledek logické operace nula, nastaví se bit Z ve stavovém registru procesoru. Bity N a V se naplní obsahem bitů 6 a 7 paměťové buňky před logickou operací (jako u instrukce BIT).

Příklady:

```
FLAG .BYTE 3
TEST .BYTE $FF
     ...
     LDA #$FF
     TRB FLAG  ;vynulování všech bitů
     ...
     LDA #0
     TSB TEST  ;jen testuje hodnotu
```

KAPITOLA 8: TECHNIKY PROGRAMOVÁNÍ S MAC/65

Tato kapitola uvádí soubor doporučení k dosažení maximální účinnosti MAC/65.

8.1 Obsazení paměti u MAC/65 a DDT

Tabulka ukazuje úseky paměti, používané MAC/65 a DDT a účel a okamžik jejich použití:

interval adres	používá MAC/65	DDT	k čemu
$80-$AF	ano	ano	ukazatele a pracovní paměť
$B0-$D3	ano	ne	- ,, -
$D4-$FF	ano	ne	registry pohyblivé řádové čárky
$100-$1FF	ano	ano	zásobník procesoru
$3FD-$47F	ne	ano	buffery a oblast obrazovky
$480-$57F	ano	ano	buffery a pracovní oblast
$580-$67F	ano	ne	vstupní buffery apod.
"size"	ano	*	text programu atd.

Poznamenejme, že "size" označuje oblast mezi levou a prostřední adresou při výpisu povelem "SIZE" editoru MAC/65. * ve sloupci DDT znamená, že DDT ukládá nulovou stránku MAC/65 (a některé další důležité buňky) do oblasti "size".

Nejhorším omezením, vyplývajícím z rozdělení paměti je (zejména pro uživatele Atari BASICu) použití části stránky 6, protože řada programů v časopisech předpokládá, že je stránka 6 volná. Tyto programy nepracují pod MAC/65 a DDT tak, jak jsou. Prostudujte proto další odstavec, kde jsou popsány metody, které je nutno uplatnit, pokud MUSÍTE používat stránku 6.

8.2 Překlad s posunem: .SET 6

V odstavci 4.19 jsme uvedli, že direktiva ".SET 6,hodnota", může být použita k určení aditivní konstanty, která se před uložením kódu do souboru nebo paměti přičítá k adrese. V tomto odstavci ukážeme metodu, jak toho užitečně využít.

Předpokládejme, že chceme přeložit programek do stránky 6 ($600-$6FF). Program vypadá takto:

```
10      *= $600
20 COLOR4 = $2C8
30 ;
40 START
50      PLA      ;počet parametrů ze zásobníku
60      CMP #0   ;jsou parametry?
70      BEQ *    ;nejsou=> nekonečná smyčka
80      LDA COLOR4 ;načti hodnotu barvy pozadí
90      CLC
100     ADC #$10    ;zvyš světelnost o 1
110     STA COLOR4 ;zapiš hodnotu do stínového registru
120     INC COUNT  ;a zvyš počítadlo
130     RTS
140 COUNT .BYTE 0 ;jednobytové počítadlo
150     .END
```

Přeložíme-li tuto rutinku, bude překlad bez chyby. (Uživatel BASICu bezpochyby pozná, že jde o podprogram pro ATARI BASIC podle instrukce PLA na začátku a podle kontroly počtu parametrů.)

Procedura byla ale navržena pro stránku 6. Jak to zařídíme? Přidáme následující dva řádky:

```
12      .OPT OBJ
14      .SET 6,$3000
```

Nyní je přeložený kód umístěn od adresy $3600. Překladač opravdu umístí kód na tuto adresu. Podívejme se ale na řádku 120. Strojová instrukce nezvyšuje adresu $3612, ale $612. V tabulce symbolů má návěští START hodnotu $600 a COUNT $612. Nyní přejdeme do DDT a povelem

```
M 360006000080 [RETURN]
```

překopírujeme $80 (128) bytů z $3600 na $600. Povelem DDT

```
* 0600 [RETURN]
```

si můžeme prohlédnout obsah adresy $600 a dalších. Na prohlížení je možno používat kurzorové klávesy (bez CTRL). Program je umístěn tam, kam patří a můžeme jej ladit a testovat.

Několik poznámek na konec: MAC/65 může vytvořit tento "posunutý" kód jak do paměti, tak do souboru. Je-li potom takový soubor načten DOSem, nebo povelem BLOAD, uloží se do paměti na místo podle listingu. Toto místo je vypočteno jako součet hodnoty čítače adresy a hodnoty druhého parametru direktivy .SET. Je na nás, abychom si kód přesunuli tam, kam patří. Tato technika není jednoduchá, ale s její pomocí můžeme přepsat DOS, nebo vytvořit kód, který má pracovat v oblasti modulů. U .SET 6 můžeme použít i záporný posun. Je to naprosto legální.

Pozn.př.: Pokud překládáme zdrojový program s výstupem do souboru, nebo zařízení, nejsme co do umístění programu v adresové oblasti nijak omezeni. Bez problému můžeme překládat i rutiny určené pro stránku 6 i jiné. Pokud použijeme jiný ladicí program, než DDT, nebudeme mít se stránkou 6 problémy.

8.3 JAK MAC/65 JEŠTĚ ZRYCHLIT

Jestliže včleníme soubor, obsahující jen přiřazení (např. SYSEQU.M65), nebo definice (ne volání!) maker (IOMAC.LIB), existuje technika,

kterou lze překlad poněkud zrychlit.

V zásadě není důvod, proč by se měla přiřazení a definice maker číst ve druhém průchodu znovu. Následující program ukazuje, jak čtení definic ve druhém průchodu potlačit:

```
        *= 0
PASS    .= PASS+1       ;tohle se udělá jen jednou
        .IF PASS=1
          .INCLUDE #D:soubor definic
          .ENDIF
        *= začátek
```

Proč to funguje: Normálně má nedefinované návěští hodnotu 0. Direktiva .= způsobí podivnou věc: nedefinované návěští, použité na pravé straně direktivy dostane momentální hodnotu čítače adresy. Proto nastavíme čítač adresy na 0 příkazem "*=0".

V každém případě dostane návěští PASS při prvním průchodu překladu hodnotu 1 (a vybudí chybu "undefined label", chyby se ale při prvním průchodu nevypisují). Při druhém průchodu dostane PASS hodnotu 2. Jednoduché a účinné.

Pokud by se ".=" umístilo před jakoukoliv instrukcí "*=", není vlastně první řádka našeho příkladu třeba, protože čítač adresy má počáteční hodnotu 0, pokud není určeno jinak.

Příloha A: Systémová přiřazení

Zde uvádíme výpis určitých systémových adres, které považujeme za užitečné a nutné při programování na počítačích Atari.

Mnoho z těchto přiřazení se vztahuje na DOS XL. Pokud pracujete se systémovými zdroji (jako IOCB, nebo CIO), jsou adresy pro DOS Atari to-ožné. Pokusili jsme se zvláště označit adresy, spojené pouze s DOSem XL (zvláště provádění dávek povelů a povelovou řádku).

Některé řádky se mohou poněkud lišit od adres, publikovaných firmou Atari (ve výpisu operačního systému) nebo v publikovaných knihách (Mapping the Atari - COMPUTE! books). Rozdíly jsou minimální (např. ICAX1 místo ICAUX1).

Můžete si tento výpis naťukat a uložit povelem "SAVE" na disk, nebo kazetu. Pokud jej uložíme na disk, můžeme jej povelem .INCLUDE používat ve svých programech. Stejně dobře můžete použít tento výpis jako informaci a do svých programů opsat jen ty adresy, které potřebujete.

```
1000      .PAGE "OSS SYSTEMOVA PRIRAZENI PRO ATARI"
1010 ;
1020 ;  NAZEV = #DN:SYSEQU.M65
1030 ;
1040 ;
1050 ; PRIRAZENI RIDICICH BLOKU I/O OPERACI
1060 ;
1065 SAVEPC = *       ; USCHOVANI CITACE ADRESY
1067 ;
1070      *=  $0340   ;ZACATEK RIDICICH BLOKU IOCB
1075 IOCB
1080 ;
1090 ICHID *= *+1     ;HANDLER ZARIZENI, NASTAVUJE OS
1100 ICDNO *= *+1     ;CISLO ZARIZENI, NASTAVUJE OS
1110 ICCOM *= *+1     ;POVEL I/O
1120 ICSTA *= *+1     ;STATUS I/O
1130 ICBADR *= *+2    ;ADDRESA BUFFERU
1140 ICPUT *= *+2     ;ADR-1 PUT RUTINY
1150 ICBLEN *= *+2    ;DELKA BUFFERU
1160 ICAUX1 *= *+1    ;AUX 1
1170 ICAUX2 *= *+1    ;AUX 2
1180 ICAUX3 *= *+1    ;AUX 3
1190 ICAUX4 *= *+1    ;AUX 4
1200 ICAUX5 *= *+1    ;AUX 5
1210 ICAUX6 *= *+1    ;AUX 6
1220 ;
1230 IOCBLEN = *-IOCB ;DELKA JEDNOHO IOCB
1240 ;
1250 ; PRIRAZENI HODNOTY POVELU IOCB
1260 ;
1270 COPN =  3        ;OPEN
1280 CGBINR = 7       ;GET-CTENI RADY BYTU
1290 CGTXTR = 5       ;CTENI VETY (KONCI EOL)
1300 CPBINR = 11      ;ZAPIS RADY BYTU
1310 CPTXTR = 9       ;ZAPIS VETY
1320 CCLOSE = 12      ;CLOSE
1330 CSTAT = 13       ;CIST STATUS
```

```
1340 ;
1350 ; POVELY SPECIFICKE PRO FILE MANAGER
1360 ;
1370 CREN =  32       ;RENAME
1380 CERA =  33       ;ERASE
1390 CPRO =  35       ;PROTECT
1400 CUNP =  36       ;UNPROTECT
1410 CPOINT = 37      ;POINT
1420 CNOTE = 38       ;NOTE
1430 ;
1440 ; HODNOTY AUX1 POTREBNE PRO OPEN
1450 ;
1460 OPIN =  4        ;OTEVRENI PRO VSTUP
1470 OPOUT = 8        ;OTEVRENI PRO VYSTUP
1480 OPUPD = 12       ;PRO UPDATE
1490 OPAPND = 9       ;PRO APPEND
1500 OPDIR = 6        ;OTEVRENI KATALOGU DISKETY
1510 ;
1520     .PAGE
1530 ;
1540 ;    DEFINICE PROVADECICH PRIZNAKU (DOS XL)
1550 ;
1560 EXCYES = $80     ; PROVADENI POKRACUJE
1570 EXCSCR = $40     ; OPAKUJ VSTUPY NA OBRAZOVKU
1580 EXCNEW = $10     ; VYKONEJ START UP MODUS
1590 EXCSUP = $20     ; PRIZNAK PROVEDENI STUDENEHO STARTU
1600 ;
1610 ; RUZNA PRIRAZENI
1620 ;
1630 CPALOC = $0A     ; UKAZATEL NA CP/A
1640 WARMST = $08     ; TEPLY START (0=COLD)
1650 MEMLO = $02E7    ; UKAZATEL DOSTUPNE PAMETI (DOLNI)
1660 MEMTOP = $02E5   ; UKAZATEL DOSTUPNE PAMETI (HORNI)
1670 APPMHI = $0E     ; HORNI LIMIT VYUZITI PAMETI
1680 INITADR = $02E2  ; INICIALIZACNI ADRESA
1690 GOADR = $02E0    ; ROZBEHOVA ADRESA
1700 CARTLOC = $BFFA  ; ROZBEHOVA ADRESA MODULU
1710 CIO =    $E456   ; ADRESA CIO
1720 EOL =    $9B     ; ZNAK KONEC RADKY
1730 ;
1740 ;  FUNKCNI A HODNOTOVE INDEXY CP/A
1750 ;      (NEPRIMO PRES  CPALOC)
1760 ;            TJ. (CPALOC),Y
1770 ;
1780 CPGNFN = 3       ; DALSI JMENO SOUBORU
1790 CPDFDV = $07     ; STANDARDNI DISK (3 BYTY)
1800 CPBUFP = $0A     ; UKAZATEL NA DALSI ZNAK POVELOVEHO BUFFERU
1810 CPEXFL = $0B     ; PROVADECI PRIZNAK
1820 CPEXFN = $0C     ; SOUBOR K PROVEDENI (16 BYTE)
1830 CPEXNP = $1C     ; HODNOTY NOTE/POINT PRO VYKONANI
1840 CPFNAM = $21     ; BUFFER JMENA SOUBORU
1850 RUNLOC = $3D     ; ZAVADECI/ROZBEHOVA ADRESA CP/A
1860 CPCMDB = $3F     ; BUFFER POVELU (60 BYTU)
1870 CPCMDGO = $F3
1880 ;
1890     *=  SAVEPC   ; OBNOVENI CITACE ADRESY
1900 ;
```

Příloha B: Některé užitečné makroinstrukce

V následujícím textu uvádíme výpis několika makroinstrukcí. Tyto makroinstrukce jsou navrženy pro snadné ovládání I/O operací. Napíšete-li si je tak, jak jsou napsány, získáte užitečnou knihovnu maker.

Doporučujeme makra napsat a uložit na disk, nebo kazetu. Pokud je uložíme na disk, můžeme je začlenit do programu povelem .INCLUDE. Makra, uložená na kazetu můžeme do programu zatřídit, nebo připojit.

Upozornění: Tato makra využívají soubor systémových přiřazení, uvedený v příloze A. Můžete jej do programu začlenit pomocí .INCLUDE, nebo napsat před definice maker. (Která přiřazení je třeba napsat, se zjistí zkušebním překladem bez souboru definic. Chybějící návěští způsobí chyby při překladu.

Před výpisem maker uvádíme jejich souhrn v předpokládaném pořadí použití. Připomínáme, že použití makra znamená uvedení jeho jména ve sloupci instrukce, eventuelně i s parametry ve sloupci operandů.

```
OPEN kanál,aux1,aux2,filename
     Otvírá zadaný kanál pro uvedený soubor s použitím
     definice přístupu aux1 a aux2.

PRINT kanál [ ,buffer [ ,délka ] ]
     Není-li zadán buffer, tiskne přes kanál jen EOL.
     Není-li zadána délka, předpokládá se 255, nebo pozice
     prvního znaku EOL, pokud je menší. Je-li buffer textová
     konstanta (přímý operand), ignoruje se délka, pokud
     byla uvedena.

INPUT kanál [ ,buffer [ ,délka ] ]
     Není-li zadána délka, předpokládá se 255 bytů.

BGET kanál,buffer,délka
     Binární čtení ve stylu BASIC XL. Do bufferu se čtou
     byty, jejichž počet je určen délkou.

BPUT kanál,buffer,délka
     Binární zápis bytů od zadané adresy bufferu. Jejich
     počet je určen délkou.

CLOSE kanál
     Zavření kanálu.

XIO povel,kanál [ ,aux1,aux2 ] [ jméno souboru ]
     Je popsáno v kapitole 5.
```

Poznámky:

"Kanál" může být buď konstanta (0 až 7), nebo adresa paměti, obsahující číslo kanálu (0 až 7).

"aux1", "aux2", "délka" a "povel" mohou být buď konstanty (0 - 255), nebo místa v paměti.

"jméno souboru" může být buď textová konstanta (např. D:FILE1.DAT), nebo místo v paměti, kde se předpokládá

začátek stringu, obsahujícího specifikaci souboru.

Pokud jsou místo konstant použity paměťové adresy, nesmí být ve stránce nula (t.j. mezi 0 až $FF) a musí být definovány PŘED použitím ve volání makra.

Tohle se nesmí napsat:

```
            PRINT 3,MESSAGE1     ;spatně!
              ...
MESSAGE1 .BYTE "Tohle nebude fungovat!!!"
```

Uvedené makroinstrukce jsou užitečným nástrojem, ale jsou míněny jen jako příklady, které mají ukázat, co lze s MAC/65 dělat. Prostudujte si je a změňte je podle svých potřeb.

```
1000      .TITLE "IOMAC.LIB -- OSS systemova I/O makra"
1010      .PAGE "   Podpora pro hlavni makra"
1020        .IF .NOT .DEF IOCB
1030        .ERROR "Musite pred makra zaclenit SYSEQU.M65!!"
1040        .ENDIF
1050 ;
1060 ; Tato makra jsou volana makroinstrukcemi I/O
1070 ; aby pripravila obsahy registru.
1080 ;
1090 ;
1100 ; MAKRO:  @CH
1110 ;
1120 ; Nacita cislo IOCB (parametr 1) do registru X
1130 ;
1140 ; Je-li hodnota parameru 0 az 7, predpoklada se konstanta
1150 ;   cisla kanalu
1160 ;
1170 ; Je-li hodnota  > 7, předpokládá se misto v
1180 ;   pameti, obsahujici cislo kanalu.
1190 ;
1200      .MACRO @CH
1210        .IF %1>7
1220        LDA %1
1230        ASL A
1240        ASL A
1250        ASL A
1260        ASL A
1270        TAX
1280        .ELSE
1290        LDX #%1*16
1300        .ENDIF
1310      .ENDM
```

```
1320 ;
1330 ;
1340 ; MAKRO:  @CV
1350 ;
1360 ; Naplni do akumulatoru hodnotu, nebo konstantu
1370 ;
1380 ; Je-li hodnota parametru 0-255, @CV
1390 ; predpoklada konstantu
1400 ;
1410 ; Jinak se predpoklada adresa v pameti,
1420 ; mimo zakladni stranku, obsahujici hodnotu
1430 ;
1440 ;
1450 ;
1460     .MACRO @CV
1470       .IF %1<256
1480       LDA #%1
1490       .ELSE
1500       LDA %1
1510       .ENDIF
1520     .ENDM
1530 ;
1540 ;
1550 ;
1560 ;
1570 ; MAKRO:  @FL
1580 ;
1590 ; @FL se pouziva k urceni specifikace souboru
1600 ;
1610 ; Je-li predano jako primy operand, @FL
1620 ; vygeneruje nazev v radce, preskoci ji
1630 ; a umisti jeji adresu v IOCB, na
1640 ; ktery ukazuje X-registr.
1650 ;
1660 ; Je-li predano navesti mimo zakladni
1670 ; stranku, predpoklada se, ze jde o
1680 ; navesti platne specifikace a pouzije jeho adresu.
1690 ;
1700 ;
1710 ;
1720     .MACRO @FL
1730       .IF %1<256
1740       JMP *+%1+4
1750 @F    .BYTE %$1,0
1760       LDA # <@F
1770       STA ICBADR,X
1780       LDA # >@F
1790       STA ICBADR+1,X
1800       .ELSE
1810       LDA # <%1
1820       STA ICBADR,X
1830       LDA # >%1
1840       STA ICBADR+1,X
1850       .ENDIF
1860     .ENDM
```

```
1870      .PAGE "   makro XIO "
1880 ;
1890 ; MAKRO:  XIO
1900 ;
1910 ; VOLANI: XIO povel,kanal [,aux1,aux2][,filespec]
1920 ;
1930 ; kanal je jako v makru @CH.
1940 ; povel, aux1, aux2 jsou jako v makru @CV
1950 ; filespec je jako v makru @FL
1960 ;
1970 ; provadi zname operace XIO s/pro OS/A+
1980 ;
1990 ; Je-li dano aux1, musi byt zadano i aux2
2000 ; Jsou-li aux1 a aux2 vynechany, predpokladaji se nulove
2010 ; Je-li vynechano filespec, predpoklada se "S:"
2020 ;
2030      .MACRO XIO
2040        .IF %0<2 .OR %0>5
2050        .ERROR "XIO: chybny pocet parametru"
2060        .ELSE
2070         @CH  %2
2080         @CV  %1
2090        STA ICCOM,X ; COMMAND
2100          .IF %0>=4
2110           @CV  %3
2120          STA ICAUX1,X
2130           @CV  %4
2140          STA ICAUX2,X
2150          .ELSE
2160          LDA #0
2170          STA ICAUX1,X
2180          STA ICAUX2,X
2190          .ENDIF
2200          .IF %0=2 .OR %0=4
2210           @FL  "S:"
2220          .ELSE
2230 @@IO      .=  %0
2240           @FL  %$(@@IO)
2250          .ENDIF
2260        JSR CIO
2270        .ENDIF
2280      .ENDM
```

```
2290      .PAGE "   makro OPEN "
2300 ;
2310 ; MAKRO:  OPEN
2320 ;
2330 ; VOLANI: OPEN kanal,aux1,aux2,filespec
2340 ;
2350 ; kanal je jako v makru @CH.
2360 ; povel, aux1, aux2 jsou jako v makru @CV
2370 ; filespec je jako v makru @FL
2380 ;
2390 ; pokousi se otevrit zadany soubor pres
2400 ; zadany kanal s pouzitim "modu",
2410 ; specifikovanych aux1 a aux2
2420 ;
2430      .MACRO OPEN
2440        .IF %0<>4
2450        .ERROR "OPEN: chybny pocet parametru"
2460        .ELSE
2470          .IF %4<256
2480           XIO  COPN,%1,%2,%3,%$4
2490          .ELSE
2500           XIO  COPN,%1,%2,%3,%4
2510          .ENDIF
2520        .ENDIF
2530      .ENDM
```

```
2540      ,PAGE "   makra BGET and BPUT "
2550 ;
2560 ; MAKRA: BGET and BPUT
2570 ;
2580 ; VOLANI: BGET kanal,buffer,delka
2590 ;         BPUT kanal,buffer,delka
2600 ;
2610 ; kanal je jako v makru @CH
2620 ; delka je VZDYCKY primy operand
2630 ;   a skutecna hodnota, nikdy ne adresa
2640 ; buffer musi byt adresa prislusne vyrovnavaci
2650 ;   pameti
2660 ;
2670 ; pise nebo cte "delka" bytu z/do
2680 ;   urceneho bufferu, uziva binarni cteni/zapis
2690 ;
2700 ;
2710 ; nejprve spolecne makro
2720 ;
2730      .MACRO @GP
2740       @CH  %1
2750      LDA #%4
2760      STA ICCOM,X
2770      LDA # <%2
2780      STA ICBADR,X
2790      LDA # >%2
2800      STA ICBADR+1,X
2810      LDA # <%3
2820      STA ICBLEN,X
2830      LDA # >%3
2840      STA ICBLEN+1,X
2850      JSR CIO
2860      .ENDM
2870 ;
2880      .MACRO BGET
2890        .IF %0<>3
2900        .ERROR "BGET: chybny pocet parametru"
2910        .ELSE
2920         @GP  %1,%2,%3,CGBINR
2930        .ENDIF
2940      .ENDM
2950 ;
2960      .MACRO BPUT
2970        .IF %0<>3
2980        .ERROR "BPUT: chybny pocet parametru"
2990        .ELSE
3000         @GP  %1,%2,%3,CPBINR
3010        .ENDIF
3020      .ENDM
3030 ;
```

```
3040      .PAGE "   makro PRINT "
3050 ;
3060 ; MAKRO:   PRINT
3070 ;
3080 ;  FORM:   PRINT kanal[,buffer[,length]]
3090 ;
3100 ; kanal je jako v makru @CH
3110 ; neni-li udan buffer, tiskne RETURN
3120 ; neni-li delka, predpoklada se 255
3130 ;
3140 ; Uziva se k tisku textu. Bez RETURNu.
3150 ; musi byt zadana delka. Viz OS/A+ manual
3160 ;
3170 ; VYJIMKA: druhy parametr muze byt textova
3180 ;  konstanta (napr. PRINT 0,"test"), v tomto
3190 ;  pripade se delka ignoruje.
3200 ;
3210      .MACRO PRINT
3220        .IF %0<1 .OR %0>3
3230        .ERROR "PRINT: chybny pocet parametru"
3240        .ELSE
3250          .IF %0>1
3260            .IF %2<128
3270            JMP *+4+%2
3280 @IO        .BYTE %$2,$9B
3290             @GP  %1,@IO,%2+1,CPTXTR
3300            .ELSE
3310              .IF %0=2
3320               @GP  %1,%2,255,CPTXTR
3330              .ELSE
3340               *@GP  %1,%2,%3,CPTXTR
3350              .ENDIF
3360            .ENDIF
3370          .ELSE
3380          JMP *+4
3390 @IO      .BYTE $9B
3400           @GP  %1,@IO,1,CPTXTR
3410          .ENDIF
3420        .ENDIF
3430      .ENDM
3440 ;
```

```
3450      .PAGE "   makro INPUT "
3460 ;
3470 ; MAKRO:   INPUT
3480 ;
3490 ;  FORM:   INPUT kanal,buffer,delka
3500 ;
3510 ; kanal je jako v makru @CH
3520 ; buffer MUSI byt spravna adresa buferu
3530 ; delka muze byt vynechana, pak se predpoklada 255
3540 ;
3550 ; cte radku textu do zadane vyrovnavaci
3560 ;   pameti, maximalne "delka" bytu
3570 ;
3580      .MACRO INPUT
3590        .IF %0<2 .OR %0>3
3600        .ERROR "INPUT: chybny pocet parametru"
3610        .ELSE
3620          .IF %0=2
3630           @GP  %1,%2,255,CGTXTR
3640          .ELSE
3650           @GP  %1,%2,%3,CGTXTR
3660          .ENDIF
3670        .ENDIF
3680      .ENDM
3690      .PAGE "   makro CLOSE "
3700 ;
3710 ; MAKRO:   CLOSE
3720 ;
3730 ; VOLANI: CLOSE kanal
3740 ;
3750 ; kanal je jako v makru @CH
3760 ;
3770 ; zavira zadany kanal
3780 ;
3790      .MACRO CLOSE
3800        .IF %0<>1
3810        .ERROR "CLOSE: chybny pocet parametru"
3820        .ELSE
3830         @CH  %1
3840        LDA #CCLOSE
3850        STA ICCOM,X
3860        JSR CIO
3870        .ENDIF
3880      .ENDM
3890 ;
3900 ;;;;;;;;;;;; KONEC IOMAC.LIB ;;;;;;;;;;;;
3910 ;
```

Příloha C: POPIS CHYB

Při výskytu chyby napíše systém
    *** ERROR -
následované číslem chyby (pokud nebyla chyba generována direktivou .ERROR) a u většiny chyb krátký popis (anglicky)

Poznámka: Překladač může vytisknout až 3 chyby na jedné řádce.

Formát následujícího popisu chyb je jednoduchý: číslo chyby, chybová zpráva, popis a možné příčiny chyby.

1 - MEMORY FULL
Byla použita celá uživatelská paměť. Je-li tato chyba vypsána editorem, nelze již do paměti vložit žádnou zdrojovou řádku. Pokud vypíše tuto chybu překladač, nelze definovat další návěští a makra.
Poznámka: Pokud se tato chyba vyskytne při překladu, a máte-li v paměti zdrojový program, uložte jej pomocí SAVE, napište NEW a překládejte ze souboru. Překladač nyní může použít paměť, která byla dříve obsazena textem.
Pozn.př.: Tuto chybu lze také způsobit tím, že si nastavíme LOMEM vysoko pro ladění a při následné opravě a rekompilaci programu zapomeneme LOMEM vrátit zpět.

2 - INVALID DELETE
Buď není v paměti první řádka, nebo je číslo druhé řádky menší, než první.

3 - BRANCH RANGE
Relativní instrukce má adresový interval větší, než +129, nebo menší, než -126 bytů od adresy výskytu.

4 - NOT ZERO PAGE/IMMEDIATE MODE
Výraz pro nepřímé adresování, nebo přímý operand má po vyčíslení hodnotu > 255 ($FF)

5 - UNDEFINED
Překladač narazil na nedefinované návěští.

6 - EXPRESSION TOO COMPLEX
Přetekl zásobník operátorů. Pokud je nezbytné použít tak složitý výraz, že vybudil tuto chybu, zkuste jej rozdělit na menší pomocí dočasného nastavení hodnoty návěští (tedy .=).

7 - DUPLICATE LABEL
Překladač nalezl ve sloupci návěští název, který se již v tomto sloupci jednou vyskytoval.

8 - BUFFER OVERFLOW
Přetečení bufferu pro syntaktickou kontrolu (editor). Zkraťte vstupní řádku.

9 - CONDITIONALS NESTING
Konstrukce .IF-.ELSE-.ENDIF není správně vnořena do jiné. Protože MAC/65 nemůže identifikovat nadbytečné .ENDIFy, musí být chyba způsobena přebytečným .ELSE, nebo .ENDIF

10 - VALUE > 255
Výsledek výrazu, určeného pro umístění v jednom bytu, překročil 255.

11 - CONDITIONAL STACK
Vnořené konstrukce .IF-.ELSE-.ENDIF přesáhly hloubku 14-ti úrovní.

12 - NESTED MACRO DEFINITION
Překladač narazil na další direktivu .MACRO před direktivou .ENDM. Tato chyba zhroutí překlad.

13 - OUT OF PHASE
Adresa vygenerovaná pro návěští při prvním průchodu se liší od adresy, vygenerované při druhém průchodu. Obvyklou příčinou těchto chyb jsou dopředné reference adres. Pokud používáme podmíněný překlad (s makry i bez nich), může chyba vzniknout tím, že podmínka za .IF se při prvním průchodu vyhodnotí jako "nepravda" a při druhém jako "pravda".

14 - *= EXPRESSION UNDEFINED
Dopředná reference čítače adresy.

15 - SYNTAX OVERFLOW
Editor nemůže syntakticky analyzovat zdrojovou řádku. Zjednodušte složité výrazy nebo rozdělte řádku na více částí.

16 - DUPLICATE MACRO NAME
Pokus definovat další makro pod stejným jménem. Platí pouze první definice.

17 - LINE # > 65535
Editor nemůže akceptovat číslo řádky větší, než 65535.

18 - MISSING .ENDM
V definici makra se našel EOF (konec souboru) před odpovídajícím terminátorem .ENDM. Makrodefinice nemohou překročit hranici souboru. Tato chyba zhroutí překlad.

19 - NO ORIGIN
V programu chybí direktiva *= .
Poznámka: Tato chyba se objeví jen při zápisu strojového kódu.

20 - NUM/REN OVERFLOW
Při povelech REN a NUM vzniklo číslo řádky větší, než 65535. Vznikla-li chyba při REN, lze chybu napravit zadáním REN se správnými parametry. Pokud vznikla chyba při NUM, automatické číslování se přeruší.

21 - NESTED .INCLUDE
Včleňovaný soubor nesmí obsahovat další .INCLUDE

22 - LIST OVERFLOW
Výstupní buffer pro LIST překračuje 255 znaků. Použijte menší čísla v direktivě .TAB.

23 - NOT SAVE FILE
Pokus o zavedení nebo překlad souboru, který nebyl vytvořen

povelem SAVE.

24 - LOAD TOO BIG
Tokenizovaný soubor se nevejde do paměti.

25 - NOT BINARY SAVE
Soubor není platný binární (strojový, object) program.

26 - není v originále uvedeno

27 - INVALID .SET
První parametr u .SET určuje neexistující systémový parametr překladače.

28 - 29 není v originále uvedeno

30 - UNDEFINED MACRO
Překladač narazil na volání makra, které nebylo definováno. Makra musí být před použitím definována.

31 - MACRO NESTING
Maximální hloubka vnořených maker překročila 14 úrovní.

32 - BAD PARAMETER
Při volání makra byl učiněn odkaz na neexistující parametr, nebo číslo specifikovaného parametru je větší, než 63.

128 - 255 [chyby operačního systému]
Čísla chyb, větší než 128, generuje příslušný (OS ROM + DOS) operační systém. Prostudujte příslušný manuál.

OBSAH

Úvod 1
Zapnutí 5
Teplý start 5
Syntaxe 6

Kapitola 1 -- Editor 7
1.1 Všeobecné užití editoru 7
1.2 TEXT modus 7
1.3 EDIT modus 8

Kapitola 2 -- Povely editoru 9
2.1 ASM překlad programu 9
2.2 BLOAD binární čtení 10
2.3 BSAVE binární ukládání 11
2.4 BYE přechod do selftestu 11
2.5 DDT přechod do ladicího programu . . 11
2.6 DEL vymazání řádek 12
2.7 DOS přechod do DOSu 12
2.8 ENTER čtení textového souboru(ATASCII) 13
2.9 FIND hledání textového řetězce . . . 13
2.10 LIST výpis programu v paměti 14
2.11 LOAD načtení uloženého programu . . . 14
2.12 LOMEM nastavení nové hranice paměti . 15
2.13 NEW vymazání textové paměti 15
2.14 NUM automatické číslování řádek . . 15
2.15 PRINT výpis programu bez čísel řádek . 16
2.16 REN přečíslování řádek 16
2.17 REP nahrazení textového řetězce . . 16
2.18 SAVE uložení zdrojového textu MAC/65 17
2.19 SIZE informace o obsazení paměti . . 17
2.20 TEXT přepnutí do textového modu . . . 17
2.21 ? konverze dek/hexa 18

Kapitola 3 -- Makroassembler 19
3.1 vstup do překladače 19
3.2 formát instrukcí 20
3.3 návěští 21
3.4 operandy 21
3.5 operátory 22
3.6 výrazy v assembleru 25
3.7 priorita operátorů 26
3.8 numerické konstanty 26
3.9 Stringy (textové řetězce) 27

Kapitola 4 -- direktivy (pseudoinstrukce) 28
4.1 *= (a .ORG) 28
4.2 = (a .EQU) 29
4.3 .= 29
4.4 .BYTE (A .SBYTE) 29
4.5 .CBYTE 31
4.6 .DBYTE 31
4.7 .DS 32
4.8 .ELSE 32
4.9 .END 32
4.10 .ENDIF 32
4.11 .ERROR 32
4.12 .FLOAT 33

4.13 .IF 33
4.14 .INCLUDE 34
4.15 .LOCAL 35
4.16 .OPT 35
4.17 .PAGE 37
4.18 .SBYTE (viz též .BYTE) 37
4.19 .SET 37
4.20 .TAB 38
4.21 .TITLE 39
4.22 .WORD 39

Kapitola 5 -- makroinstrukce 40
5.1 .ENDM 40
5.2 .MACRO 40
5.3 volání makra, část 1 41
5.4 parametry maker 42
5.5 volání makra, část 2 43
5.6 stringové parametry 44
5.7 doporučení pro makra 45
5.8 komplexní příklad makroinstrukcí 46

Kapitola 6 -- kompatibilita 49
6.1 modul Atari Assembler/Editor 49

Kapitola 7 -- nové instrukce 65C02 51
7.1 hlavní přidané adresové módy 51
7.2 drobné změny v instrukcích 6502 52
7.3 zcela nové instrukce 65C02 53

Kapitola 8 -- techniky programování 55
8.1 obsazení paměti u MAC/65 a DDT 55
8.2 překlad s posunem: .SET 6 55
8.3 zrychlení MAC/65 56

příloha A -- výpis systémových přiřazení 58

příloha B -- příklady výpisů makroinstrukcí 60

příloha C -- popisy chyb. 68

Igi/2019