zpravodaj atari klubu

IV/87

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., F. Tvrdek, ing. M. Vavrda.
Otisk povolen se souhlasem redakce při zachování autorských práv a s uvedením pramene. Rukopisy nevyžádané redakcí se nevracejí. Za původnost a věcnou správnost ručí autor.
Vychází šestkrát ročně. Neprodejné. Členům klubu distribuováno zdarma. Nepravidelné přílohy na objednávku jsou kompenzovány zvláštním klubovým příspěvkem.
Rozsah čísla stran. Neprošlo jazykovou úpravou.

TISK SNV zak. č. 449/88

Do tisku předáno v VI/88
Vydávání schváleno OV Svazarmu Praha 4 a OŠK ONV Praha 4.
Evidenční číslo ÚVTEI 86 042.

ING. VÁCLAV FAJTA

Základy programování mikroprocesoru 6502

Recenze RNDr. Jiří Bok, CSc.
Technická redakce
Otilie Strnadová

Předmluva

Základy programování mikroprocesoru 6502 jsou určeny zájemcům o osmibitové počítače ATARI, kteří se chtějí naučit programovat v jazyku symbolických adres 6502. Obsah publikace a výběr příkladů je orientován na domácí mikropočítače ATARI 800XL, 65XE, 130XE, 800XE.

Publikace předpokládá základní znalosti z výpočetní techniky.

Kapitola 1. podává základní informace o hardware mikropočítače ATARI.

Kapitola 2. seznamuje s jazykem symbolických adres a druhy asemblerů.

Kapitola 3. uvádí pravidla pro zápis instrukcí JSA.

Kapitoly 4., 5. a 6. tvoří těžiště publikace. Seznamují nás s adresováním, instrukčním souborem mikroprocesoru 6502 a pseudoinstrukcemi, přičemž uvedená pravidla zápisu a jejich příklady jsou orientovány na překladač ATMAS-II. Tento překladač je podrobně popsán v kapitole 7.

Kapitola 8. seznamuje se zásadami programování.

Nadstavbou publikace je kapitola 9., která je určena pro vyspělejší programátory a seznamuje je se systémem přerušení.

V kapitole 10. je kompletní příklad, kde je ukázáno programování a především operování s překladačem ATMAS-II, připojení strojového kódu k Basicu.

V příloze jsou uvedeny tabulky, které obsahují základní informace o instrukcích.

Doplňkem této publikace je "Sbírka řešených úloh v jazyku symbolických adres mikroprocesoru 6502", která nabízí řešení řady praktických úloh.

Rád bych touto cestou poděkoval RNDr. J. Bokovi, CSc. za náměty a zapůjčení literatury.

Čtenářům, a především začátečníkům, přeji hodně trpělivosti při studiu a úspěšné zdolání všech úskalí.

Za případné připomínky a podněty předem děkuji.

Autor

Václav Fajta

Václav Fajta

V Hradci Králové 5. listopadu 1987

OBSAH

Použité zkratky a symboly

Registry:

X	index-registr X
Y	index-registr Y
A	akumulátor /střadač/
S	ukazatel vrcholu zásobníku /Stack/
PC	programový čítač /Program Counter/
PCH	vyšší /významnější/ byte programového čítače
PCL	nižší /méně významný/ byte programového čítače
P	příznakový registr

Příznaky:

N	příznak znaménka - Negative
V	příznak přetečení mantisy /b_7 xor b_6/ - Overflow
B	příznak programového přerušení - Break
D	příznak dekadického režimu - Decimal
I	příznak masky přerušení - Interrupt
Z	příznak nulovosti - Zero
C	příznak přenosu - Carry

Operátory:

+	součet
-	rozdíl
and	logický součin
or	logický součet
xor	exklusivní logický součet, nonekvivalence

TAB. 1 - LOGICKÉ OPERACE

B	A	B and A	B or A	B xor A	$\overline{A}$
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	1
1	1	1	1	0	0

Adresy:

EA	efektivní /skutečná/ adresa
M	efektivní /skutečná/ adresa
BA	bázová adresa
VA	vztažná /relativní/ adresa
PA	pomocná adresa
NA	návratová adresa

Symboly:

⟨ ⟩	obsah adresy nebo registru
( )	nepřímá adresace
⟶	přesun
↑	sledující registr nebo proměnná
#	symbol označující bezprostřední adresování
$	znak uvádějící čísla zapsaná v hexadecimálním tvaru
%	znak uvádějící čísla v binárním tvaru
opc	operační kód
opr	operand
op_8	osmibitový /1 byte/ operand instrukce
op_{16}	šestnáctibitový /2 byte/ operand instrukce
op_{16H}	významnější byte operandu op_{16}
op_{16L}	méně významný byte operandu op_{16}
Hi	High /vyšší, významnější část/
Lo	Low /nižší, méně významná část/
TOP	vrchol zásobníku
b_i	i-tý bit
SW	Software
HW	Hardware
JSA	jazyk symbolických adres
OS	operační systém
IRQ	Interrupt request /maskovatelné přerušení/
NMI	Non Maskable Interrupt /nemaskovatelné přerušení/
VBI	Vertical blank Interrupt /přerušení od vertikálního zatmění/

CIO	Central Input Output /hlavní rutina pro vstup a výstup/
V/V	Vstup/Výstup
DMA	Direct Memory Access /přímý přístup do paměti/
PIA	Petipheral Interface Adapter
GTIA	Graphics Television Interface Adapter
POKEY	Potenciometer Keyboard
ANTIC	Alpha Nureric Television Interface
CCNTL	Cartridge Control

1. Mikropočítač ATARI

Mikropočítače ATARI 800XL a 130XE obsahují:

- mikroprocesor MOSTEK 65C02,*/ nebo 6502 fy MOS Technology,
- osm paměťových obvodů 64 Kbit dynamické paměti RAM /ATARI 130XE má dvakrát tolik paměťových obvodů/,
- programovatelné obvody:
  - zákaznické obvody: ANTIC,
    GTIA,
    POKEY,
    FREDDIE,**/
  - komerční obvod PIA 6520
- sadu neprogramovatelných obvodů a pasivních prvků.

Jednotlivé obvody jsou vzájemně propojeny systémovou sběrnicí, která je složena z:

- adresové sběrnice /16-ti bitové/,
- datové sběrnice / 8-mi bitové/,
- řídicí sběrnice.

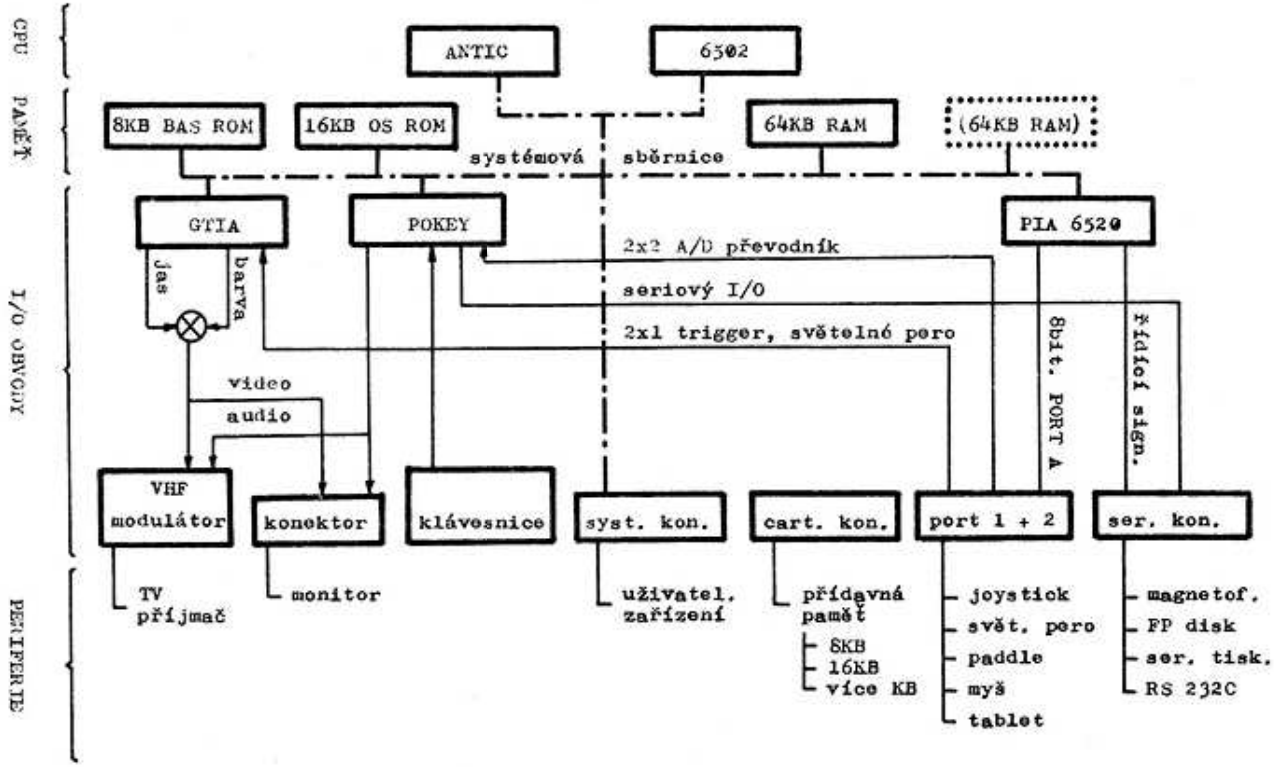
ATARI 800XL a 130XE jsou dvouprocesorové systémy, obsahují procesory 6502 a ANTIC. Oba procesory mají vlastní centrální procesorovou jednotku /CPU/ a také vlastní instrukční soubor. Vzájemně si půjčují systémovou sběrnici, přes kterou ovládají činnost počítače.

V této učebnici je popsán instrukční soubor mikroprocesoru 6502.

Pozn.: */ Pouze u novějších modelů

**/ Tento obvod je pouze u novějších modelů ATARI místo obvodu GTIA

viz Machulda, M.: Nová verze počítačů ATARI 800XL
Zpravodaj AK č.2, str.24, Praha, 1987



obr. 1 - Blokové schéma mikropočítače ATARI

1.1 Mikroprocesor 6502

Mikroprocesor se skládá ze tří částí:
- aritmeticko-logické jednotky,
- pole registrů,
- řadiče.

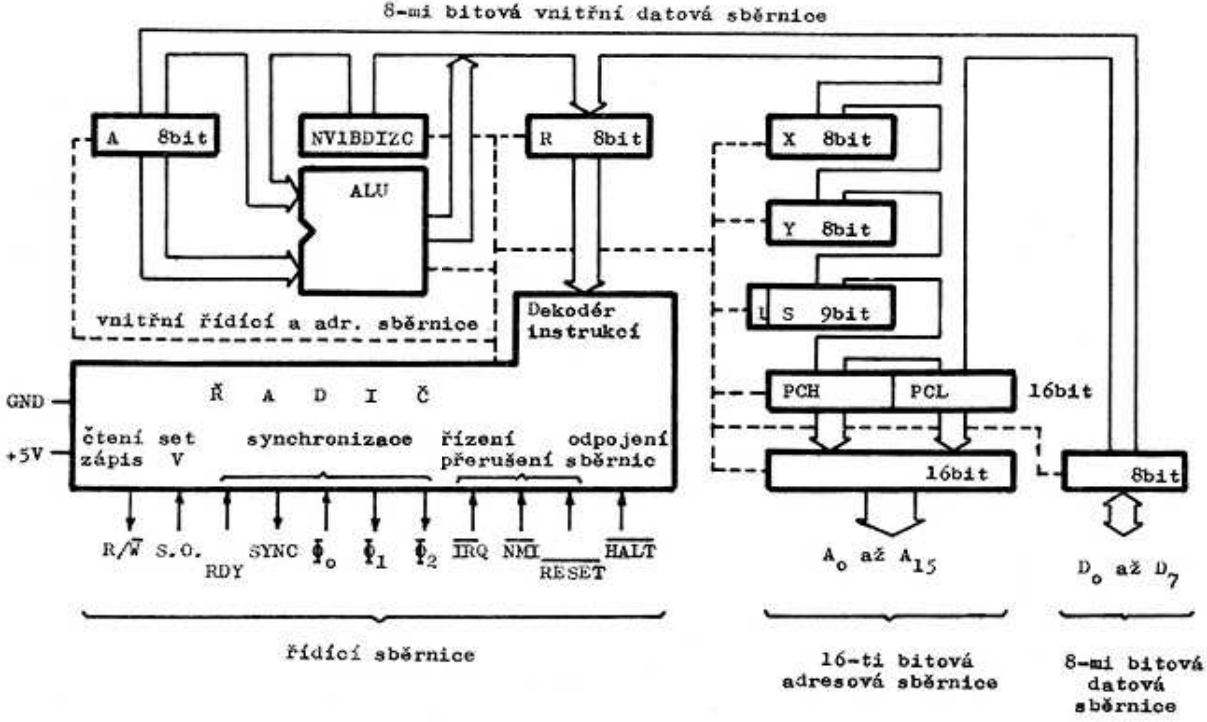
Aritmeticko-logická jednotka provádí matematicko-logické operace.

Pole registrů slouží k ukládání různých informací a dat.

Řadič řídí vnitřní činnost procesoru a přes řídicí sběrnici i činnost celého počítače.

Mikroprocesor MOSTEK 65C02 má oproti klasickému mikroprocesoru SY6502 /Synertek/ a R6502 /Rockwell/ navíc řídicí signál HALT, který umožňuje odpojení mikroprocesoru 6502 od systémové sběrnice a její předání druhému mikroprocesoru /ANTIC/.

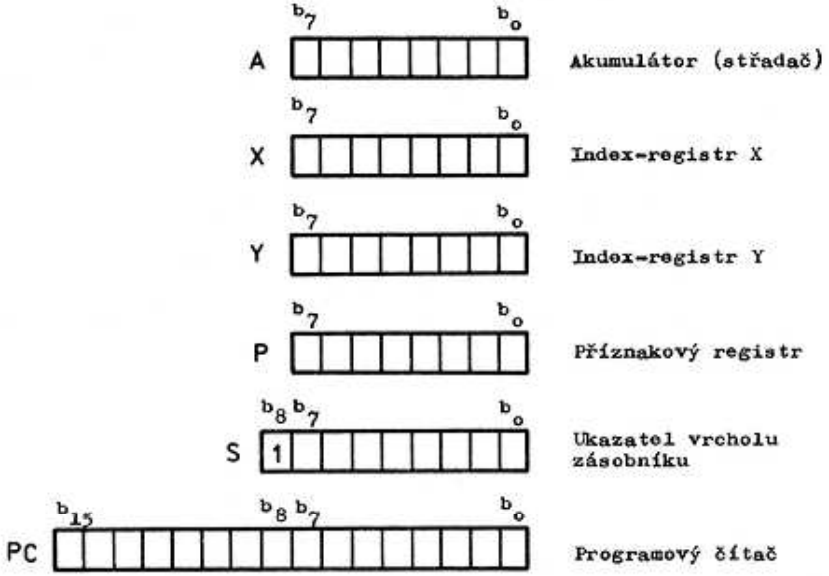
Mikroprocesor MOSTEK 65C02 pracuje s taktovací frekvencí 1.79 MHz. Má 40 vývodů. Oproti klasickému 6502 má o něco širší instrukční soubor.



obr. 2 - Blokové schéma mikroprocesoru 6502

1.2 Pole registrů

Registry jsou paměťová místa uvnitř mikroprocesoru, která jsou jím využívána pro různé operace. Pro programátora jsou přístupné: A, X, Y, P, S, PC.



obr. 3 - Pole registrů 6502

Akumulátor neboli střadač je osmibitový registr, který je nejvíce využíván. Velmi často se přes něj vstupuje do aritmeticko-logické jednotky, která do něj obvykle ukládá výsledky operací. Pomocí tohoto registru lze číst kterouko-li buňku paměti a zapisovat do buněk paměti RAM. Umožňuje také zapisovat informace do vnitřních registrů implementova-ných obvodů /ANTIC, GTIA, POKEY, PIA 6520/.

Index-registry X a Y jsou osmibitové registry podporu-jící indexové adresování. Rozsah jejich používání je oproti střadači omezen.

Registr S /Stack Pointer/ je devítibitový registr nazý-vaný ukazatel vrcholu zásobníku. Na nejvýznamnějším bitu je trvale log. 1.

Zásobník je část paměti, která je hardwarově umístěna na první stránce paměti /od adresy $100 do adresy $1FF včet-ně/. Slouží k přechodnému ukládání různých údajů /konstant, proměnných, mezivýsledků, parametrů, návratových adres atd./. Zásobník je typu LIFO /Last-In,First-Out/, což znamená, že informace, která do něho byla naposledy uložena, je vybírána jako první.

Zásobník se naplňuje směrem od vyšších adres k nižším. Jako první se tedy obsadí adresa $1FF, druhá $1FE atd. Na první volnou adresu zásobníku /tzv. vrchol zásobníku/ ukazuje vždy obsah registru S, proto je nazýván ukazatelem vrcholu zásobníku.

Při každém uložení dat do zásobníku se obsah registru S sníží o jedničku. Při výběru dat ze zásobníku se naopak o jedničku zvýší.

Registr PC /Program Counter/ je šestnáctibitový čítač adres neboli programový ukazatel. Obsahuje adresu právě prováděné instrukce.

Přeložený program ve strojovém kódu je uložen na urči-tém místě v paměti. Při běhu programu je v registru PC vždy uložena adresa, na níž leží v paměti právě prováděná in-strukce.

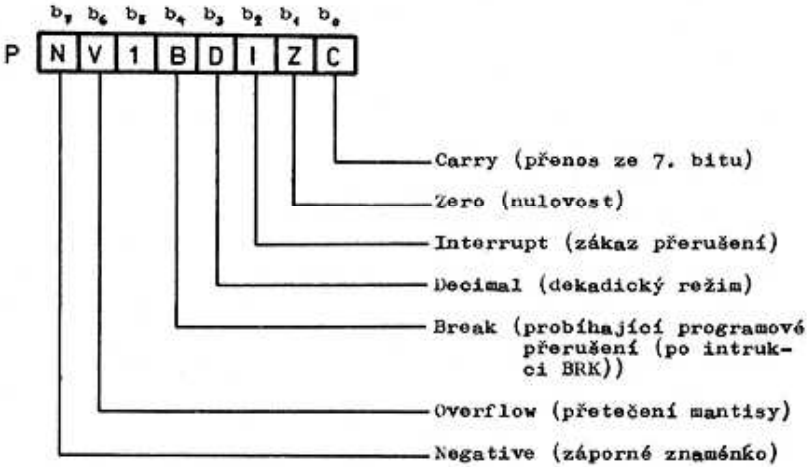
Po provedení jedno-, dvou-, tříbajtové instrukce se hod-nota PC registru zvýší o 1, 2, 3. Ukazuje pak na operační kód

následující instrukce. Výjimku tvoří provedení instrukcí skoku, při kterých je do PC uložena adresa operačního kódu instrukce, kterou má program pokračovat. Druhou výjimkou je zpracování přerušení. Do PC je v tomto případě uložena adresa první instrukce programu zpracovávajícího přerušení. Viz kapitola 9. Přerušení.

Registr P /Status registr/ je osmibitový stavový neboli příznakový registr. Využito je však pouze sedm bitů. Jednotlivé bity registru P nesou odlišné informace.

Příznakový registr je nastavován podle průběhu prováděné operace nebo podle jejího výsledku. Různé instrukce ovlivňují různé příznakové bity. Často nastavují pouze některé bity, ostatní pak zachovávají svoji původní hodnotu. Některé instrukce neovlivňují žádné příznakové bity.

Význam jednotlivých bitů registru P:



obr. 4 - Příznakové bity

Příznak C: Nastavuje se na log. 1, jestliže došlo k přetečení ze 7. bitu /b_7/. Například při součtu čísel, když výsledek přesáhne rozsah buňky /$FF, tj.255/. Nedošlo-li k přetečení, nastaví se na log. 0.

Příznak Z: Nastavuje se na log. 1, jestliže výsledek operace nebo přenášená data jsou nulové. Jsou-li nenulová, nastaví se na log. 0.

Příznak I: Nastavuje se na log. 1, jestliže došlo k zákazu zpracování přerušení /instrukcí SEI, HW přerušovacím signálem $\overline{IRQ}$ nebo přerušovacím signálem $\overline{NMI}$/.
Povolení zpracování přerušení /nastavení na log. 0/ se provádí instrukcí CLI.

Příznak D: Je-li nastaven na log. 1, indikuje tzv. decimální mód. Potom jsou matematické operace prováděny v BCD kódu /desítkové soustavě/. Tento režim se nastavuje instrukcí SED. Standardně je však příznak D nastaven na log. 0, což indikuje hexadecimální /binární/ mód. Je-li potřeba znovu nastavit standardní mód, použije se instrukce CLD.

Příznak B: Nastavuje se na log. 1, jestliže probíhá programové přerušení po instrukci BRK. Příznak B se nastaví automaticky na log. 0 při zpracování instrukce RTI.

bit b_6 : Nemá význam. Je trvale nastaven na log. 1.

Příznak V: Je nastaven instrukcí součtu ADC /resp. rozdílu SBC/, dojde-li k přetečení /podtečení/ z /do/ bitu b_6 a zároveň nedojde k přetečení /podtečení/ z /do/ bitu b_7, nebo dojde-li k přetečení /podtečení/ z /do/ bitu b_7 a zároveň nedojde k přetečení /podtečení/ z /do/ bitu b_6. Schématicky to lze vyjádřit takto:
pro ADC: V=/přetečení z b_6/ xor /přetečení z b_7/
pro SBC: V=/podtečení do b_6/ xor /podtečení do b_7/

Příznak N: Nastavuje se na log. 1, jestliže výsledek operace nebo přenášená data mají zápornou hodnotu /tj. nejvýznamnější bit b_7 má hodnotu log. 1/. Jsou-li da-

ta kladná /nejvýznamnější bit b_7 má hodnotu log. 0/, nastaví se na log. 0.

Příznaky lze rozdělit do dvou skupin:
a/ informační /B, D, I/,
b/ pro větvení programu /N, V, Z, C/.

Pro začínajícího programátora mají význam především příznaky Z, N, C /případně V/, které může využívat pro testování výsledků a dat a pro větvení programu.

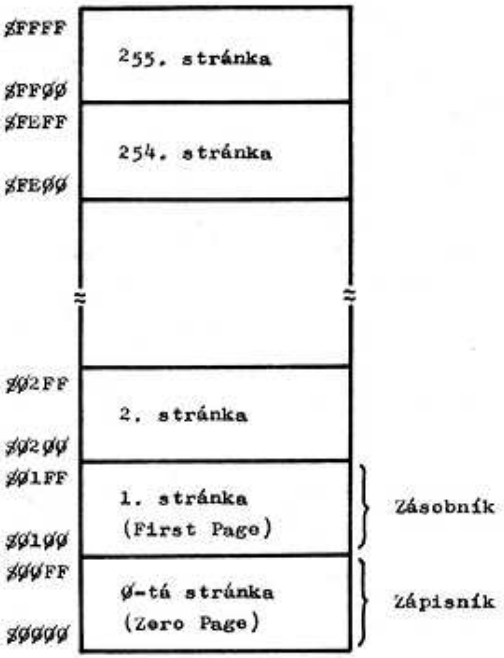
1.3 Paměť

Paměť počítače má dvě části: RAM a ROM. Do paměti typu ROM nelze zapisovat, slouží pouze pro čtení a je v ní uložen program operačního systému a program interpretu jazyka BASIC. Naproti tomu paměť typu RAM je přístupná pro čtení i zápis.

Nejmenší adresovatelnou jednotkou je byte /buňka, slabika/. Je složen z osmi bitů. Každý bit může nabývat hodnoty log. 0 nebo log. 1.

Vzhledem k tomu, že mikroprocesor 6502 má šestnáctibitovou adresovou sběrnici, lze přímo adresovat prostor od adresy $0000 do $FFFF včetně /tj. 0 až 65 535/, což představuje 64 KB.

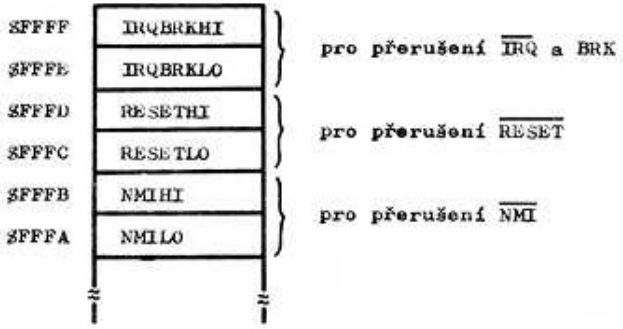
Paměť lze dělit na stránky po $100 /tj. 256/ byte. To znamená, že na adrese $00 až $FF leží nultá stránka paměti, na adrese $100 až $1FF leží první stránka paměti atd.



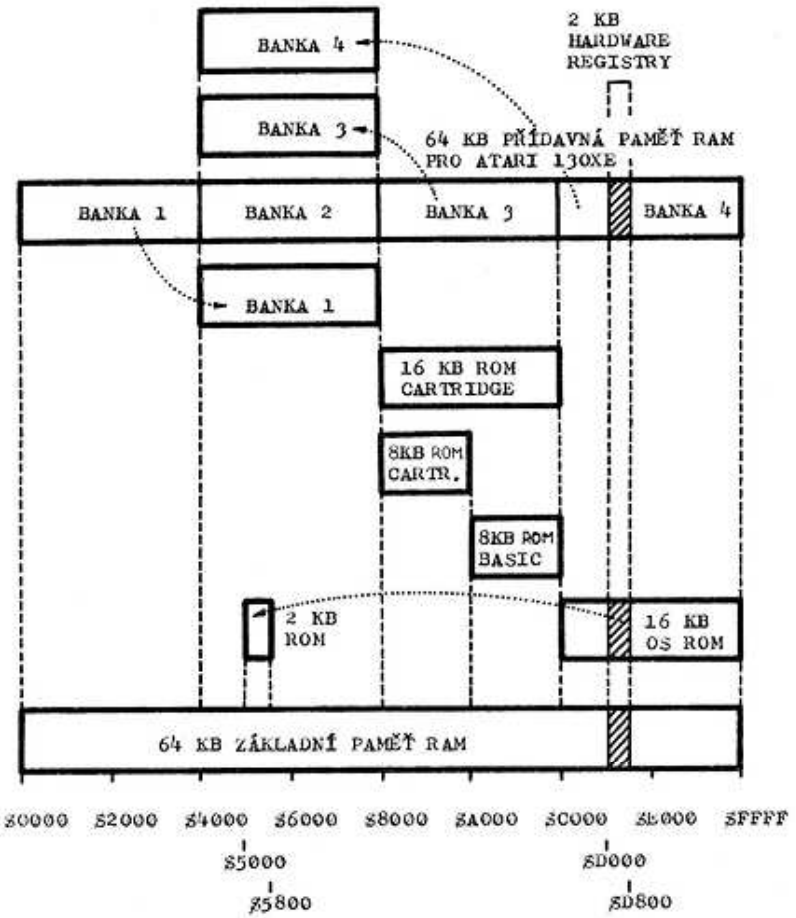
obr. 5 - Rozdělení paměti na stránky

Zvláštní význam má nultá a první stránka paměti. Nultá stránka se nazývá zápisníková paměť. Operování na ní je nejrychlejší, neboť je podporováno speciálními instrukcemi mikroprocesoru. První stránka paměti je tzv. zásobníková paměť /zásobník/ a je speciálně využívána mikroprocesorem.

Speciální význam pro mikroprocesoru 6502 má také posledních šest byte adresovatelného prostoru. Na těchto buňkách jsou uloženy efektivní adresy počátků programů zpracovávajících přerušení.



obr. 6 - Efektivní adresy přerušení



obr. 7 - Stínové využití 64 KB adresovatelného prostoru u ATARI 800XL resp. ATARI 130XE

2. Co je JSA a co asembler?

Počítač /resp. CPU/ je schopen porozumět pouze programům ve strojovém kódu, což je strojový jazyk představovaný různými čísly s jednoznačným významem. Strojový jazyk je však pro člověka těžko zapamatovatelný a nic neříkající. Proto vznikly vyšší jazyky, které nahrazují čísla slovy nebo zkratkami a nabízejí programátorovi často ještě další komfort pro jeho práci.

Každý program napsaný ve vyšším jazyku musí být přeložen do strojového kódu, což se provádí buď při běhu programu nebo ještě před jeho spuštěním.

Jazyk symbolických adres /někdy přeneseně nazývaný asembler/ je po strojovém kódu druhý nejnižší jazyk, má k němu tedy nejblíže. Ke každé strojové instrukci je přiřazena instrukce v JSA vyjádřená zkratkami z anglických slov, která charakterizují činnost instrukce. Také důležité konstanty, adresy a návěští lze nahradit zkratkami. Symbolicky vyjádřená instrukce v JSA je pak snadněji zapamatovatelná než suché, nic neříkající číslo strojového kódu.

Například instrukce TAX znamená v JSA "Transfer from Accumulator to Register X", což lze přeložit jako přesun obsahu akumulátoru do registru X. Zkratka TAX je jistě srozumitelnější a zapamatovatelnější než její ekvivalent A3 ve strojovém kódu.

Program napsaný v JSA se nazývá zdrojový text. Do strojového kódu se překládá pomocí programu nazývaného asembler. Asembler je tedy překladač jazyka symbolických adres do strojového kódu.

obr. 8 - Princip překladu

2.1 Druhy asemblerů

Existuje mnoho druhů překladačů jazyka symbolických adres, které se liší svojí kvalitou a možnostmi.

Překladače lze dělit z různých hledisek:

Podle místa překladu:

1/ vlastní asembler /self-assembler, resident-assembler/
- překlad vykonává přímo počítač /např., ATMAS-II/,

2/ nevlastní asembler /křížový, cross-assembler/
- překlad se provádí na jiném, zpravidla výkonnějším počítači

Podle editoru:

1/ Jednořádkový - překládá každý řádek zvlášť ihned po jeho vložení do počítače. Neumožňuje provádět odkazy a používat symbolická návěští, zpravidla má velmi omezený soubor pseudoinstrukcí. Pro programy běžné úrovně je prakticky nepoužitelný.

2/ Editorový
Pomocí editoru se připraví celý zdrojový text programu a ten se pak celý najednou přeloží. Zdrojový text musí respektovat syntax použitého překladače.

Kvalitnější překladače mají v sobě implementovaný editor. Editorový překladač má podstatně vyšší možnosti než jednořádkový.

Podle počtu průchodů:

1/ Jednoprůchodový
Překladač čte zdrojový text pouze jedenkrát a rovnou generuje cílový strojový kód. Tím je dáno omezení jednoprůchodového překladače. Neumožňuje provádět dopředné odkazy a lze jej použít pouze pro jednodušší úlohy.

2/ Dvouprůchodový
Čte zdrojový text dvakrát. Při prvním průchodu vyhledává a třídí symboly použité v programu, přiřazuje skutečné adresy návěštím, zabývá se přemísťujícími informacemi, globálními proměnnými a pod. Ve druhém průchodu se uskutečňuje vlastní překlad.

3/ Trojprůchodový
V prvních dvou krocích provádí podobnou činnost jako dvouprůchodový překladač. Ve třetím kroku se zpravidla generuje výpis překladu /listing/.

Podle typu cílového souboru:

1/ Překladače generující absolutní spustitelný tvar /com-file/, který lze kdykoli zavést na určitou adresu a spustit.

2/ Překladače generující přemístitelný tvar /relocatable object code/, který se před vlastním spuštěním musí zavádět speciálním zavaděčem nebo spojovat linkerem, který vytváří com-file.

Podle možností zpracování makroinstrukcí:

1/ Asembler neumožňující zpracovat makroinstrukce.

2/ Makroasembler
Umožňuje zpracovat makroinstrukce a představuje vyšší úroveň překladače.

Jednotlivé druhy asembleru se liší:

- v úrovni programování, kterou umožňují,
- v komfortu používání,
- ve způsobu operování,
- v počtu a kvalitě **pseudoinstrukcí**,
- v možnosti použití makroinstrukcí,
- v možnosti použití symbolů, matematických výrazů aped.,
- v syntaxi,
- ve vlastní práci překladače /rozsáhlost programu, rychlost překladu, přidělování paměti, obsažnost listingu a chybových hlášení/,
- v návaznosti a slučitelnosti s dalším programovým vybavením, jako je:
  - debugger /program pro ladění/,
  - tracer /program pro sledování chodu programu/,
  - monitor /program pro statické sledování programu a dat/,
  - disassembler/program pro převod strojového kódu do jazyka blízkého JSA/,
  - editor /program pro pořizování a opravování zdrojového textu/,
  - linker /zaváděcí a spojovací program/,
- atd.

Vzhledem k tomu, že pravidla programování v JSA jsou závislá na používaném překladači, nelze je popsat zcela obecně. Pro popis jazyka symbolických adres jsme vybrali překladač ATMAS-II. Tento překladač je mezi členy klubu velmi rozšířen, existuje v kazetové i disketové verzi /i na cartridge/, je rychlý, komfortní a práce s ním je jednoduchá.

2.2 Přehled asemblerů pro ATARI

Podle literatury /4/ byly pro mikropočítače ATARI vyvinuty následující překladače jazyka symbolických adres 6502 a jejich podpůrné programové vybavení:

1. ATMASD - Macro Assembler/Editor
   ELCOMP PUBLISHING, Inc., 53 Redrock Lane, Pomona, CA 91766
2. ATAS-I - Assembler/Editor, ATMONA-1 Monitor , ATMONA-2 (Debugger)
   ELCOMP PUBLISHING, Inc., 53 Redrock Lane, Pomona, CA 91766
3. ATMAS-II - Macro Assembler/Editor/Monitor
   ELCOMP PUBLISHING, Inc., 53 Redrock Lane, Pomona, CA 91766
4. The Assembler/Editor Cartridge
   ATARI, Inc.
5. AMAC (Atari Macro Assembler), Medit (Editor), DDT (Debugger) APX, ATARI, Inc.
6. MAC/65 - Macro Assembler, BUG-65 Debugger
   Optimized System Software, Inc., 10379, Lansdale Avenue, Cupertins, California 95014
7. SYNASSEMBLER
   Synapse Software, Inc., Jacuzzi, Suite 1, Richmond, California 94804
8. MAE (Macro Assembler/Text Editor)
   Eastern House Software, Inc., 3239 Linda Drive, Winston--Salem, North California 27106
9. EDIT 6502
   LJK Enterprises, P.O. Box 10827, St. Louis, Missouri 63129

	ATMAS-II ATAS	AMAC	ASSEMBLER /EDITOR	MAC-65	SYNASSEMBLER	MAE	EDIT 6502
začátek programu	ORG	ORG	*=	*=	.OR	.BA	ORG
definice byte	DFB	DB	.BYTE		.BS	.BYTE	DFB
definice slova	DFW	DW	.WORD		.DA		DFW
přiřazeni	EQU	EQU	=		.EQ		EQU
nulostránkové přiřazeni	EPZ						
ukončeni překladu	(CTRL-Z)		.END		.EN	.END	END
ASCII řetězec	ASC		.BYTE"..."		.AS		

TAB. 2 Tabulka odlišnosti v zápisu ekvivalentnich pseudoinstrukci pro různé překladače JSA pro ATARI

	ATMAS-II ATAS	AMAC	ASSEMBLER /EDITOR	MAC-65	SYNASSEMBLER	MAE	EDIT 6502
Píší se čísla řádků?	ne	-	ano	ano	ano	ano	ne
Příklad zápisu nižšího byte operandu	LDA#LOOP LDA#LOOP:L	LOW	LDA#LOOP&255		LDA#LOOP	#L	
Příklad zápisu vyššího byte operandu	LDA#LOOP/256 LDA#LOOP:H	HIGH	LDA#LOOP/256		LDA /LOOP	#H	
Umožňuje aritmetiku mezi symboly operandu?	ano + - / *( )	ano + - / *	ano + - / *	ano + - / *	ano + -	ano + -	ano + - / *
Maximální počet znaků v symbolu návěstí	8	6	6	127	32	31	

TAB. 3 Tabulka odlišností v syntaxi JSA pro překladače pro ATARI

2.3 Basic nebo asembler?

Basic je interpret, překlad se provádí vždy až při běhu programu. Překládají se jednotlivé instrukce samostatně. Překlad programu napsaného v Basicu se tedy provádí automaticky při každém jeho spuštění.

Asembler je překladač typu kompilátor. Překlad zdrojového textu se provede pouze jednou a pak už se spouští pouze přeložený program ve strojovém kódu.

Rychlost běhu programu ve strojovém kódu je průměrně 100 až 1000 krát vyšší, než u srovnatelného programu v ATARI Basicu. Nelze zanedbat ani úsporu paměti.

Basic patří k vyšším jazykům než asembler, má tedy komfortnější příkazy a pro řadu aplikací se hodí lépe. Naproti tomu asembler má instrukce pouze pro základní úkony, často je třeba pro stejnou věc, kterou lze zapsat v Basicu jednou instrukcí, sestavit několikařádkovou sekvenci příkazů. Může však maximálně využívat hardware i software počítače, čímž umožňuje řešit mnohem více problémů než Basic.

Volbu jazyka je třeba důkladně uvážit podle povahy řešené úlohy.

3. Pravidla zápisu instrukcí v JSA

Stejně jako jiné jazyky, má i JSA svá pravidla, která je nutno při zápisu zdrojového textu dodržovat. Pravidla se do určité míry liší podle použitého překladače. Zde jsou popsána pravidla pro ATMAS-II.

Na rozdíl od Basicu lze v JSA zapsat na jeden řádek pouze jednu instrukci /příp. pseudoinstrukci či makroinstrukci/. Každý řádek musí být ukončen RETURNem.

Rozlišujeme čtyři typy zdrojových řádků:

1/ instrukce,

2/ pseudoinstrukce,

3/ makroinstrukce,

4/ celořádkový komentář.

Zdrojový řádek instrukcí, pseudoinstrukcí a makroinstrukcí má čtyři části:

1/ pole návěští,

2/ pole typu operace,

3/ pole operandu,

4/ pole komentáře.

Jednotlivé části musí být seřazeny v uvedeném pořadí a odděleny alespoň jedním oddělovačem /SPACE nebo TAB/.

Pole návěští

Návěští je většinou nepovinné. Vyžadují ho pouze pseudoinstrukce EQU a EPZ a definice makroinstrukce. U definice makroinstrukce se v poli návěští uvádí jméno definované makroinstrukce.

Návěští slouží k odkazům na určité místo v programu. Nelze uvést stejné návěští vícekrát. Výjimkou jsou lokální návěští v makroinstrukcích, kde v těle makroinstrukce se může vyskytovat stejné lokální návěští jako v jiné makroinstrukci.

Pro ATMAS-II se návěští může skládat z jednoho až osmi alfanumerických znaků, kde první znak musí být písmeno. Neodděluje se dvojtečkou.

Nemá-li zdrojový řádek návěští, musí se před typem operace uvést alespoň jeden oddělovač. Zpravidla se používá TAB.

Pole typu operace

V tomto poli se může vyskytovat:

1/ instrukce ze souboru JSA,

2/ pseudoinstrukce,

3/ jméno volané makroinstrukce.

Pole operandu

Zde se uvádí:

1/ operand instrukce, pokud jej instrukce vyžaduje,

2/ seznam formálních parametrů v definici makroinstrukce,

3/ seznam skutečných parametrů při volání makroinstrukce.

Pole komentáře

V poli komentáře mohou být libovolné znaky. Slouží pro poznámky a vysvětlivky, do strojového kódu se nepřekládá. Pro ATMAS-II nemusí být oddělen středníkem. Jako oddělovače lze použít kromě znaků SPACE a TAB i další znaky /např. tečku, středník a pod./.

Celořádkový komentář se v první pozici uvádí hvězdičkou. Do strojového kódu se nepřekládá. Je to obdoba příkazu REM v Basicu.

Příklad zdrojového textu:

```
          pole    pole
pole      typu    ope-    pole
návěští   ope-    randu   komentáře
          race

*  PRIKLAD ZDROJOVEHO TEXTU

          ORG     $5000   poč. adr. stroj. pgmu
          LDX     #$A0    $A0 ⟶ ⟨X⟩
SKOK      DEX             ⟨X⟩ - 1 ⟶ ⟨X⟩
          BNE     SKOK    jestliže  ⟨X⟩  ≠ 0, skok
```

TAB. 4 Formální zápis zdrojového textu v JSA

3.1 Operandy instrukcí JSA

Operandy instrukcí jazyka symbolických adres mohou být jednobajtové /op_8/ nebo dvoubajtové /op_{16}/. Některé instrukce jazyka symbolických adres operand nemají.

Způsob zápisu operandů závisí na používaném překladači. Uvádíme zde pravidla pro ATMAS-II.

Jednobajtovým operandem může být:

a/ Decimální číslo v rozsahu 0 až 255.
/například 254/

b/ Hexadecimální číslo v rozsahu $00 až $FF. Hexadecimální čísla musejí být uvedena znakem $.
/například $FE/

c/ Binární číslo v rozsahu %00000000 až %11111111. Binární čísla je nutno uvádět znakem %.
/například %11111110/

d/ ASCII znak uvedený jedním apostrofem.
/například 'A/

e/ Symbol reprezentující osmibitovou hodnotu a deklarovaný pseudoinstrukcí EPZ.
/příklad: Jako operand lze použít symbol ALFA, jestliže jsme ho deklarovali například takto:
ALFA EPZ $FE

f/ Symbol reprezentující šestnáctibitovou hodnotu, v úvahu se pak bere pouze méně významný byte hodnoty symbolu. Pozor! Nelze používat ve všech případech.
/příklad: Symbol BETA je deklarován takto:
BETA EQU $20A8. Zapíšeme-li symbol BETA na místo jednobajtového operandu, bude se brát v úvahu pouze hodnota $A8./

g/ Šestnáctibitový symbol omezený znakem :L, v úvahu se pak bere pouze méně významný byte hodnoty symbolu. Výsledek je stejný jako v předcházejícím případě, ale tento způsob zápisu je možný ve všech případech.

/příklad: Symbol GAMA je deklarován takto:
GAMA EQU $3F66. Zapíšeme-li na místo jednobajtového operandu GAMA:L, bude se brát v úvahu hodnota $66. Je to totéž, jako když zapíšeme pouze GAMA bez omezení, což však, jak jsme uvedli, není v některých případech použitelné./

h/ Šestnáctibitový symbol omezený znakem :H, v úvahu se pak bere významnější byte hodnoty symbolu.
/příklad: Symbol DELTA je deklarován takto:
DELTA EQU $22AA. Zapíšeme-l í na místo jednobajtového operandu DELTA:H, bude v úvahu vzata hodnota $22. Je to totéž, jako bychom zapsali DELTA/256./

i/ Aritmetický výraz s operátory + - * /, jehož hodnota musí být celé číslo v rozsahu 0 až 256 /tj. $00 až $FF/.
/příklad: (K+3)/$10)

Dvoubajtovým operandem může být:

a/ Decimální číslo v rozsahu 0 až 65535.
/příklad: 32768/

b/ Hexadecimální číslo v rozsahu $0000 až $FFFF. Musí být uvedeno znakem $.
/příklad: $8000/

c/ Binární číslo v rozsahu %0000000000000000 až %1111111111111111. Musí být uvedeno znakem %.
/příklad: %1000000000000000/

d/ Symbol reprezentující šestnáctibitovou hodnotu a deklarovaný pomocí pseudoinstrukce EQU nebo použitý jako návěští.
/příklad: ALFA/

e/ Symbol reprezentující osmibitovou hodnotu. Významnější byte bude doplněn nulami.
/příklad: Symbol PRUM je deklarován takto:
PRUM EPZ $AC. Zapíšeme-li ho na místo dvoubajtového operandu, bude jeho hodnota $00AC./

To platí pouze pro instrukce, kde je očekáván dvoubajtový operand.

f/ Znak *, který znamená aktuální stav čítače adres PC.

g/ Aritmetický výraz s operátory + - * / , jehož hodnota musí být celé číslo v rozsahu $0000 až $FFFF /tj. 0 až 65535/. /příklad: * + 3 znamená aktuální stav čítače adres + 3./

4. Adresování

Adresováním se rozumí určení místa v paměti. Buňku paměti lze adresovat mnoha způsoby. U mikroprocesoru 6502 existuje 13 způsobů adresování.

Zkratka	Anglický název	Český překlad
IMPL	Implied	zahrnuto v sobě /implicitní/
ACC	Accumulator	přímé adresování
IMM	Immediate	bezprostřední adresování
ABS	Absolute	absolutní adresování
ABS,X	Absolute X-indexed	absolutní adresování přes index-registr X
ABS,Y	Absolute Y-indexed	absolutní adresování přes index-registr Y
ZP	Zero Page	nulostránkové absolutní adresování
ZP,X	Zero Page X-indexed	nulostránkové adresování přes index-registr X
ZP,Y	Zero Page Y-indexed	nulostránkové adresování přes index-registr Y
REL	Relative	relativní adresování
(IND)	Indirect	nepřímé adresování
(IND,X)	X-indexed Indirect	nepřímé adresování přes index-registr X
(IND),Y	Indirect Y-indexed	nepřímé adresování přes index-registr Y

TAB. 5 Přehled způsobů adresování

Některé instrukce mohou využívat více způsobů adresování, některé pouze jeden.

Jednotlivé způsoby adresování se liší postupem určení efektivní /skutečné/ adresy dat. Efektivní adresou se rozumí adresa, na které jsou data skutečně uložena nebo kam se mají uložit po provedení instrukce.

Jednotlivé způsoby adresování se také liší formálním zápisem operandu. Právě podle způsobu zápisu operandu pozná překladač, které adresování je použito a jak tedy bude určovat efektivní adresu dat.

Instrukce, které mají více způsobů adresování, mají také více operačních kódů. Při překladu zdrojového textu programu do strojového kódu je instrukci přidělen operační kód podle použitého způsobu adresování /tedy podle způsobu zápisu operandu/. A naopak to znamená, že operační kód si v sobě vždy nese informaci o způsobu adresování a tím o určení efektivní adresy dat.

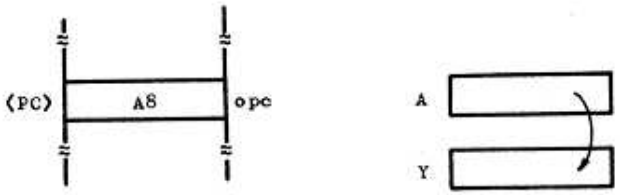
Přehled přípustných způsobů adresování pro jednotlivé instrukce, způsob zápisu operandu a příslušné operační kódy jsou uvedeny v tabulce v příloze 1.

Adresování je důležitou kapitolou učebnice, jeho pochopení je předpokladem pro správné pochopení instrukcí. Je proto nutné věnovat této kapitole náležitou pozornost.

4.1 Adresování IMPL - Implied - zahrnuto v sobě

Instrukce používající tento způsob adresování si nesou všechny nutné informace samy v sobě. Proto také nemají žádný operand, sama instrukce bez operandu vyjadřuje co a jak dělat. Není možno u nich použít žádný jiný způsob adresování.

Příklad:
Instrukce TAY přesouvá obsah akumulátoru do index-registru Y. a má operační kód $A8. Objeví-li se tedy v programu přeložené do strojového kódu operační kód $A8, systém ví, že má být přenesen obsah akumulátoru do registru X, ačkoli instrukce neměla uveden žádný operand.

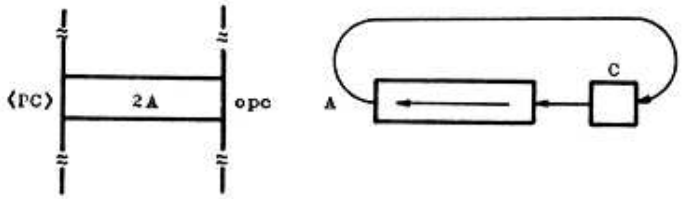


obr. 9 - Adresování IMPL

4.2 Adresování ACC - Accumulator - přímé adresování akumulátoru

Využívá-li instrukce adresování ACC, pak pracuje pouze s akumulátorem. To znamená, že zdrojem dat je akumulátor a výsledek operace je uložen zpět do akumulátoru. Tyto instrukce mají na místě operandu uveden střadač A. Po překladu do strojového kódu jsou jednobajtové.

Příklad:
Instrukce ROL A /operační kód $2A/ provádí tzv. "levou rotaci" obsahu střadače a ovlivňuje příznak C. Výsledek rotace bude uložen zpět do střadače.



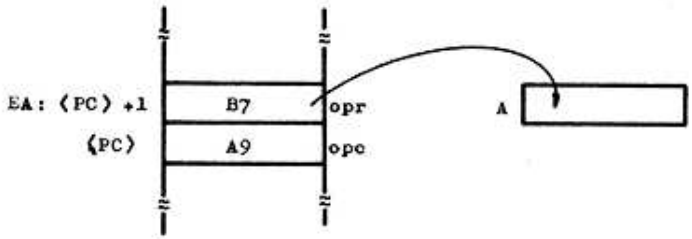
obr. 10 - Adresování ACC

4.3 Adresování IMM - Immediate - bezprostřední adresování

Adresování IMM umožňuje provádět operace s konstantami. Před operandem musí být uveden znak#, který způsobí, že znaky za ním uvedené jsou chápány jako konstanta, nikoli jako adresa. Používá se pro vkládání dat.

Instrukce používající toto adresování jsou dvoubajtové, po překladu do strojového kódu je tedy instrukce tvořena dvěma byte. První obsahuje operační kód, druhý uvedenou konstantu /data/. Uložení dat bezprostředně za operačním kódem je pro tento způsob adresování charakteristické.

Příklad:
Instrukce LDA#$87 provádí přesun konstanty $87 do akumulátoru. Po překladu do strojového kódu bude tvořena dvěma byte. V prvním bude operační kód $A9 , ve druhém konstanta $87. Po provedení instrukce bude ve střadači uložena hodnota $87.



obr. 11 - Adresování IMM

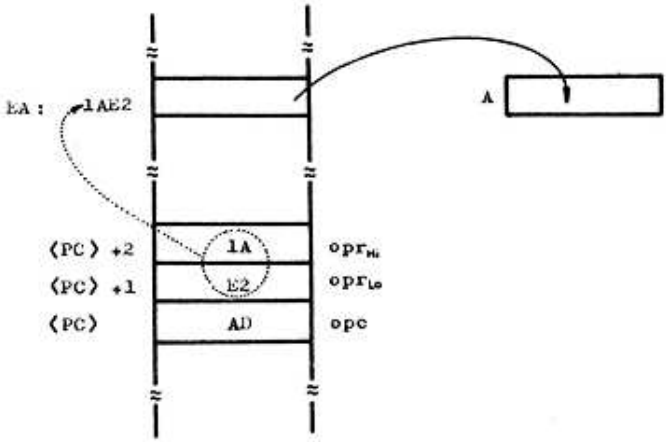
4.4 Adresování ABS - Absolute - absolutní adresování

Využívá-li instrukce adresování ABS, pak má na místě operandu uvedenu dvoubajtovou absolutní adresu. Na této adrese se nacházejí data, se kterými má být operace provedena. Takže zatímco u adresování IMM jsou v instrukci uvedena přímo data, u adresování ABS je v instrukci uvedena adresa, která určuje, kde data leží.

Instrukce používající toto adresování jsou po překladu do strojového kódu tříbajtové. První byte obsahuje operační kód, další dva adresu uložení dat. Přičemž platí zásada: na nižší adrese je uložen méně významný byte adresy dat, na vyšší adrese je významnější byte adresy.

Protože operand, který vyjadřuje skutečnou /efektivní/ adresu dat, je dvoubajtový, lze adresovat kteroukoli buňku 64 KB paměti. Operand může nabývat hodnot od $0000 až po $FFFF /tj. 0 až 65 535/.

Příklad:
Instrukce LDA $1AE2 provádí přesun dat uložených na adrese $1AE2 do akumulátoru. Po překladu do strojového kódu:



obr. 12 - Adresování ABS

4.5 Adresování ABS,X - Absolute X-indexed - absolutní adresování přes index-registr X

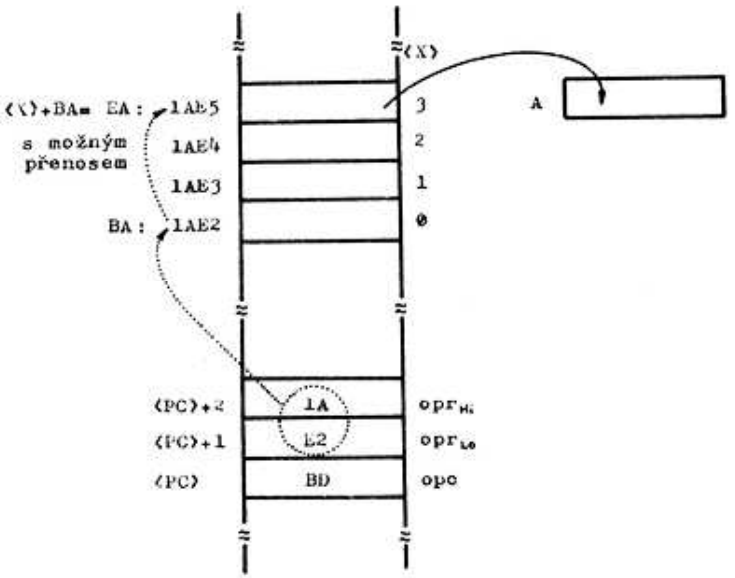
Využívá-li instrukce adresování ABS,X, pak na místě operandu je uvedena bázová adresa a registr X. Efektivní adresa dat je určena součtem bázové adresy s obsahem index-registru X.

Instrukce jsou po překladu do strojového kódu tříbajtové. První byte obsahuje operační kód, další dva byte bázovou adresu /na nižší adrese méně významný byte adresy, na vyšší významnější byte adresy/.

Zatímco u adresování ABS obsahoval operand přímo efektivní adresu dat, u adresování ABS,X je efektivní adresa určena až po součtu uvedené bázové adresy s obsahem index-registru X.

Jelikož index-registr X může obsahovat hodnoty od $00 až do $FF /tj. 0 až 256/, může efektivní adresa ležet na stejné stránce paměti jako bázová adresa nebo o stránku výše.

Příklad:
Předpokládejme, že index-registr X obsahuje hodnotu $03. Instrukce LDA $1AE2,X potom přesune do akumulátoru data uložená na adrese $1AE2+$03, tj. adresa $1AE5.



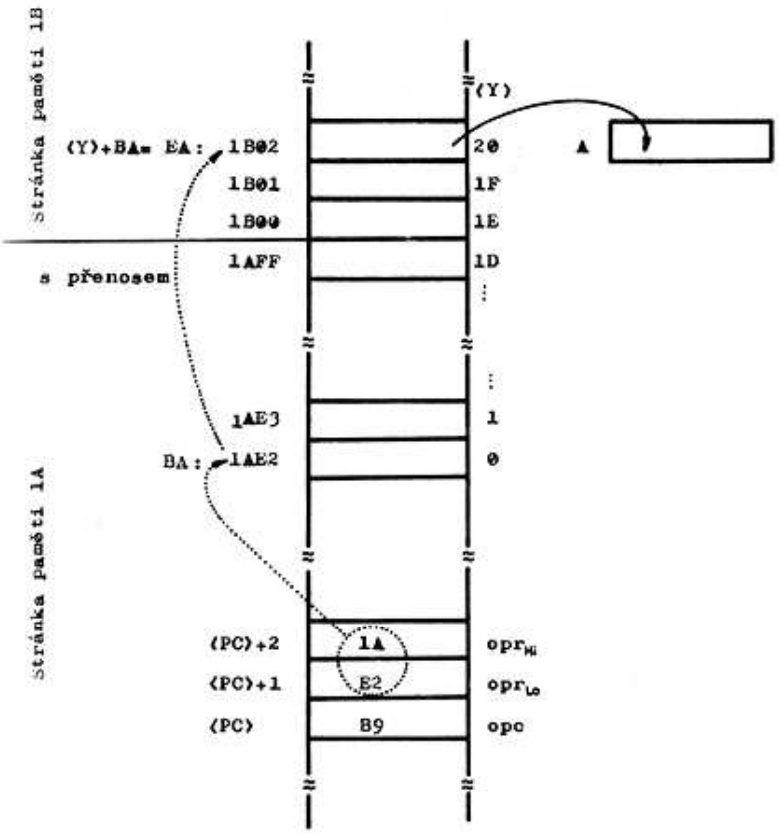
obr. 13 - Adresování ABS,X

4.6 Adresování ABS,Y - Absolute Y-indexed - absolutní adresování přes index-registr Y

Adresování ABS,Y je shodné s adresováním ABS,X, liší se pouze používaným index-registrem. Operand obsahuje bázovou adresu a index-registr Y. Efektivní adresa se určí součtem bázové adresy s obsahem index-registru Y a může ležet na stejné stránce jako bázová adresa nebo o stránku výše.

Instrukce jsou po překladu do strojového kódu tříbajtové. První byte obsahuje operační kód, další dva byte bázovou adresu.

Příklad:
Předpokládejme, že index-registr Y obsahuje hodnotu $20. Instrukce LDA $1AE2,Y přesune do akumulátoru data uložená na adrese $1AE2+$20, tj. adresa $1B02. Efektivní adresa tedy leží o stránku výše než bázová adresa.



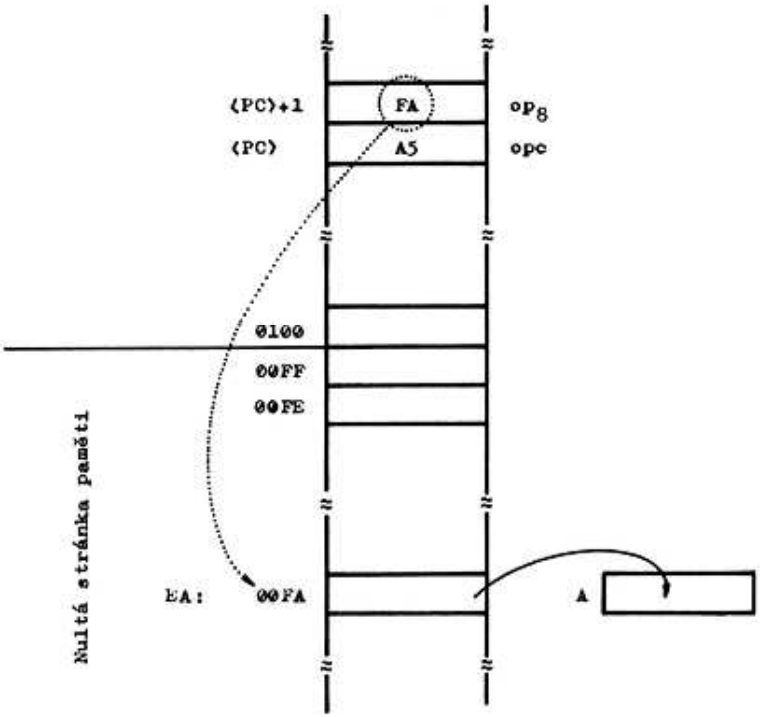
obr. 14 - Adresování ABS,Y

4.7 Adresování ZP - Zero Page - nulostránkové absolutní adresování

Adresování ZP je velmi podobné absolutnímu adresování ABS. Operand také uvádí absolutní adresu dat, v tomto případě však jeho hodnota může být pouze v rozmezí $00 až $FF. Adresuje tedy pouze nultou stránku paměti.

Instrukce jsou po překladu do strojového kódu dvoubajtové. První byte obsahuje operační kód, druhý jednobajtovou absolutní adresu dat.

Příklad:
Instrukce LDA $FA provádí přesun dat z adresy $FA do akumulátoru.



obr. 15 - Adresování ZP

Pozn.: Pozor na rozdíl mezi zápisy LDA $00FA /adresování typu ABS/ a LDA $FA /adresování typu ZP/. Výsledek operace bude sice stejný, ale rozdíl je v počtu byte, které instrukce zabere, a v času provedení instrukce.

4.8 Adresování ZP,X - Zero Page X-indexed - nulostránkové adresování přes index-registr X

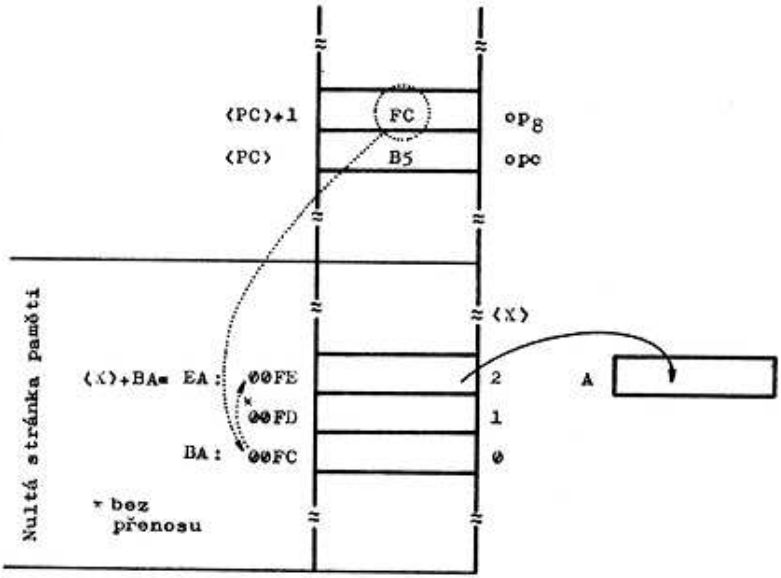
Využívá-li instrukce toto adresování, pak má na místě operandu uvedenu bázovou adresu a index-registr X. Bázová adresa ukazuje na nultou stránku paměti, je tedy v rozmezí $00 až $FF. Efektivní adresa je určena součtem bázové adresy s obsahem index-registru X, při němž však dochází k zanedbání případného přechodu na vyšší stránku paměti. Efektivní adresa tedy vždy leží na nulté stránce paměti.

Zatímco u adresování ZP obsahoval operand přímo efektivní adresu dat, je u adresování ZP,X efektivní adresa určena přes obsah index-registru X.

Adresování ZP,X je velmi podobné adresování ABS,X s tím rozdílem, že lze adresovat pouze nultou stránku paměti.

Instrukce jsou po překladu do strojového kódu dvoubajtové. První byte obsahuje operační kód, druhý bázovou adresu.

Příklad:
Předpokládejme, že obsah index-registru X je $02. Instrukce LDA $FC,X pak přesune do akumulátoru data z adresy $FC+$02, což je adresa $FE.



obr. 16 - Adresování ZP,X

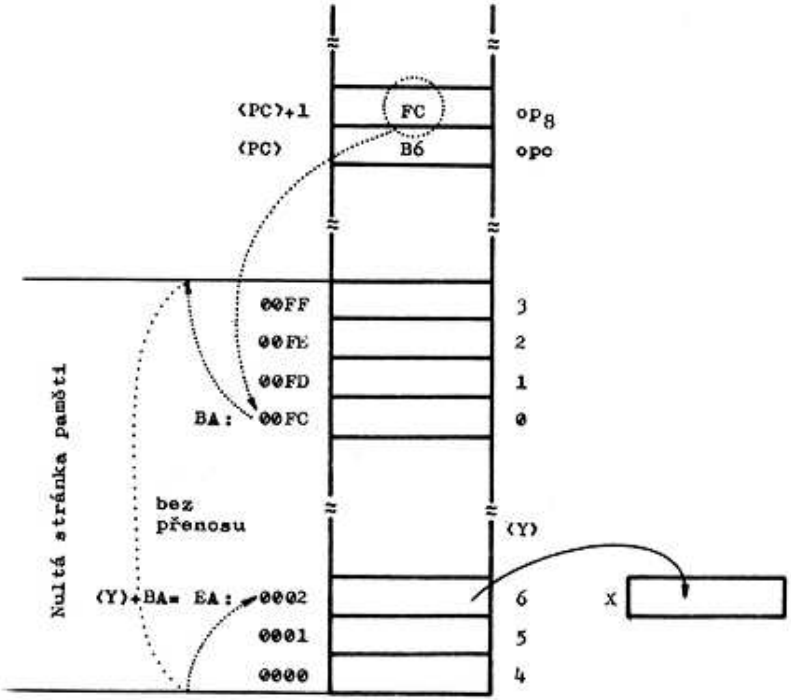
4.9 Adresování ZP,Y - Zero Page Y-indexed - nulostránkové adresování přes index-registr Y

Je velmi podobné adresování ZP,X. Liší se pouze v používaném index-registru. Operand instrukce obsahuje bázovou adresu a index-registr Y. Bázová adresa je v rozmezí $00 až $FF /nultá stránka paměti/. Efektivní adresa je určena součtem

bázové adresy s obsahem index-registru Y, při němž se případný přechod na vyšší stránku zanedbává. Adresovat lze tedy pouze nultou stránku paměti. Tím se liší od adresování ABS,Y.

Instrukce jsou po překladu do strojového kódu dvoubajtové. První byte obsahuje operační kód, druhý bázovou adresu.

Příklad:
Instrukce LDX provádí přesun dat z efektivní adresy do registru X. Předpokládejme, že v index-registru Y je uložena hodnota $06. Instrukce LDX $FC,Y přesune do registru X obsah adresy vypočtené ze součtu $FC+$06 se zanedbáním přechodu na vyšší stránku paměti. Součet bez zanedbání by byl $102, po zanedbání přechodu je tedy efektivní adresa přenášených dat $02.

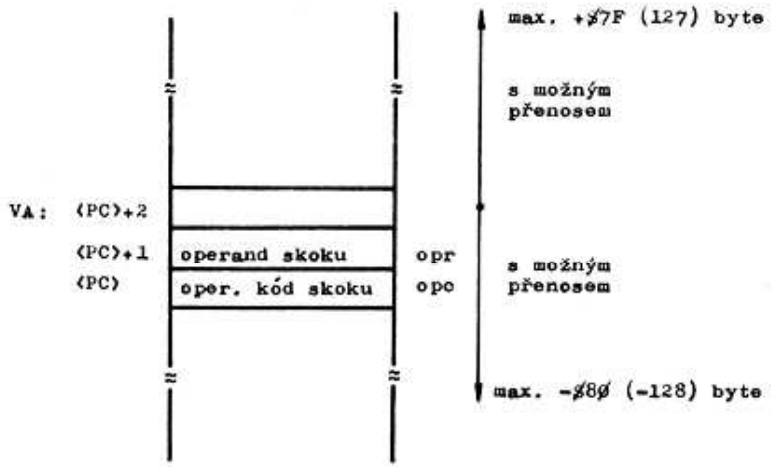


obr. 17 - Adresování ZP,Y

4.10 Adresování REL - Relative - relativní adresování

Tento způsob adresování používají pouze instrukce podmíněných skoků. Na místě operandu je uvedena adresa, na kterou má být skok proveden, jestliže je splněna podmínka skoku. Ve zdrojovém textu je tedy uvedena absolutní adresa skoku. Při překladu do strojového kódu je převedena na relativní adresu, která se vztahuje k tzv. "vztažné adrese".

Vztažná adresa je odvozena od programového ukazatele PC /Program Counter/ a je to vždy adresa následující za instrukcí skoku, jak ukazuje obrázek.



obr. 18 - Adresování REL

Směr skoku je udán nejvýznamnějším bitem operandu strojového kódu. Při b_7=0 dojde ke skoku vpřed, při b_7=1 ke skoku vzad.

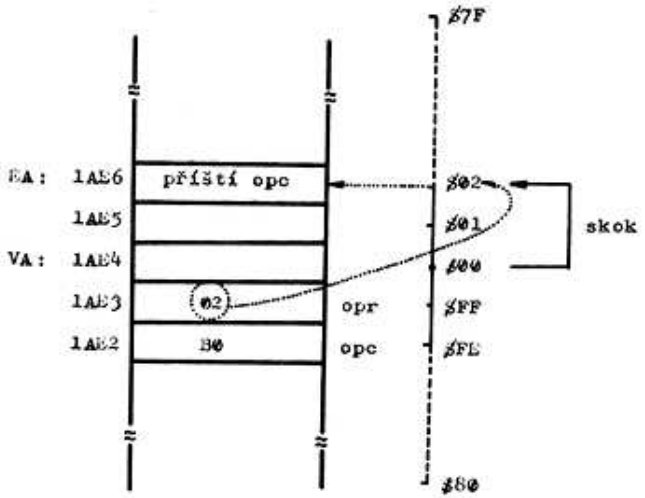
Jde-li o skok vpřed (b_7=0), pak ostatní bity udávají počet byte, o které se má skočit dopředu, tj. na vyšší adresy.

Jde-li o skok vzad, (b_7=1), pak počet byte skoku je vyjádřen doplňkem operandu k číslu $100.

Například chceme-li skočit o 3 byte zpět, bude operand obsahovat: $100-$03, což je $FD.

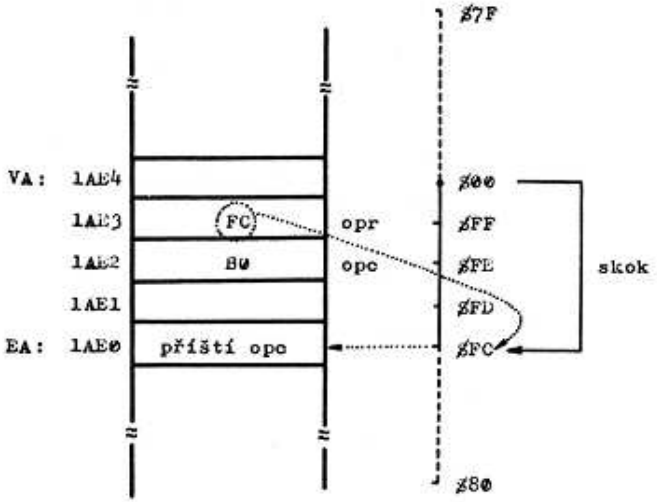
Protože operand je uložen na jednom byte, nelze provést skok na jakoukoli adresu. Směrem dopředu /k vyšším adresám/ lze skočit maximálně o $7F /tj. 127/ byte od vztažné adresy. Směrem vzad /k nižším adresám/ maximálně o $80 /tj. 128/ byte od vztažné adresy.

Příklad 1:
Předpokládejme, že operační kód instrukce skoku leží na adrese $1AE2 a chceme provést skok na adresu $1AE6. Instrukce, která tento skok provede, je BNE $1AE6. Absolutní adresa skoku $1AE6 bude při překladu do strojového kódu převedena na relativní adresu $02, neboť vztažná adresa je $1AE4 a adresa skoku $1AE6 je od ní vzdálena o 2 buňky vpřed, jak ukazuje obrázek.



obr. 19 - Adresování REL - skok vpřed.

Příklad 2:
Podobně jako v příkladu 1 provedeme skok z adresy $1AE2, ale tentokrát směrem vzad na adresu $1AE0, což zajistí instrukce BNE $1AE0. Absolutní adresa skoku $1AE0 bude ve strojovém kódu převedena na relativní adresu $FC, neboť vztažná adresa je $1AE4 a adresa skoku $1AE0 je od ní vzdálena o 4 buňky vzad a doplněk čísla $04 k číslu $100 je $100 - $04, tj. $FC.



obr. 20 - Adresování REL - skok vzad

4.11 Adresování (IND) - Indirect - nepřímé adresování

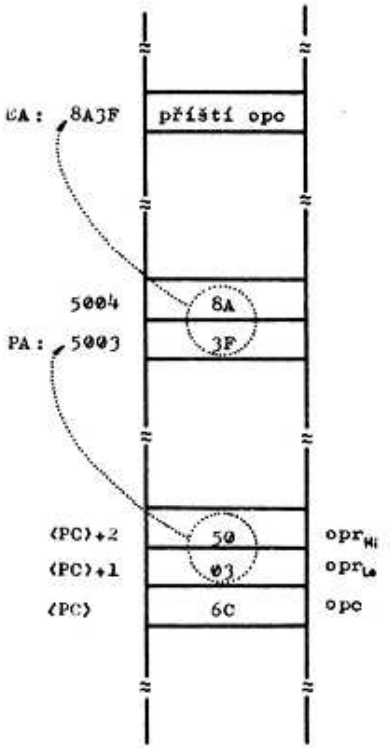
Adresování IND se týká pouze instrukce nepodmíněného skoku JMP. Operand musí být uveden v závorce, jinak by byl chápán jako přímá absolutní adresa.

Operand představuje pomocnou adresu. Na ní leží nižší byte efektivní adresy, o adresu výše je vyšší byte efektivní adresy.

Instrukce je po překladu do strojového kódu tříbajtová. První byte obsahuje operační kód, další dva pomocnou adresu. Protože jsou pro adresu vyhrazeny dva byte, může

pomocná adresa ležet kdekoli v paměti. Totéž platí o efektivní adrese.

Příklad:
Předpokládejme, že ⟨$5003⟩ =$3F a ⟨$5004⟩ =$8A. Potom instrukce JMP ($5003) provede skok na adresu $8A3F, jak ukazuje obrázek.



obr. 21 - Adresování (IND)

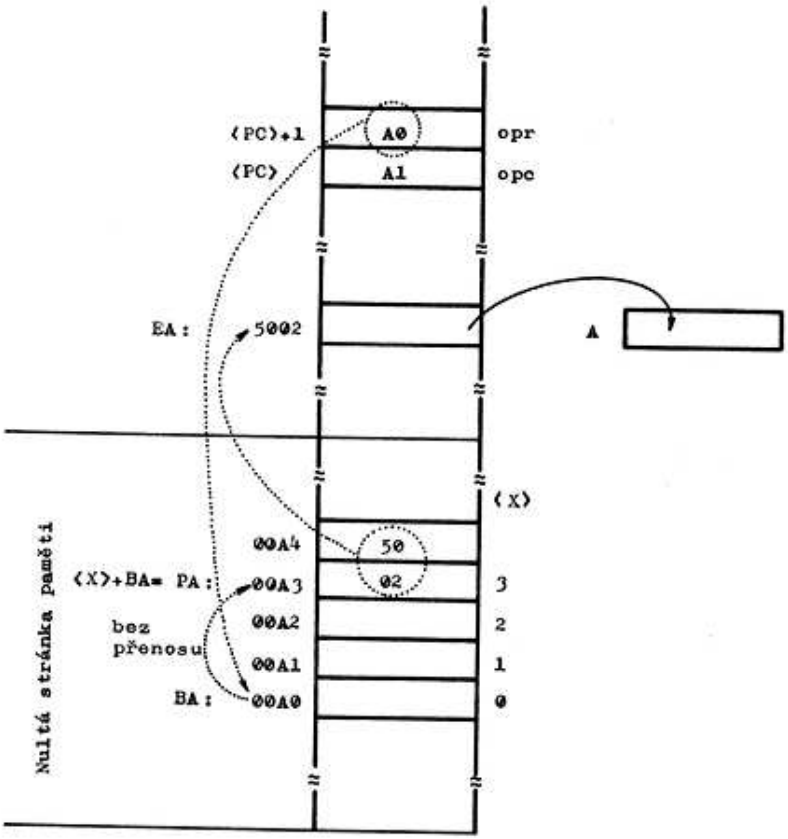
POZOR na nepřímý skok typu JMP ($xyFF), kde xy určují libovolnou stránku paměti $00 až $FF. V tomto případě je na adrese $xyFF nižší byte efektivní adresy a na adrese $xy00 je uložen vyšší byte efektivní adresy! Nedochází k přenosu na vyšší stránku paměti.

4.12 Adresování (IND,X) - X-indexed Indirect - nepřímé adresování přes index-registr X

Jednobajtový operand instrukcí s adresováním (IND,X) obsahuje bázovou adresu a index-registr X. Bázová adresa ukazuje na nultou stránku paměti. Součtem bázové adresy s obsahem index-registru X je určena pomocná adresa, přičemž případný přechod na vyšší stránku paměti se zanedbává. Na pomocné adrese je uložen nižší byte efektivní adresy, o adresu výše je vyšší byte efektivní adresy dat. Efektivní adresa je tedy dvoubajtová a adresovat lze jakékoli místo v paměti.

Instrukce jsou po překladu do strojového kódu dvoubajtové. První byte obsahuje operační kód, druhý bázovou adresu.

Příklad:
Předpokládejme, že ⟨X⟩ =$03, ⟨$00A3⟩ =$02, ⟨$00A4⟩ =$50. Instrukce LDA ($A0,X) pak provede přesun dat z efektivní adresy $5002 do akumulátoru, neboť pomocná adresa je $A0+ $03 = $A3, na adrese $A3 leží hodnota $02 /tj. nižší byte efektivní adresy/ a o adresu výše leží hodnota $50 /vyšší byte efektivní adresy/, jak ukazuje obrázek.



obr. 22 - Adresování (IND,X)

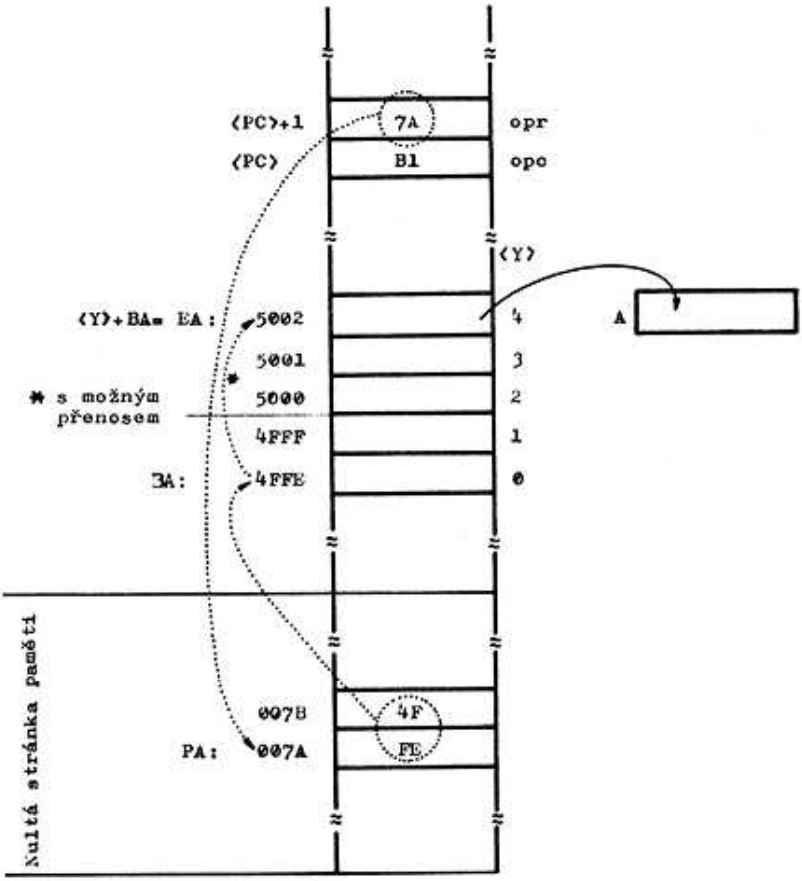
4.13 Adresování (IND),Y - Indirect Y-indexed - nepřímé adresování přes index-registr Y

Jednobajtový operand instrukcí a adresováním (IND),Y obsahuje pomocnou adresu, která ukazuje na nultou stránku paměti. Na pomocné adrese je uložen nižší byte dvoubajtové bázové adresy, o buňku výše je vyšší byte bázové adresy. Efektivní adresa se určí součtem bázové adresy s obsahem index-registru Y. Efektivní adresa je dvoubajtová, adresována tudíž může být kterákoli buňka paměti.

Instrukce jsou po překladu do strojového kódu dvoubajtové. První byte obsahuje operační kód, druhý pomocnou adresu.

Pozor! Rozdíl mezi adresováním (IND,X) a (IND),Y není pouze v používaném index-registru!

Příklad:
Předpokládejme, že ⟨Y⟩ =$04, ⟨$007A⟩ =$FE, ⟨$007B⟩=$4F. Instrukce LDA ($7A),Y přesune do akumulátoru data z efektivní adresy $5002, neboť pomocná adresa $7A obsahuje hodnotu $FE /tj. vyšší byte bázové adresy/, na adrese $7B je hodnota $4F /vyšší byte bázové adresy/ a po sečtení bázové adresy s obsahem akumulátoru ($4FFE+$04) je určena efektivní adresa $5002. Situaci ukazuje obrázek.



obr. 23 - Adresování (IND),Y

5. Instrukční soubor JSA 6502

Instrukční soubor JSA 6502 se skládá z 56 instrukcí, tj. z 56 tříznakových zkratek, které jsou shodné pro všechny překladače rodiny mikroprocesorů 65xx.

Vzhledem k tomu, že instrukce mají více způsobů adresování, existuje celkem 151 různých modifikací. Každé modifikaci je přiřazen jeden operační kód strojové instrukce. Z toho je patrné, že mikroprocesor může vykonávat pouze 151 různých operací.

Operační kód zabírá vždy jeden byte /8 bitů/, což umožňuje vytvořit 2^8 = 256 různých kombinací, ale z toho jich je využito pouze 151. Operační kódy jsou opět shodné pro všechny mikroprocesory 65xx.

Strojová instrukce se skládá z operačního kódu a jednobajtového nebo dvoubajtového operandu. Některé strojové instrukce operand nemají.

Pozor! Operand strojové instrukce není totéž, co operand instrukce v JSA, i když spolu úzce souvisejí.

Instrukční soubor lze členit podle různých hledisek:

I/ Podle způsobu adresování - na instrukce používající:

- a/ implicitní adresování - IMPL,
- b/ přímé adresování - ACC,
- c/ bezprostřední adresování - IMM,
- d/ absolutní adresování - ABS,
- e/ absolutní adresování přes index-registr X - ABS,X,
- f/ absolutní adresování přes index-registr Y - ABS,Y,
- g/ nulostránkové absolutní adresování - ZP,
- h/ nulostránkové adresování přes index-registr X - ZP,X,
- i/ nulostránkové adresování přes index-registr Y - ZP,Y,
- j/ relativní adresování - REL,
- k/ nepřímé adresování - (IND),
- l/ nepřímé adresování přes index-registr X - (IND,X),
- m/ nepřímé adresování přes index-registr Y - (IND),Y.

2/ Podle počtu byte, které zabírá strojová instrukce:

a/ jednobajtové - strojová instrukce má pouze operační kód:

1 byte

b/ dvoubajtové - strojová instrukce je složena z operačního kódu a jednobajtového operandu:

2 byte

c/ tříbajtové - strojová instrukce je složena z operačního kódu a dvoubajtového operandu:

3 byte

3/ Podle činnosti, kterou vykonávají:

a/ instrukce přesunu,

b/ skokové instrukce,

c/ aritmetické instrukce,

d/ logické instrukce,

e/ instrukce NOP.

Existují ještě další způsoby dělení instrukcí.

Přehled všech instrukcí JSA 6502 je uveden v tabulce v příloze 1. Pro jednotlivé instrukce jsou v tabulce uvedeny všechny přípustné způsoby adresování, operandy, počet period, které daná modifikace zabírá, její operační kód, počet byte strojové instrukce a příznaky, které ovlivňuje. Uvedeno je také prováděcí schéma znázorňující činnost instrukce.

5.1 Instrukce přesunů

Tyto instrukce provádějí přesuny dat ze zdrojového do cílového místa. Místem zdroje nebo cíle může být registr, byte paměti, byte zápisníku /tj. byte nulté stránky paměti/, byte zásobníku.

Podle místa zdroje a cíle lze instrukce přesunů rozdělit takto:

- přesuny dat mezi registry A, X, Y, S
  /TAX, TAY, TXA, TYA, TSX, TXS/,
- přesuny dat mezi registry A, X, Y a pamětí
  /LDA, LDX, LDY, STA, STX, STY/,
- přesuny dat mezi zásobníkem a registry A, P
  /PHA, PHL, PLA, PLP/.

U všech instrukcí přesunů platí, že obsah zdroje je po přesunu zachován.

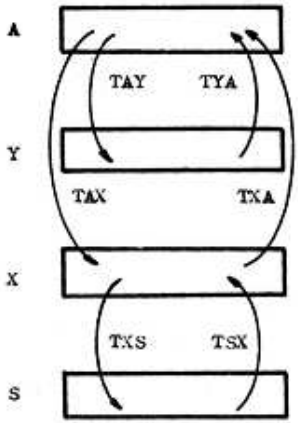
5.1.1 Instrukce přesunů dat mezi registry A, X, Y, S

Do této skupiny patří instrukce TAX, TAY, TXA, TYA, TSX, TXS. Ovlivňují pouze příznaky N /Negative/ a Z /Zero/ podle hodnoty přenášených dat, ostatní příznaky zůstávají zachovány. Výjimkou je instrukce TXS, která neovlivňuje žádný příznak.

Všechny používají adresování IMPL /implicitní/. To znamená, že informaci o zdrojovém i cílovém registru je zakódována přímo v operačním kódu.

Mnemotechnická pomůcka: V názvu instrukcí je vždy první znak T /Transfer = přenést/, druhý znak označuje zdrojový registr a třetí znak cílový registr.

Přehled možných přenosů mezi registry poskytuje obrázek:



obr. 24 - Instrukce typu Transfer

Instrukce TAX

Provádí přesun obsahu akumulátoru do index-registru X. Podle hodnoty přenášených dat jsou nastaveny příznaky N a Z.

Schéma:
⟨A⟩ ⟶ ⟨X⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
TAX	IMPL	⟨A⟩ ⟶ ⟨X⟩

TAB. 6 Instrukce TAX

Instrukce TAY

Provádí přesun obsahu akumulátoru do index-registru Y. Podle hodnoty přenášených dat jsou nastaveny příznaky N a Z.

Schéma:
⟨A⟩ ⟶ ⟨Y⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
TAY	IMPL	⟨A⟩ ⟶ ⟨Y⟩

TAB. 7 Instrukce TAY

Instrukce TXA

Provádí přesun obsahu index-registru X do akumulátoru. Podle hodnoty přenášených dat jsou nastaveny příznaky N a Z.

Schéma:
⟨X⟩ ⟶ ⟨A⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
TXA	IMPL	⟨X⟩ ⟶ ⟨A⟩

TAB. 8 Instrukce TXA

Instrukce TSX

Provádí přesun obsahu registru S (ukazatele zásobníku) do index-registru X. Podle hodnoty přenášených dat jsou nastaveny příznaky N a Z.

Schéma:
⟨S⟩ ⟶ ⟨X⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
TSX	IMPL	⟨S⟩ ⟶ ⟨X⟩

TAB. 9 Instrukce TSX

Používá se například pro testování polohy ukazatele nebo pro případné matematické operace s ukazovátkem.

Instrukce TXS

Provádí přesun obsahu index-registru X do registru S (ukazatele vrcholu zásobníku). Neovlivňuje žádné příznaky.

Schéma:
⟨X⟩ ⟶ ⟨S⟩ ovlivní: -

Formální zápis inst.	Adres.	Činnost
TXS	IMPL	⟨X⟩ → ⟨S⟩

TAB. 10 Instrukce TXS

Instrukce slouží k nastavení ukazatele zásobníku. Proto je třeba s ní pracovat velmi opatrně. Nevhodné použití vede k havárii systému.

Pomocí instrukce TXS nastavuje operační systém po zapnutí počítače tzv. vrchol zásobníku.

Příklady:

1/ Přesun obsahu registru X do registru Y:

```
TXA     ; ⟨X⟩ → ⟨A⟩
TAY     ; ⟨A⟩ → ⟨Y⟩
```

2/ Nastavení ukazatele vrcholu zásobníku S na hodnotu uloženou v registru Y:

```
TYA     ; ⟨Y⟩ → ⟨A⟩
TAX     ; ⟨A⟩ → ⟨X⟩
TXS     ; ⟨X⟩ → ⟨S⟩
```

Příklady ukazují přesun dat mezi registry, jestliže není možno provést přímý přesun.

5.1.2 Instrukce přesunů dat mezi registry A, X, Y a pamětí

Do této skupiny patří instrukce: LDA, LDX, LDY, STA, STX, STY. Používají různé způsoby adresování a podle toho jsou pak dvoubajtové nebo tříbajtové.

Instrukce LDA, LDX, LDY provádějí přesun dat z jednoho byte paměti do příslušného registru. Podle hodnoty přenášených dat ovlivňují příznaky N /Negative/ a Z /Zero/. Ostatní příznaky zachovávají svoji původní hodnotu.

Instrukce STA, STX, STY přesouvají obsah registrů A, X, Y na příslušné místo v paměti, které je určeno použitým způsobem adresování. Neovlivňují žádné příznaky!

Instrukce LDA

Provádí přesun dat do akumulátoru. Přenášenými daty může být konstanta uvedená jako operand nebo data z efektivní adresy. Efektivní adresa je určena v závislosti na použitém způsobu adresování.

Podle hodnoty přenášených dat jsou nastaveny příznaky N a Z.

Schéma:
$\langle M \rangle \longrightarrow \langle A \rangle$ ovlivní: $\langle N \rangle$, $\langle Z \rangle$

Formální zápis inst.	Adres.	Činnost
LDA#op_8	IMM	přesun konstanty op_8 do akumulátoru
LDA op_{16}	ABS	přesun dat z absolutní adresy op_{16} do akumulátoru
LDA op_{16},X	ABS,X	přesun dat z adresy $op_{16}+\langle X \rangle$ do akumulátoru
LDA op_{16},Y	ABS,Y	přesun dat z adresy $op_{16}+\langle Y \rangle$ do akumulátoru
LDA op_8	ZP	přesun dat z nulostránkové adresy op_8 do akumulátoru
LDA op_8,X	ZP,X	přesun dat do akumulátoru z adresy $op_8+\langle X \rangle$ bez přechodu na vyšší stránku paměti
LDA (op_8,X)	(IND,X)	přesun dat z adresy $(op_8+\langle X \rangle)$ do akumulátoru

LDA (op_8),Y	(IND),Y	přesun dat z adresy (op_8)+⟨Y⟩

TAB. 11 Instrukce LDA

Instrukce LDX

Provádí přesun dat do index-registru X. Přenášenými daty může být konstanta uvedená jako operand nebo data z efektivní adresy, která je určena podle použitého způsobu adresování.

Podle hodnoty přenášených dat jsou ovlivněny příznaky N a Z.

Schéma:
⟨M⟩ ⟶ ⟨X⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
LDX #op_8	IMM	přesun konstanty op_8 do index-registru X
LDX op_{16}	ABS	přesun dat z absolutní adresy op_{16} do index-registru X
LDX op_{16},Y	ABS,Y	přesun dat z adresy op_{16}+⟨Y⟩ do index-registru Y
LDX op_8	ZP	přesun dat z nulostránkové adresy op_8 do index-registru X
LDX op_8,Y	ZP,Y	přesun dat do index-registru X z adresy op_8+⟨Y⟩ bez přechodu na vyšší stránku paměti

TAB. 12 Instrukce LDX

Instrukce LDY

Provádí přesun dat do index-registru Y. Přenášenými daty může být konstanta uvedená jako operand nebo data z efektivní adresy, která je určena podle použitého způsobu adresování.

Podle hodnoty přenášených dat jsou nastaveny příznaky N a Z.

Schéma:
⟨M⟩ ⟶ ⟨Y⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
LDY #op_8	IMM	přesun konstanty op_8 do registru Y
LDY op_{16}	ABS	přesun dat z absolutní adresy op_{16} do registru Y
LDY op_{16},X	ABS,X	přesun dat z adresy op_{16}+⟨X⟩ do registru Y
LDY op_8	ZP	přesun dat z nulostránkové adresy op_8 do registru Y
LDY op_8,X	ZP,X	přesun dat z adresy op_8+⟨X⟩ (bez přechodu na vyšší stránku paměti) do registru X

TAB. 13 Instrukce LDY

Instrukce STA

Provádí přesun obsahu akumulátoru na paměťové místo, jehož efektivní adresa je určena podle použitého způsobu adresování.

Hodnoty přenášených dat neovlivňuje žádné příznaky.

Schéma:
⟨A⟩ ⟶ ⟨M⟩ ovlivní: -

Formální zápis inst.	Adres.	Činnost
STA op_{16}	ABS	přesun dat z akumulátoru na adresu op_{16}
STA op_{16},X	ABS,X	přesun dat z akumulátoru na adresu op_{16}+ ⟨X⟩
STA op_{16},Y	ABS,Y	přesun dat z akumulátoru na adresu op_{16}+ ⟨Y⟩
STA op_8	ZP	přesun dat z akumulátoru na nulostránkovou adresu op_8
STA op_8,X	ZP,X	přesun dat z akumulátoru na adresu op_8+ ⟨X⟩ bez přechodu na vyšší stránku paměti
STA (op_8,X)	(IND,X)	přesun dat z akumulátoru na adresu (op_8+⟨X⟩)
STA (op_8),Y	(IND),Y	přesun dat z akumulátoru na adresu (op_8)+⟨Y⟩

TAB. 14 Instrukce STA

Instrukce STX

Provádí přesun dat z registru X na paměťové místo, jehož efektivní adresa je určena použitým způsobem adresování.

Hodnota přenášených dat neovlivňuje žádné příznaky.

Schéma:

⟨X⟩ ⟶ ⟨M⟩ ovlivní: -

Formální zápis inst.	Adres.	Činnost
STX op_{16}	ABS	přesun dat z registru X na adresu op_{16}
STX op_8	ZP	přesun dat z registru X na nulostránkovou adresu op_8
STX op_8,Y	ZP,Y	přesun dat z registru X na adresu op_8+⟨Y⟩ bez přechodu na vyšší stránku paměti

TAB. 15 Instrukce STX

Instrukce STY

Provádí přesun dat z registru Y na paměťové místo, jehož efektivní adresa je určena použitým způsobem adresování.

Hodnota přenášených dat neovlivňuje žádné příznaky.

Schéma:
⟨Y⟩ ⟶ ⟨M⟩ ovlivní: -

Formální zápis inst.	Adres.	Činnost
STY op_{16}	ABS	přesun dat z registru Y na adresu op_{16}
STY op_8	ZP	přesun dat z registru Y na nulostránkovou adresu op_8
STY op_8,X	ZP,X	přesun dat z registru Y na adresu op_8+⟨X⟩ bez přechodu na vyšší stránku paměti

TAB. 16 Instrukce STY

Příklady:

1/ Přesuňte obsah buňky $CC na buňku $70AB.

a/
```
LDA $CC
STA $70AB
```

b/
```
LDX $CC
STX $70AB
```

2/ Vynulujte registry A, X, Y.

a/
```
LDA#0
LDX#0
LDY#0
```

b/
```
LDA#0
TAX
TAY
```

Řešení b/ je úspornější z hlediska paměti, ale z hlediska času provedení jsou obě řešení naprosto stejná.

5.1.3 Instrukce přesunů dat mezi zásobníkem a registry A, P

Do této skupiny patří instrukce: PHA, PHL, PLA, PLP. První dvě instrukce neovlivňují žádné příznaky. Naopak instrukce PLP ovlivňuje všechny příznaky. **Instrukce PLA ovlivňuje příznaky N, Z.**

Instrukce PHA

Ukládá obsah akumulátoru na vrchol zásobníku, tedy na adresu uloženou v registru S. Po provedení operace se sníží hodnota obsahu registru S o jedničku, čímž se nastaví nový vrchol zásobníku.

Neovlivňuje žádné příznaky.

Schéma:

$\langle A \rangle \longrightarrow \langle TOP \rangle$

$\langle S \rangle - 1 \longrightarrow \langle S \rangle$

ovlivní: -

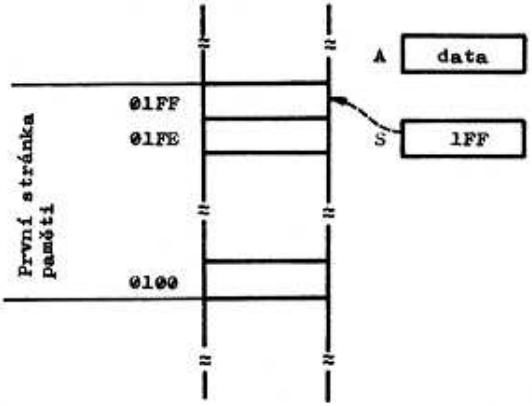
Formální zápis inst.	Adres.	Činnost
PHA	IMPL	uložení A na vrchol zásobníku

TAB. 17 Instrukce PHA

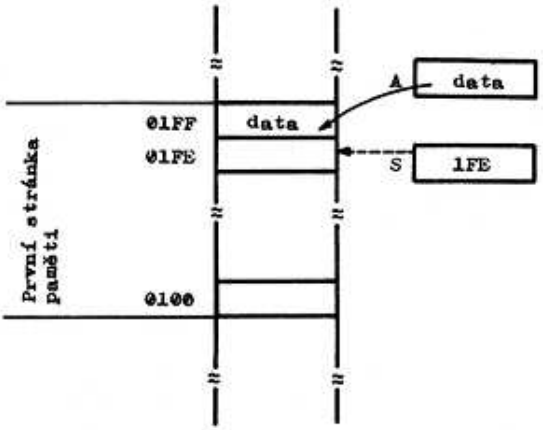
Používá se ke krátkodobému ukládání dat, k předávání parametrů mezi moduly a pod.

Příklad:
Předpokládejme, že ⟨S⟩=$1FF, tj. zásobník je zcela prázdný a první volná buňka je na adrese $1FF. Situace před a po provedení instrukce PHA vypadá takto:

Před provedením operace :



Po provedení operace :



obr, 25 - Činnost instrukce PHA

Instrukce PHP

Ukládá obsah příznakového registru na vrchol zásobníku, tj. na adresu uloženou v registru S. Po provedení operace sníží obsah registru S o jedničku, čímž se nastaví nový vrchol zásobníku.

Neovlivňuje žádné příznaky.

Schéma:

⟨P⟩ ⟶ ⟨TOP⟩

⟨S⟩ -1 ⟶ ⟨S⟩

ovlivní: -

Formální zápis inst.	Adres.	Činnost
PHP	IMPL	uložení P na vrchol zásobníku

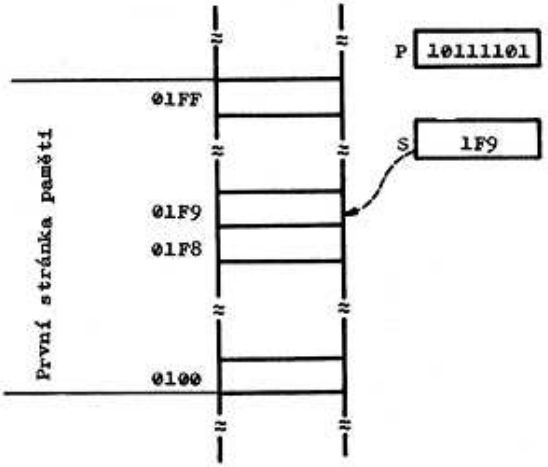
TAB. 18 Instrukce PHP

Používá se například pro uchování hodnot příznaků pro jejich pozdější testování a větvení programu podle jejich hodnot.

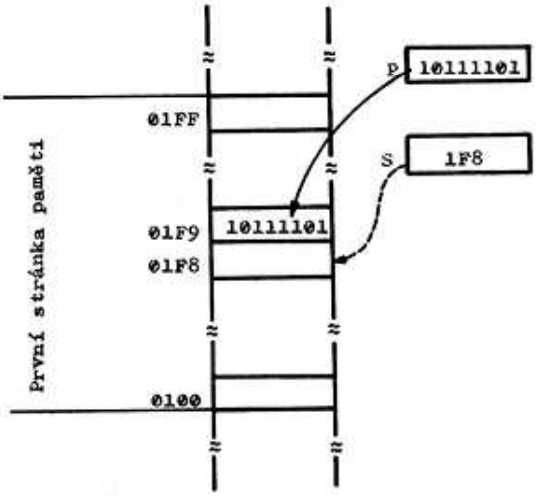
Příklad:

Předpokládejme, že ⟨S⟩=$1F9. Pak situace před a po provedení instrukce PHP vypadá takto:

Před provedením operace :



Po provedení operace :



obr. 26 - Činnost instrukce PHP

Instrukce PLA

Provádí nejprve zvýšení hodnoty ukazatele vrcholu zásobníku S o jedničku, pak přesun obsahu vrcholu zásobníku do akumulátoru. Provádí opačnou činnost než instrukce PHA.

Ovlivňuje příznaky N, Z.

Schéma:

⟨S⟩ +1 → ⟨S⟩

⟨TOP⟩ → ⟨A⟩

ovlivní: ⟨N⟩, ⟨Z⟩

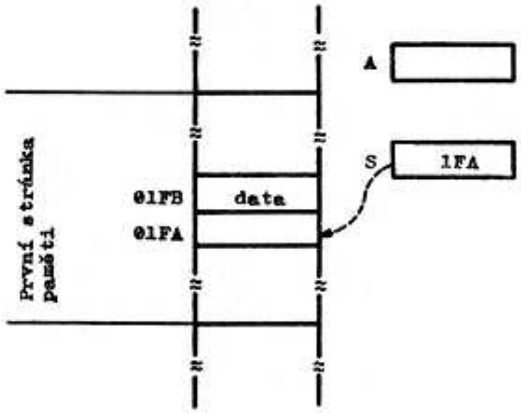
Formální zápis inst.	Adres.	Činnost
PLA	IMPL	přesun obsahu vrcholu zásobníku do registru A

TAB. 19 Instrukce PLA

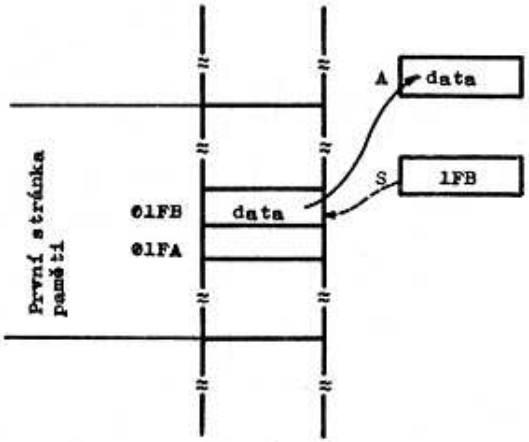
Příklad:

Předpokládejme, že ⟨S⟩=$1FA. Pak situace před a po provedení instrukce PLA vypadá takto:

Před provedením operace:



Po provedení operace:



obr. 27 - Činnost instrukce PLA

Instrukce PLP

Provede nejprve zvýšení hodnoty registru S o jedničku, pak přesune obsah vrcholu zásobníku do příznakového registru P. Provádí opačnou činnost než instrukce PHP.

Nastavuje všechny příznaky.

Schéma:

⟨S⟩ +1 → ⟨S⟩
⟨TOP⟩ → ⟨P⟩ ovlivní: ⟨N⟩, ⟨V⟩, ⟨B⟩, ⟨D⟩, ⟨I⟩, ⟨Z⟩, ⟨C⟩

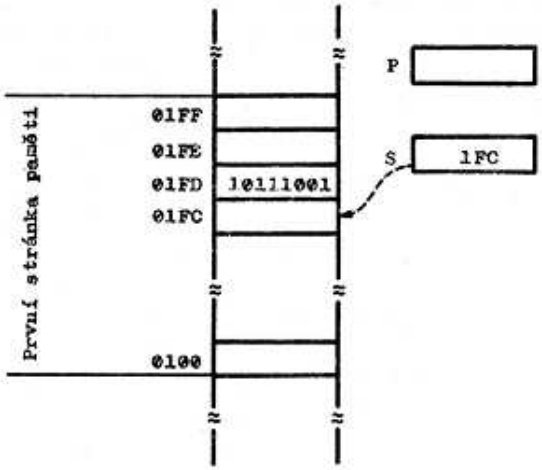
Formální zápis inst.	Adres.	Činnost
PLP	IMPL	přesun obsahu vrcholu zásobníku do příznakového registru P

TAB. 20 Instrukce PLP

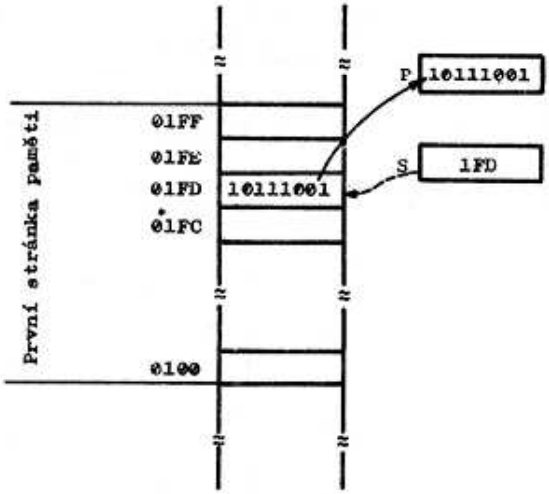
Používá se například k obnovení obsahu příznakového registru nebo k nepřímému nastavení příznaků.

Příklad:
Předpokládejme, že ⟨S⟩=$1FC, pak situaci před a po provedení instrukce PLP vypadá takto:

Před provedením operace:



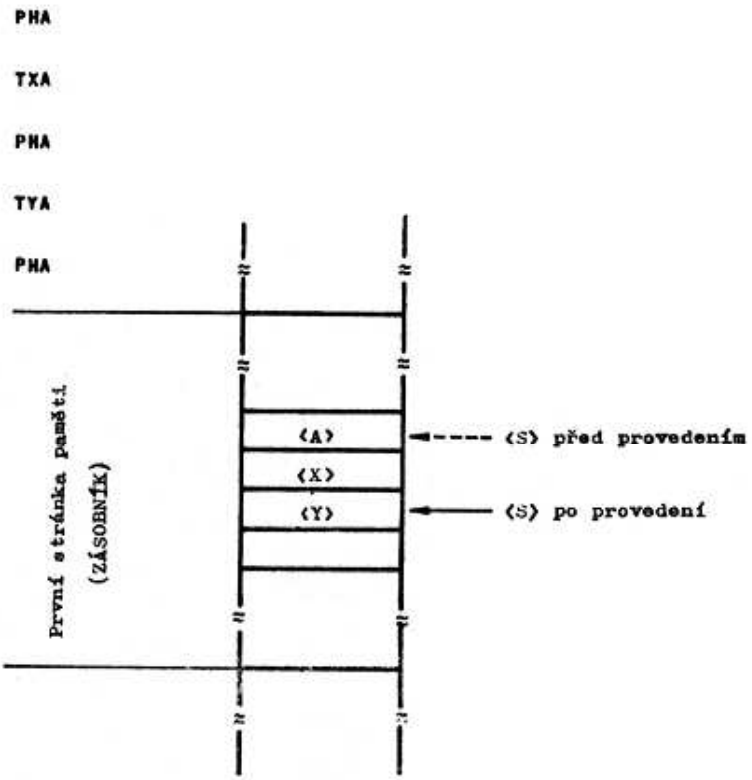
Po provedení operace:



obr. 28 - Činnost instrukce PLP

Příklady:

1/ uložte obsahy registrů A, X, Y do zásobníku.



obr. 29 - Princip ukládání informací do zásobníku

Sekvence SEKV1 se obvykle používá na začátku podprogramů, sekvence SEKV2 na konci, tj. těsně před instrukcí RTS. Případně se SEKV1 používá před voláním podprogramů (tj. před instrukcí JSR) a SEKV2 po návratu z podprogramů (tj. po instrukci JSR). Těmito postupy se zaručí, že se zachovají původní hodnoty registrů.

2/ Nastavte obsah příznakového registru P podle obsahu buňky $3A a uložte ho do zásobníku.

```
        LDA $3A         ; ovlivní příznaky N a Z
        PHP
```

3/ Přesuňte obsah příznakového registru P do akumulátoru.

```
        PHP
        PLA
```

4/ Nastavte příznakový registr P na hodnotu %11100001 (tj. ⟨N⟩=1, ⟨V⟩=1, ⟨B⟩=0, ⟨D⟩=0, ⟨I⟩=0, ⟨Z⟩=0, ⟨C⟩=1).

```
        LDA#%11100001     ; N, V, 1, B, D, I, Z, C
        PHA
        PLP
```

5/ Sekvencí SEKV1 uložte obsahy registrů P, A, X, Y do zásobníku a sekvencí SEKV2 vraťte těmto registrům jejich původní hodnoty.

```
SEKV1   PHP
        PHA
        TXA
        PHA
        TYA
        PHA
        ...
SEKV2   PLA
        TAY
        PLA
        TAX
        PLA
        PLP
```

5.2 Skokové instrukce

Skokové instrukce předávají řízení na jiné místo programu, tj. mění obsah programového ukazatele PC. Novým obsahem PC je efektivní adresa příští instrukce, resp. jejího operačního kódu.

Můžeme je rozdělit na dvě skupiny:

- instrukce nepodmíněných skoků
  /JMP, JSR, RTS, BRK, RTI/,
- instrukce podmíněných skoků
  /BNE, BEQ, BMI, BPL, BVC, BVS, BCC, BCS/.

Skokové instrukce slouží k větvení programu, hnízdění /vnořování/, k vyvedení programu ze slepé větve atp.

5.2.1 Instrukce nepodmíněných skoků

Provádějí bezpodmínečné předání řízení na jiné místo programu. Patří sem instrukce:

- přímého skoku JMP op_{16}
- nepřímého skoku JMP (op_{16})
- volání podprogramu JSR op_{16}
- návratu z podprogramu RTS
- programového přerušení BRK
- návratu z přerušení RTI

Instrukce RTI ovlivňuje všechny příznaky, BRK nastavuje příznak B /Break/ na log. 1. Ostatní instrukce neovlivňují žádné příznaky.

Instrukce JMP

Provádí skok na jiné místo programu, které je určeno podle použitého způsobu adresování.

Nenastavuje žádné příznaky.

Schéma:

M ⟶ ⟨PC⟩ ovlivní: -

Formální zápis inst.	Adres.	Činnost
JMP op_{16}	ABS	přímý skok na absolutní adresu op_{16}
JMP (op_{16})	IND	nepřímý skok na adresu (op_{16})

TAB. 21 Instrukce JMP

JMP op_{16} provádí přímý skok na absolutní adresu op_{16}. Je to obdoba instrukce GOTO adr. v Basicu. Adresa uvedená v operandu instrukce je přímo efektivní adresa skoku. Do PC bude tedy uložena přímo adresa op_{16}.

Používá se:

- k překlenutí zóny dat nebo jiného programu,
- k vyvedení ze slepé větve,
- k realizaci tabulky vnějších vstupů, tabulky spojů, tabulky knihovny podprogramů

JMP (op_{16}) provádí nepřímý skok na efektivní adresu. Adresa op_{16} je pomocná, ukazuje na místo, kde je uložen nižší byte efektivní adresy. O buňku výše je vyšší byte efektivní adresy. Do PC bude tedy uložena tímto nepřímým způsobem určená efektivní adresa skoku. Jinak řečeno - na adrese op_{16} je uložen vektor skoku.

Používá se podobně jako JMP op_{16}, ale navíc:

- pro tabulkově řízené skoky, skoky přes vektory,
- k vyvedení rutiny operačního systému do uživatelské oblasti /viz. např. program operačního systému pro zpracování přerušení $\overline{NMI}$, $\overline{IRQ}$ a pod./.

POZOR na skok typu JMP ($xyFF), kde xy určuje stránku $00 až $FF. V tomto případě je na adrese op_{16} /tj. $xyFF/ uložen nižší byte efektivní adresy a na adrese $xy00 /tj. na adrese o 255 byte níže/ je uložen vyšší byte efektivní adresy! Instrukci totiž chybí jeden takt na případné zpracování přetečení do vyšší stránky.

Příklady:

1/ Proveďte skok na adresu $A000, kde je vstupní bod interpreteru Basic /teplý start/.

Řešení A: /přímý skok/

```
          JMP $A000
```

Řešení B: /přímý skok pomocí návěští/

```
BASIC     BRU $A000
          JMP BASIC
```

Řešení C: /nepřímý skok/

```
          JMP (VEKTOR)
VEKTOR    DFW $A000
```

2/ Je-li aktivní OS ROM a je-li vyvoláno HW přerušení IRQ nebo SW přerušení instrukcí BRK, pokračuje program na adrese, kde je uložen následující program:

```
          CLD
          JMP ($0216)
```

Adresa počátku tohoto programu je uložena na posledních dvou buňkách paměti. Viz kapitola 9. Přerušení.

Instrukce JSR

Je to instrukce volání podprogramu. Předává řízení na první instrukci podprogramu, která je určena přímo operandem instrukce. Řízení se vrátí zpět, narazí-li se na instrukci RTS.

Při volání podprogramu instrukcí JSR se na vrchol zásobníku ukládá "návratová adresa". Ve skutečnosti je na vrchol zásobníku uložena návratová adresa zmenšená o jedničku. Skutečná návratová adresa /tj. adresa instrukce následující bezprostředně za instrukcí JSR/ je při provádění instrukce RTS určena součtem obsahu vrcholu zásobníku s jedničkou.

Pozor na rozdíl od mikroprocesorů I8080 a Z80, kde je na vrchol zásobníku ukládána skutečná adresa.

Schéma:

$\langle PC \rangle + 2 \longrightarrow \langle PC \rangle$

$\langle PCH \rangle \longrightarrow \langle TOP \rangle$

$\langle S \rangle - 1 \longrightarrow \langle S \rangle$

$\langle PCL \rangle \longrightarrow \langle TOP \rangle$

$\langle S - 1 \rangle \longrightarrow \langle S \rangle$

$op_{16} \longrightarrow \langle PC \rangle$

ovlivní: -

Formální zápis inst.	Adres.	Činnost
JSR op_{16}	ABS	volání podprogramu

TAB. 22 Instrukce JSR

Pozn.: Uvnitř podprogramu můžeme volat další podprogram. Úroveň vnoření je omezena rozsahem zásobníku.

Používá se:

- pro sekvence příkazů, které se opakují - vede k úspoře paměti,
- pro volání podprogramů operačního systému,
- pro rozložení úlohy na jednodušší části - dekompozice, což je základní prvek strukturovaného programování /viz. program RAMTEST ve "Sbírce"/.

Příklad:

Zobrazte písmeno A na obrazovku na aktuální pozici kurzoru.

Řešení:

```
LDA #'A          ; rutina operačního systému
JSR $F2B0        ; pouze pro ATARI řady XL a XE
```

Instrukce RTS

Je to instrukce návratu z podprogramu. Ukončuje podprogram, který byl volán instrukcí JSR. Je to obdoba příkazu RETURN v Basicu.

Při provádění instrukce RTS se ze zásobniku vybere "návratová adresa", přičte se k ní jednička. Tím je určena skutečná návratová adresa, kterou je naplněn programový čítač PC. Program pak pokračuje instrukcí následující bezprostředně za voláním podprogramu.

Schéma:

⟨S⟩ +1 ⟶ ⟨S⟩ ovlivní: -
⟨TOP⟩ ⟶ ⟨PCL⟩
⟨S⟩ +1 ⟶ ⟨S⟩
⟨TOP⟩ ⟶ ⟨PCH⟩
⟨PC⟩ +1 ⟶ ⟨PC⟩

Formální zápis inst.	Adres.	Činnost
RTS	IMPL	návrat z podprogramu

TAB. 23 Instrukce RTS

Používá se:

- k ukončení podprogramu volaného instrukcí JSR,
- k trikovému skoku na absolutní adresu /tzv. vektorový skok - Paulova metoda/ - tímto způsobem je v operačním systému předáváno řízení výkonným rutinám, které ošetřují vstupní/výstupní operace.

Příklad:
Pomocí Paulovy metody předejte řízení na adresu $A000.
Řešení:

```
ADRESA   EQU $A000-1        ; korekce pro RTS
         LDA#ADRESA:H
```

```
PHA                 ; Hi
LDA#ADRESA:L
PHA                 ; Lo
RTS                 ; přímý skok na adresu $A000
```

Instrukce BRK

Je to instrukce programového přerušení. Dojde-li program na tuto instrukci, je přerušen. Řízení je předáno na efektivní adresu, která je uložena na posledních dvou buňkách paměti $FFFE, $FFFF . Do PC bude tedy vložen obsah posledních dvou buněk paměti. Do přerušeného programu se řízení vrátí, narazí-li se na instrukci RTI.
POZOR! Řízení je vráceno o jednu adresu dále než je následující adresa za instrukcí BRK. Neboli o dvě adresy dál než je adresa instrukce BRK.

Činnost je podobná jako u hardwarového přerušení typu $\overline{IRQ}$. Zde však instrukce RTI vrací řízení bezprostředně za naposledy provedenou instrukcí před přerušením.

Při vykonávání instrukce BRK dochází k uložení redukované návratové adresy do zásobníku. Na vrchol zásobníku se ukládá také aktuální obsah příznakového registru P.

Systém je o probíhající rutině programového přerušení informován příznakem B, který je nastaven na log. 1.

Schéma:

ovlivní: ⟨B⟩

⟨PC⟩ +1 ⟶ ⟨PC⟩
⟨PCH⟩ ⟶ ⟨TOP⟩
⟨S⟩ -1 ⟶ ⟨S⟩
⟨PCL⟩ ⟶ ⟨TOP⟩
⟨S⟩ -1 ⟶ ⟨S⟩
⟨P⟩ ⟶ ⟨TOP⟩
⟨S⟩ -1 ⟶ ⟨S⟩
⟨$FFFE⟩ ⟶ ⟨PCL⟩
⟨$FFFF⟩ ⟶ ⟨PCH⟩
1 ⟶ ⟨B⟩

Formální zápis inst.	Adres.	Činnost
BRK	IMPL	programové přerušení

TAB. 24 Instrukce BRK

Používá se k nastavování ladicích bodů /break point/ při ladění programů.

Příklad:

V makroasembleru ATMAS-II lze instrukci BRK použít jako ladicí bod. Pokud program na tuto instrukci narazí, pak je řízení předáno zpět do monitoru a je vypsán obsah registrů: PC, A, X, Y, S, P. Touto instrukcí můžeme též ukončit program v ATMAS-II.

Například:

```
          ORG $5000
          LDA#$55
          LDX#$AA
          LDY#$4F
          BRK
```

Po přeložení a odstartování této sekvence vypíše systém na obrazovku:

```
MONITOR
PC    AC  XR  YR  NV.BDIZC
5006  55  AA  4F  00110000

MONITOR
■
```

Instrukce RTI

Ukončuje program přerušovací rutiny, která byla vyvolána buď programovým přerušením instrukcí BRK, nebo hardwarovým přerušením, které bylo vyvoláno vstupním signálem

mikroprocesoru $\overline{IRQ}$ /maskovatelné přerušení/ nebo signálem $\overline{NMI}$ /nemaskovatelné přerušení/.

Při provádění instrukce RTI se ze zásobníku vybere nejprve příznakový registr P, potom skutečná návratová adresa, která se uloží do PC a tím se řízení vrátí do místa, kde k přerušení došlo.

Schéma:

⟨S⟩ +1 ⟶ ⟨S⟩
⟨TOP⟩ ⟶ ⟨P⟩
⟨S⟩ +1 ⟶ ⟨S⟩
⟨TOP⟩ ⟶ ⟨PCL⟩
⟨S⟩ +1 ⟶ ⟨S⟩
⟨TOP⟩ ⟶ ⟨PCH⟩

ovlivní: ⟨N⟩, ⟨V⟩, ⟨B⟩, ⟨D⟩, ⟨I⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
RTI	IMPL	návrat z programového nebo hardwarového přerušení

TAB. 25 Instrukce RTI

5.2.2 Instrukce podmíněných skoků

Předávají řízení na jiné místo programu, jestliže je splněna podmínka skoku. Není-li podmínka splněna, skok se neprovede a program pokračuje instrukcí uvedenou bezprostředně za instrukcí podmíněného skoku.

Operand v JSA obsahuje absolutní adresu skoku, která smí být vzdálena maximálně o +127 byte /tj. $7F/ nebo -128 byte /tj. -$80/ od adresy následující instrukce.

Při překladu do strojového kódu je absolutní adresa skoku převedena na relativní adresu, která udává počet byte, o které se má skočit, a směr skoku. Počet byte skoku

se počítá od adresy instrukce, která následuje za instrukcí skoku /viz kapitola 4.10/.

Není-li podmínka skoku splněna, trvají instrukce podmíněného skoku 2-hodinové cykly /tzv. periody/. Je-li podmínka splněna a adresa skoku leží na stejné stránce paměti, pak trvají 3 periody, leží-li na předcházející nebo následující stránce paměti, trvají 4 periody.

Do této skupiny patří instrukce: BEQ, BNE, BMI, BPL, BCS, BCC, BVS, BVC.

Podmínky, které musejí být splněny pro jednotlivé instrukce, aby došlo ke skoku, jsou uvedeny v tabulce:

Instrukce	Podmínka	
BEQ	⟨Z⟩ =1	nulovost
BNE	⟨Z⟩ =0	nenulovost
BMI	⟨N⟩ =1	záporност
BPL	⟨N⟩ =0	kladnost
BCS	⟨C⟩ =1	přenos
BCC	⟨C⟩ =0	nenastal přenos
BVS	⟨V⟩ =1	přetečení mantisy
BVC	⟨V⟩ =0	nepřetečení mantisy

TAB. 26 Instrukce podmíněných skoků

Instrukce neovlivňují žádné příznaky, pouze je využívají.

Instrukce BEQ

Provádí skok na adresu uvedenou v operandu, jestliže příznak nulovosti Z obsahuje log. 1. Není-li podmínka splněna, skok se neprovede a program pokračuje na další instrukci /uvedenou za instrukcí BEQ/.

Nenastavuje žádné příznaky.

Schéma:
skok, jestliže ⟨Z⟩ =1 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BEQ op_{16}	REL	skok na adresu op_{16}, je-li $\langle Z \rangle = 1$

TAB. 27 Instrukce BEQ

Příklad:

Realizujte podmíněné rozvětvení programu podle příznaku Zero. Bude-li $\langle Z \rangle = 1$, předejte řízení na adresu ADR1, bude-li $\langle Z \rangle = 0$, na adresu ADR2, přičemž adresy ADR1 a ADR2 mohou ležet kdekoli v paměti.

Řešení:

```
        BEQ ZERO
        JMP ADR2            ; skok, je-li <Z>=0
ZERO    JMP ADR1            ; skok, je-li <Z>=1
```

Tento postup není vhodné používat, protože:
- nevychází ze zásad strukturovaného programování,
- znepřehledňuje řešení,
- není kompaktní,
- zavádí prvky absolutního adresování a tudíž na úrovni strojového kódu není program přemístitelný.

Výhodnější je /po úpravě algoritmu řešení/ přejít na jednodušší konstrukci. Například na konstrukci podmíněný přeskok. Viz kapitola 8. Zásady programování v JSA, resp. 8.1.5 Zásada přednosti jednodušší konstrukce před složitější.

Instrukce BNE

Provádí skok na adresu uvedenou v operandu instrukce, jestliže příznak nulovosti Z obsahuje log. 0. Není-li podmínka splněna, pokračuje program následující instrukcí.

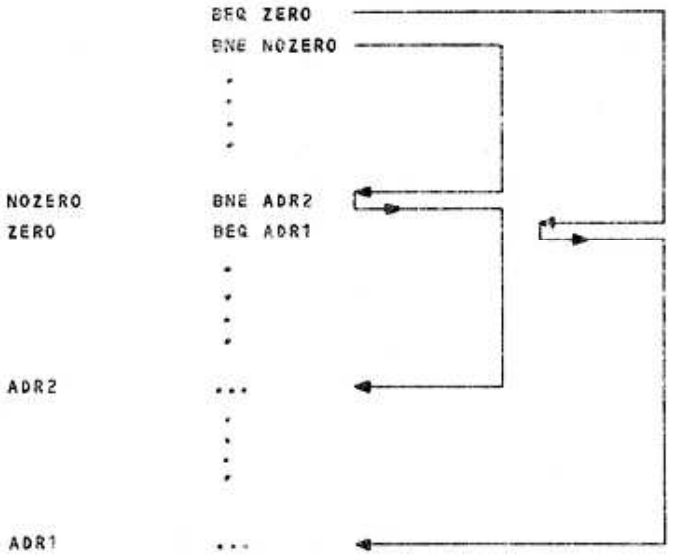
Neovlivňuje žádné příznaky.

Formální zápis inst.	Adres.	Činnost
BNE op_{16}	REL	skok na adresu op_{16}, je-li $\langle Z \rangle = 0$

TAB. 28 Instrukce BNE

Příklad:

Realizujte podmíněné rozvětvení programu podle příznaku Zero jako v předcházejícím příkladu. Tedy chceme realizovat skok mimo rozsah relativního adresování /-128 +127 byte/. Chceme však zachovat přemístitelnost strojového kódu, což lze realizovat metodou ping-pong.



Instrukce BMI

Provádí skok na adresu uvedenou v operandu instrukce, jestliže obsah příznaku N je log. 1. Není-li podmínka splněna, pokračuje program následující instrukcí.

Neovlivňuje žádné příznaky.

Schéma:
skok, jestliže ⟨N⟩=1 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BMI op_{16}	REL	skok na adresu op_{16}, je-li ⟨N⟩=1

TAB. 29 Instrukce BMI

Instrukce BPL

Provádí skok na adresu uvedenou v operandu instrukce, jestliže obsah příznaku N je log. 0. Není-li podmínka splněna, pokračuje program následující instrukcí.

Neovlivňuje žádné příznaky.

Schéma:
skok, jestliže ⟨N⟩=0 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BPL op_{16}	REL	skok na adresu op_{16}, je-li ⟨N⟩=0

TAB. 30 Instrukce BPL

Instrukce BCS

Provádí skok na adresu uvedenou v operandu instrukce, jestliže obsah příznaku C je log. 1. Není-li podmínka splněna, pokračuje program následující instrukcí.

Neovlivňuje žádné příznaky.

Schéma:
skok, jestliže ⟨C⟩=1 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BCS op_{16}	REL	skok na adresu op_{16}, je-li ⟨C⟩=1

TAB. 31 Instrukce BCS

Instrukce BCC

Provádí skok na adresu uvedenou v operandu instrukce, jestliže obsah příznakového bitu C je log. 0. Není-li podmínka splněna, pokračuje program následující instrukcí za instrukcí BCC.

Neovlivňuje žádné příznaky.

Schéma:
skok, jestliže ⟨C⟩=0 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BCC op_{16}	REL	skok na adresu op_{16}, je-li ⟨C⟩=0

TAB. 32 Instrukce BCC

Instrukce BVS

Provádí skok na adresu uvedenou v operandu instrukce, jestliže obsah příznaku V je log. 1. Není-li podmínka splněna, pokračuje program následující instrukcí.

Neovlivňuje žádné příznaky.

Schéma:
skok, jestliže ⟨V⟩=1 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BVS op_{16}	REL	skok na adresu op_{16}, je-li ⟨V⟩=1

TAB. 33 Instrukce BVS

Instrukce BVC

Provádí skok na adresu uvedenou v operandu instrukce, jestliže obsah příznaku V je log. 0. Není-li podmínka splněna, pokračuje program následující instrukcí.

Nenastavuje žádné příznaky.

Schéma:
skok, jestliže ⟨V⟩=0 ovlivní: -

Formální zápis inst.	Adres.	Činnost
BVC op_{16}	REL	skok na adresu op_{16}, je-li ⟨V⟩=0

TAB. 34 Instrukce BVC

Příklad:

Nulujte zónu paměti od adresy $600 do $605 včetně, tj. 6 byte.

Řešení A:

```
BASE    EQU $600
POCET   EPZ 6
        LDA#0
        LDX#POCET-1
NULUJ   STA BASE,X
        DEX
        BPL NULUJ
```

Řešení B:

```
BASE    EQU $600
POCET   EPZ 6
        LDA#0
        LDX#POCET
NULUJ   STA BASE-1,X
        DEX
        BNE NULUJ
```

Řešení C:

```
BASE    EQU $600
POCET   EPZ 6
DOPLNEK EPZ $100-6
        LDA#0
        LDX#DOPLNEK
NULUJ   STA BASE-DOPLNEK,X
        INX
        BMI NULUJ
```

5.3 Aritmetické instrukce

Aritmetické instrukce zajišťují základní matematické úkony:

a/ matematický osmibitový součet a rozdíl /ADC, SBC/,

b/ inkrementaci a dekrementaci
/INC, INX, INY, DEC, DEX, DEY/,
c/ matematické posuvy
/ASL, LSR/.

Pro matematický osmibitový součet a rozdíl je třeba dát pozor na nastavený mód /příznak D/. Je-li ⟨D⟩=0, je nastaven standardní hexadecimální /binární/ mód. Je-li ⟨D⟩=1, je nastaven decimální mód /kód BCD/. Obsah příznaku D má vliv pouze na instrukce ADC a SBC. Všechny ostatní instrukce i vnitřní činnost mikroprocesoru /například výpočet efektivních adres/ probíhají ve standardním hexadecimálním /binárním/ módu.

Matematické instrukce jsou základními stavebními kameny pro realizaci vícebajtové aritmetiky v plovoucí čárce.

Instrukce matematického osmibitového součtu a rozdílu ovlivňují příznaky N, V, Z a C.

Instrukce inkrementace a dekrementace nastavují pouze příznaky N a Z.

Instrukce matematického posuvu ovlivňují příznaky N, Z a C.

Pozor na inverzní funkci příznaku C při vykonávání instrukce rozdílu SBC oproti mikroprocesoru I-8080, Z-80.

Instrukce ADC

Provádí součet obsahu akumulátoru s daty uloženými na efektivní adrese s obsahem příznakového bitu C. Efektivní adresa je určena podle použitého způsobu adresování. Výsledek je uložen do akumulátoru. Podle výsledku /resp. průběhu/ operace jsou nastaveny příznaky N, V, Z a C, ostatní zachovávají původní hodnotu.

Schéma:
⟨A⟩ + ⟨M⟩ + ⟨C⟩ → ⟨A⟩ ovlivní: ⟨N⟩, ⟨V⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
ADC#op_8	IMM	součet: ⟨A⟩ +op_8+ ⟨C⟩→⟨A⟩
ADC op_8	ZP	součet: ⟨A⟩ + ⟨op_8⟩ + ⟨C⟩→⟨A⟩
ADC op_8,X	ZP,X	součet: ⟨A⟩ + ⟨op_8+ ⟨X⟩⟩ + ⟨C⟩→⟨A⟩
ADC op_{16}	ABS	součet: ⟨A⟩ + ⟨op_{16}⟩+ ⟨C⟩→⟨A⟩
ADC op_{16},X	ABS,X	součet: ⟨A⟩ + ⟨op_{16}+ ⟨X⟩⟩ + ⟨C⟩→⟨A⟩
ADC op_{16},Y	ABS,Y	součet: ⟨A⟩ + ⟨op_{16}+ ⟨Y⟩⟩ + ⟨C⟩→⟨A⟩
ADC (op_8,X)	(IND,X)	součet: ⟨A⟩ + ⟨(⟨op_8+ ⟨X⟩⟩)⟩ + ⟨C⟩→⟨A⟩
ADC (op_8),Y	(IND),Y	součet: ⟨A⟩ + ⟨(⟨op_8⟩+ ⟨Y⟩⟩ + ⟨C⟩→⟨A⟩

TAB. 35 Instrukce ADC

Příklad:

Sledujte význam příznakových bitů na těchto příkladech:

```
Příklad A:
SCIT1     EPZ $FF
SCIT2     EPZ $01
          CLD          ; standard HEX-režim
          CLC          ; standard při sčítání
          LDA#SCIT1
          ADC#SCIT2
          BRK          ; v ATMAS-II vypíše obsahy re-
                         gistrů a příznakových bitů
```

SCIT1	+ SCIT2	+ ⟨C⟩	→⟨A⟩	ovlivňuje N	V	Z	C	přetečení z b_7	b_6
$FF	$01	0	$00	0	0	1	1	ano	ano
$80	$80	0	$00	0	1	1	1	ano	ne
$7F	$01	0	$80	1	1	0	0	ne	ano
$09	$01	0	$0A	0	0	0	0	ne	ne

TAB. 36 Součet v hexadecimálním režimu podle PŘ. A

Příklad B:

```
SCIT1     EPZ $FF
SCIT2     EPZ $01
          CLD
          SEC
          LDA#SCIT1
          ADC#SCIT2
          BRK
```

SCIT1	+ SCIT2	+ ⟨C⟩	→⟨A⟩	ovlivňuje N	V	Z	C	přetečení z b_7	b_6
$FF	$01	1	$01	0	0	0	1	ano	ano
$FF	$00	1	$00	0	0	1	1	ano	ano

TAB. 37 Součet v hex. režimu podle PŘ. B

Příklad C:

```
SCIT1     EPZ $09
SCIT2     EPZ $01
          SED
          CLC
          LDA#SCIT1
```

```
ADC#SCIT2
BRK
```

SCIT1 + SCIT2 + ⟨C⟩→⟨A⟩				ovlivňuje N	V	Z	C
$09	$01	0	$10	0	0	0	0
$79	$01	0	$80	1	1	0	0
$89	$01	0	$90	1	0	0	0
$99	$01	0	$00	1	0	0	1

TAB. 38 Součet v decimálním režimu podle PŘ. C

Instrukce SBC

Provádí rozdíl: od obsahu akumulátoru odečte obsah efektivní adresy a inverzní hodnotu příznakového bitu C. Výsledek je uložen do akumulátoru. Podle výsledku /resp. průběhu / operace jsou nastaveny příznaky N /Negative/, V /Overflow/, Z /Zero/ a C /Carry/, ostatní zachovávají původní hodnotu.

Schéma:
⟨A⟩ - ⟨M⟩ - ⟨$\overline{C}$⟩→⟨A⟩ ovlivní: ⟨N⟩, ⟨V⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
SBC #op_8	IMM	rozdíl: $\langle A\rangle - op_8 - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC op_8	ZP	rozdíl: $\langle A\rangle - \langle op_8\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC op_8,X	ZP,X	rozdíl: $\langle A\rangle - \langle op_8 + \langle X\rangle\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC op_{16}	ABS	rozdíl: $\langle A\rangle - \langle op_{16}\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC op_{16},X	ABS,X	rozdíl: $\langle A\rangle - \langle op_{16} + \langle X\rangle\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC op_{16},Y	ABS,Y	rozdíl: $\langle A\rangle - \langle op_{16} + \langle Y\rangle\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC (op_8,X)	(IND,X)	rozdíl: $\langle A\rangle - \langle (op_8 + \langle X\rangle)\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$
SBC (op_8),Y	(IND),Y	rozdíl: $\langle A\rangle - \langle (op_8) + \langle Y\rangle\rangle - \overline{\langle C\rangle} \rightarrow \langle A\rangle$

TAB. 39 Instrukce SBC

Příklady:

1/ Sledujte nastavování příznakových bitů na těchto modifikacích odčítání:

Příklad A:

```
MENSENEC    EPZ $80
MENSITEL    EPZ $01
            CLD         ; standard HEX-režim
            SEC         ; standard při odčítání
            LDA # MENSENEC
            SBC # MENSITEL
            BRK
```

MENSENEC - MENSITEL - ⟨C⟩→⟨A⟩				ovlivňuje				podtečení do	
				N	V	Z	C	b_7	b_6
$80	$01	0	$7F	0	1	0	1	ne	ano
$00	$80	0	$80	1	1	0	0	ano	ne
$00	$01	0	$FF	1	0	0	0	ano	ano
$0A	$01	0	$09	0	0	0	1	ne	ne

TAB. 40 Rozdíl v hexadecimálním režimu podle PŘ. A

Příklad B:

```
MENSENEC    EPZ $10
MENSITEL    EPZ $01
            SED
            SEC
            LDA#MENSENEC
            SBC#MENSITEL
            BRK
```

MENSENEC - MENSITEL - ⟨C⟩→⟨A⟩				ovlivňuje			
				N	V	Z	C
$10	$01	0	$09	0	0	0	1
$00	$01	0	$99	1	0	0	0

TAB. 41 Rozdíl v decimálním režimu podle PŘ. B

2/ Realizujte podprogram pro rozdíl 6 byte podle schématu ⟨$D4.9⟩ - ⟨$F0.5⟩⟶⟨$D4.9⟩ s přenosem do významnějších byte. Na nižších adresách jsou uloženy méně významné byte. Pozn. symbol ⟨$D4.9⟩ znamená obsah buněk $D4 až $D9 včetně.

Řešení A: (úspornější)

```
POCET       EQU 6
POM         EQU POCET+1
            LDX#$100-POM
            SEC
LOOP        LDA $D4+POM,X
            SBC $F0+POM,X
            STA $D4+POM,X
            INX
            BNE LOOP
            RTS
```

Řešení B:

```
POCET       EQU 6
            LDX#0
            SEC
LOOP        LDA $D4,X
            SBC $F0,X
            STA $D4,X
            INX
            CMP#POCET
            BNE LOOP
            RTS
```

Instrukce INC

Provádí zvýšení obsahu buňky na efektivní adrese o jedničku /tzv. inkrementaci/. Výsledek je uložen zpět na efektivní adresu.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

⟨M⟩ +1 → ⟨M⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
INC op_8	ZP	inkrementace: $\langle op_8 \rangle + 1 \rightarrow \langle op_8 \rangle$
INC op_8,X	ZP,X	inkrementace: $\langle op_8 + \langle X \rangle \rangle + 1 \rightarrow \langle op_8 + \langle X \rangle \rangle$
INC op_{16}	ABS	inkrementace: $\langle op_{16} \rangle + 1 \rightarrow \langle op_{16} \rangle$
INC op_{16},X	ABS,X	inkrementace: $\langle op_{16} + \langle X \rangle \rangle + 1 \rightarrow \langle op_{16} + \langle X \rangle \rangle$

TAB. 42 Instrukce INC

Příklad:

Realizujte podprogram 16-ti bitového čítače, kde $CC je nižší, $CD je vyšší byte čítače.

Řešení:

```
INC $CC
BNE *+4 ; --┐
INC $CD     ¦
RTS         <-┘
```

Instrukce INX

Provádí zvýšení obsahu registru X o jedničku (tzv. inkrementaci). Výsledek je uložen zpět do registru X.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

$\langle X \rangle + 1 \rightarrow \langle X \rangle$ ovlivní: $\langle N \rangle$, $\langle Z \rangle$

Formální zápis inst.	Adres.	Činnost
INX	IMPL	inkrementace: $\langle X \rangle + 1 \rightarrow \langle X \rangle$

TAB. 43 Instrukce INX

Příklad:

Vynulujte celou šestou stránku paměti.

Řešení:

```
        LDA#0
        TAX
LOOP    STA $600,X
        INX
        BNE LOOP
```

Instrukce INY

Provádí zvýšení obsahu registru Y o jedničku /tzv. inkrementace/. Výsledek je uložen zpět do registru Y.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

⟨Y⟩ +1→⟨Y⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
INY	IMPL	inkrementace: ⟨Y⟩ +1→⟨Y⟩

TAB. 44 Instrukce INY

Instrukce DEC

Provádí zmenšení obsahu efektivní adresy o jedničku /tzv. dekrementaci/. Výsledek je uložen zpět na efektivní adresu.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

⟨M⟩ -1→⟨M⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
DEC op_8	ZP	dekrementace: $\langle op_8 \rangle - 1 \rightarrow \langle op_8 \rangle$
DEC op_8,X	ZP,X	dekrementace: $\langle op_8 + \langle X \rangle \rangle - 1 \rightarrow \langle op_8 + X \rangle \rangle$
DEC op_{16}	ABS	dekrementace: $\langle op_{16} \rangle - 1 \rightarrow \langle op_{16} \rangle$
DEC op_{16},X	ABS,X	dekrementace: $\langle op_{16} + \langle X \rangle \rangle - 1 \rightarrow \langle op_{16} + \langle X \rangle \rangle$

TAB. 45 Instrukce DEC

Příklad:

Realizujte podprogram pro 16-ti bitový odčitač /timer/ buněk TIMR s následující výstupní indikací:

⟨A⟩ ... 0 ... timer je prázdný - nedošlo k odečítání

⟨A⟩ ... 1 ... timer byl právě vynulován - byla odečtena poslední hodnota

⟨A⟩ ... 2 ... timer je plný - došlo k odečtení jednotky

Řešení:

```
TIMR      EQU $6FE          ; adresa nižšího byte odčitače
ODCITAC   LDY TIMR
          BNE ODCITLO
          LDY TIMR+1
          BEQ A0
          DEC TIMR+1        ; odečtení Hi byte
ODCITLO   DEC TIMR          ; odečtení Lo byte
          BNE A2
          LDY TIMR+1
          BNE A2
A1        LDA#1
          RTS
A0        LDA#0
          RTS
A2        LDA#2
          RTS
```

Instrukce DEX

Provádí zmenšení obsahu registru X o jedničku /tzv. dekrementaci/. Výsledek operace je uložen do registru X.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

⟨X⟩ -1→⟨X⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
DEX	IMPL	dekrementace: ⟨X⟩ -1→⟨X⟩

TAB. 46 Instrukce DEX

Příklad:

Přesuňte 16 byte z IOCB0 /tj. od adresy $0340/ na začátek šesté stránky paměti.

Řešení:

```
        LDX#15
OPET    LDA $340,X
        STA $600,X
        DEX
        BPL OPET
```

Instrukce DEY

Provádí zmenšení obsahu registru Y o jedničku /tzv. dekrementaci/. Výsledek je uložen zpět do registru Y.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

⟨Y⟩ -1→⟨Y⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
DEY	IMPL	dekrementace: ⟨Y⟩ -1→⟨Y⟩

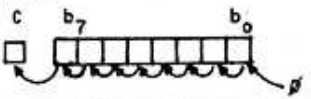
TAB. 47 Instrukce DEY

Instrukce ASL

Provádí levý posuv obsahu akumulátoru nebo efektivní adresy. Levý posuv je posuv obsahu jednotlivých bitů směrem vlevo o jeden bit. Nejvýznamnější bit se přesune do příznaku C, na nejméně významný bit je po posuvu dosazena log. 0. Výsledek operace je uložen zpět do akumulátoru nebo na efektivní adresu /podle způsobu adresování/.

Podle výsledku operace jsou nastaveny příznaky N, Z do C je dosazena log. 0.

Schéma:



ovlivní: ⟨N⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
ASL A	ACC	levý posuv ⟨A⟩
ASL op_8	ZP	levý posuv ⟨op_8⟩
ASL op_8,X	ZP,X	levý posuv ⟨op_8+⟨X⟩⟩
ASL op_{16}	ABS	levý posuv ⟨op_{16}⟩
ASL op_{16},X	ABS,X	levý posuv ⟨op_{16}+⟨X⟩⟩

TAB. 48 Instrukce ASL

Používá se k jednoduchým matematickým operacím.

Příklady:

1/ Akumulátor nabývá hodnot $00 až $1F, vynásobte jeho obsah osmi.

Řešení:

```
        ASL A           ; 2*⟨A⟩→⟨A⟩
        ASL A           ; 4*⟨A⟩→⟨A⟩
        ASL A           ; 8*⟨A⟩→⟨A⟩
```

2/ Vynásobte obsah buňky $CE třemi.

Řešení:

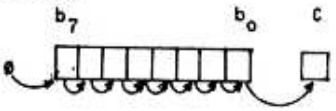
```
        LDA $CE
        ASL A           ; nebo ASL $CE
        CLC
        ADC $CE
        STA $CE
```

Instrukce LSR

Provádí pravý posuv obsahu akumulátoru nebo efektivní adresy. Pravý posuv je posuv obsahu jednotlivých bitů o jeden bit směrem vpravo. Nejméně významný bit b_0 přesunut do příznaku C, do nejvýznamnějšího bitu je po posuvu dosazena log. 0. Výsledek operace je uložen do akumulátoru nebo na efektivní adresu /podle způsobu adresování/.

Podle výsledku operace jsou nastaveny příznaky Z a C, příznak N je nulován.

Schéma:



ovlivní: ⟨N⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
LSR A	ACC	pravý posuv ⟨A⟩
LSR op_8	ZP	pravý posuv ⟨op_8⟩
LSR op_8,X	ZP,X	pravý posuv ⟨op_8+⟨X⟩⟩
LSR op_{16}	ABS	pravý posuv ⟨op_{16}⟩
LSR op_{16},X	ABS,X	pravý posuv ⟨op_{16}+⟨X⟩⟩

TAB. 49 Instrukce LSR

Příklad:

Dělte celočíselně obsah buňky $CD čtyřmi, zbytek dělení zanedbejte.

Řešení:

```
LSR $CD
LSR $CD
```

5.4 Logické instrukce

Logické instrukce mají velmi široké uplatnění. V jejich používání se často projeví kvality programátora.

Můžeme je rozdělit na:

- instrukce logických operací
  /AND, ORA, EOR/,
- instrukce rotací
  /ROL, ROR/,
- instrukce přímého ovládání příznaků
  /CLC, SEC, CLD, SED, CLI, SEI, CLV/,
- instrukce komparací
  /CMP, CPX, CPY/,
- instrukcí bitových operací
  /BIT/.

Instrukce logických operací ovlivňují příznaky N a Z.

Instrukce levé i pravé rotace ovlivňují příznaky N, Z a C.

Instrukce přímého ovládání příznaků nastavují vždy pouze příslušný příznakový bit, který je dán třetím písmenem v názvu instrukce. První dvě písmena v názvu určují hodnotu, na kterou bude příznakový bit nastaven. SE. /set/ znamená nastavení na log. 1, CL. /clear/ na log. 0.

Instrukce komparací ovlivňují příznaky N, Z a C. Pozor! Příznak C je po komparaci nastaven opačně než u I-8080, Z-80, podobně jako tomu bylo u instrukce rozdílu SBC.

Instrukce bitové operace ovlivňuje příznak Z; příznaky N a V jsou nastaveny podle bitu b_7, b_6 dat z efektivní adresy.

Instrukce AND

Provádí logický součin /bit po bitu/ obsahu akumulátoru s daty uloženými na efektivní adrese. Výsledek je uložen do akumulátoru.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

$\langle A \rangle .AND. \langle M \rangle \rightarrow \langle A \rangle$ ovlivní: $\langle N \rangle$, $\langle Z \rangle$

Formální zápis inst.	Adres.	Činnost
AND #op_8	IMM	log. součin: $\langle A \rangle .AND. op_8 \rightarrow \langle A \rangle$
AND op_8	ZP	log. součin: $\langle A \rangle .AND. \langle op_8 \rangle \rightarrow \langle A \rangle$
AND op_8,X	ZP,X	log. součin: $\langle A \rangle .AND. \langle op_8 + \langle X \rangle \rangle \rightarrow \langle A \rangle$
AND op_{16}	ABS	log. součin: $\langle A \rangle .AND. \langle op_{16} \rangle \rightarrow \langle A \rangle$
AND op_{16},X	ABS,X	log. součin: $\langle A \rangle .AND. \langle op_{16} + \langle X \rangle \rangle \rightarrow \langle A \rangle$
AND op_{16},Y	ABS,Y	log. součin: $\langle A \rangle .AND. \langle op_{16} + \langle Y \rangle \rangle \rightarrow \langle A \rangle$
AND (op_8,X)	(IND,X)	log. součin: $\langle A \rangle .AND. \langle \langle op_8 + \langle X \rangle \rangle \rangle \rightarrow \langle A \rangle$
AND (op_8),Y	(IND),Y	log. součin: $\langle A \rangle .AND. \langle \langle op_8 \rangle + \langle Y \rangle \rangle \rightarrow \langle A \rangle$

TAB. 50 Instrukce AND

Příklady:

1/ Nastavte bit b_7 v akumulátoru na log. 0. Hodnoty ostatních bitů zachovejte.

Řešení A:

```
AND #%01111111
```

Řešení B:

```
AND #$7F
```

2/ Rozdělte hodnotu akumulátoru na dva půlbajty. Méně významné čtyři bity uložte do registru X, významnější čtyři bity do registru Y. (Například: je-li ⟨A⟩=$4B, pak $0B→⟨X⟩, $40→⟨Y⟩.)

Řešení:

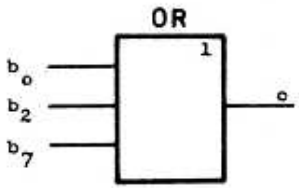
```
TAY
AND #$0F        ; maska Lo
TAX
TYA
AND #$F0        ; maska Hi
TAY
```

3/ Rozdělte hodnotu akumulátoru na dva půlbajty. Méně významné čtyři bity uložte do registru X, významnější čtyři bity uložte na pozici méně významné části registru Y. (Například: je-li ⟨A⟩=$4B, pak $0B→⟨X⟩ a $04→⟨Y⟩.)

Řešení:

```
TAY             ; uchování původní hodnoty A
AND #$0F        ; maska na méně význ. část
TAX             ; Lo - půlbajt
TYA             ; výběr původní hodnoty
LSR
LSR
LSR
LSR
TAY             ; Hi - půlbajt
```

4/ Realizujte funkci logického součtu OR na maskou vybraných bitech akumulátoru. Logickou hodnotu výsledku uložte do příznaku C. Například realizujte funkci:
$\langle C \rangle = \langle b_0 \rangle$ OR $\langle b_2 \rangle$ OR $\langle b_7 \rangle$



obr. 30 - Funkce OR

Řešení A:

```
MASKA     EPZ %10000101
          AND#MASKA
          CLC
          ADC#$FF
```

Řešení B:

```
MASKA     EPZ %10000101
          AND#MASKA
          SEC
          SBC#1
```

Instrukce ORA

Provádí logický součet /bit po bitu/ obsahu akumulátoru s daty uloženými na efektivní adrese. Výsledek je uložen do akumulátoru.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:

⟨A⟩.OR.⟨M⟩→⟨A⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
ORA #op_8	IMM	log. součet: ⟨A⟩.OR.op_8→⟨A⟩
ORA op_8	ZP	log. součet: ⟨A⟩.OR. op_8→⟨A⟩
ORA op_8,X	ZP,X	log. součet: ⟨A⟩.OR.⟨op_8+⟨X⟩⟩→⟨A⟩
ORA op_{16}	ABS	log. součet: ⟨A⟩.OR.⟨op_{16}⟩→⟨A⟩
ORA op_{16},X	ABS,X	log. součet: ⟨A⟩.OR.⟨op_{16}+⟨X⟩⟩→⟨A⟩
ORA op_{16},Y	ABS,Y	log. součet: ⟨A⟩.OR.⟨op_{16}+⟨Y⟩⟩→⟨A⟩
ORA (op_8,X)	(IND,X)	log. součet: ⟨A⟩.OR.⟨(⟨op_8+⟨X⟩⟩)⟩→⟨A⟩
ORA (op_8),Y	(IND),Y	log. součet: ⟨A⟩.OR.⟨(⟨op_8⟩+⟨Y⟩⟩→⟨A⟩

TAB. 51 Instrukce ORA

Příklady:

1/ Nastavte bit b_7 v akumulátoru na log. 1. Hodnoty ostatních bitů zachovejte.

Řešení A:

```
ORA #%10000000
```

Řešení B:

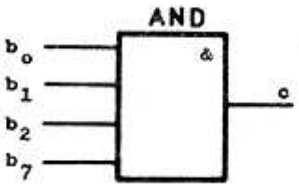
```
ORA #$80
```

Řešení C:

```
ORA #128
```

2/ Realizujte funkci logického součinu AND na maskou vybraných bitech akumulátoru. Logickou hodnotu výsledku uložte do příznaku C. Například realizujte:

⟨C⟩=⟨b_0⟩ AND ⟨b_1⟩ AND ⟨b_2⟩ AND ⟨b_7⟩



obr. 31 - Funkce AND

```
MASKAAND    EPZ %10000111
NEGMASKA    EPZ $FF-MASKAAND  ; negativní maska
            ORA#NEGMASKA
            CLC
            ADC#1
```

Instrukce EOR

Provádí nonekvivalenci /exklusivní logický součet bit po bitu/ obsahu akumulátoru s daty uloženými na efektivní adrese. Výsledek je uložen do ekumulátoru.

Podle výsledku operace jsou nastaveny příznaky N a Z.

Schéma:
⟨A⟩.EOR. ⟨M⟩→⟨A⟩ ovlivní: ⟨N⟩, ⟨Z⟩

Formální zápis inst.	Adres.	Činnost
EOR#op_8	IMM	nonekvivalence: ⟨A⟩.EOR.op_8→⟨A⟩
EOR op_8	ZP	nonekvivalence: ⟨A⟩.EOR.⟨op_8⟩→⟨A⟩
EOR op_8,X	ZP,X	nonekvivalence: ⟨A⟩.EOR.⟨op_8+⟨X⟩⟩→⟨A⟩
EOR op_{16}	ABS	nonekvivalence: ⟨A⟩.EOR.⟨op_{16}⟩→⟨A
EOR op_{16},X	ABS,X	nonekvivalence: ⟨A⟩.EOR.⟨op_{16}+⟨X⟩⟩→⟨A⟩
EOR op_{16},Y	ABS,Y	nonekvivalence: ⟨A⟩.EOR.⟨op_{16}+⟨Y⟩⟩→⟨A⟩
EOR (op_8,X)	(IND,X)	nonekvivalence: ⟨A⟩.EOR.⟨(op_8+⟨X⟩)⟩→⟨A⟩
EOR (op_8),Y	(IND),Y	nonekvivalence: ⟨A⟩.EOR.⟨(op_8)+⟨Y⟩⟩→⟨A⟩

TAB. 52 Instrukce EOR

Příklady:

1/ Negujte obsah akumulátoru.
Například, je-li ⟨A⟩=%10011100, pak po negaci ⟨A⟩=%01100011.
Řešení:

```
        EOR#$FF
```

2/ Negujte bity b_0 a b_2 buňky $CD. Ostatní bity zachovejte.
Řešení A:

```
MASKA   EPZ %00000101
        LDA#MASKA
        EOR $CD
        STA $CD
```

Řešení B:

```
MASKA   EPZ %00000101
        LDA $CD
        EOR#MASKA
        STA $CD
```

3/ Proveďte smíšené nastavení bitů akumulátoru podle předpisu:

bity b_0, b_1 .... nastavte na log. 1
b_2, b_3 .... nulujte
b_4, b_5 .... negujte
b_6, b_7 .... zachovejte

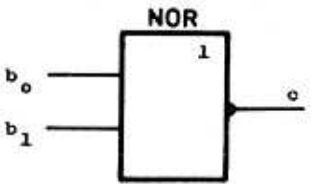
Řešení A:

```
ORA #%00001111  ; maska 1
EOR #%0011.1100 ; maska 2
```

Řešení B:

```
AND #%11110000  ; maska 1
EOR #%00110011  ; maska 2
```

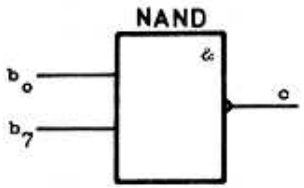
4/ Realizujte negativní logický součet NOR na maskou vybraných bitech akumulátoru. Logickou hodnotu výsledku uložte do příznaku C. Například realizujte:
$\langle C\rangle - \langle b_0\rangle$ NOR$\langle b_1\rangle$



obr. 32 - Funkce NOR

```
MASKANOR    EPZ %00000011
NEGMASK     EPZ $FF-MASKANOR  ; negativní maska
            EOR#$FF
            ORA#NEGMASK
            CLC
            ADC#1
```

5/ Realizujte negativní logický součin NAND na maskou vybraných bitech akumulátoru. Výsledek uložte do příznaku C. Například realizujte:
$\langle C \rangle = \langle b_7 \rangle$ NAND $\langle b_0 \rangle$



obr. 33 – Funkce NAND

```
MASKNAND    EPZ %10000001
            EOR#$FF
            AND#MASKNAND
            CLC
            ADC#$FF
```

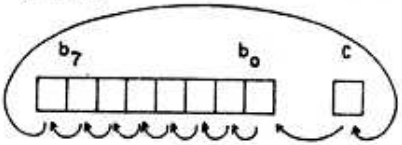
Instrukce ROL

Provádí levou rotaci jednotlivých bitů obsahu akumulátoru nebo efektivní adresy o jeden bit vlevo. Rotace se zúčastňuje příznak C, jeho obsah je přesunut do nejméně význam-

ného bitu b_0. Nejvýznamnější bit b_7 je přesunut do C. Výsledek je uložen do akumulátoru nebo na efektivní adresu podle použitého způsobu adresování.

Výsledek operace nastavuje příznaky N, Z a C.

Schéma:



ovlivní: ⟨N⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
ROL A	ACC	levá rotace ⟨A⟩
ROL op_8	ZP	levá rotace ⟨op_8⟩
ROL op_8,X	ZP,X	levá rotace ⟨op_8+⟨X⟩⟩
ROL op_{16}	ABS	levá rotace ⟨op_{16}⟩
ROL op_{16},X	ABS,X	levá rotace ⟨op_{16}+⟨X⟩⟩

TAB. 53 Instrukce ROL

Příklad:

Nastavte příznak C podle hodnoty nejvýznamnějšího bitu registru X.

Řešení:

```
TXA
ROL A
```

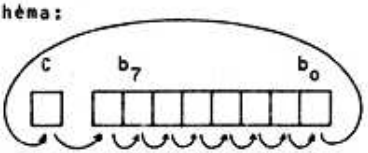
Instrukce ROR

Provádí pravou rotaci jednotlivých bitů obsahu akumulátoru nebo dat na efektivní adrese o jeden bit vpravo. Rotace se zúčastňuje příznak C, jeho obsah je přesunut do nejvýznamnějšího bitu b_7. Nejméně významný bit b_0 je přesunut

do C. Výsledek je uložen do akumulátoru nebo na efektivní adresu podle použitého způsobu adresování.

Podle výsledku operace jsou nastaveny příznaky N, Z, C.

Schéma:



ovlivní: ⟨N⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
ROR A	ACC	pravá rotace ⟨A⟩
ROR op_8	ZP	pravá rotace ⟨op_8⟩
ROR op_8,X	ZP,X	pravá rotace ⟨op_8+⟨X⟩⟩
ROR op_{16}	ABS	pravá rotace ⟨op_{16}⟩
ROR op_{16},X	ABS,X	pravá rotace ⟨op_{16}+⟨X⟩⟩

TAB. 54 Instrukce ROR

Příklad:

Nastavte příznak C podle hodnoty bitu b_2 registru Y.

Řešení A:

```
        TYA
        ROR A
        ROR A
        ROR A
```

Řešení B:

```
        TYA
        AND %00000100      ; maska na b2
        CLC
        ADC#$FF
```

Instrukce CLC

Provádí nastavení příznaku C /Carry/ na log. 0.

Schéma:

0→⟨C⟩ ovlivní: ⟨C⟩

Formální zápis inst.	Adres.	Činnost
CLC	IMPL	0→⟨C⟩

TAB. 55 Instrukce CLC

Instrukce SEC

Provádí nastavení příznaku C /Carry/ na log. 1.

Schéma:

1→⟨C⟩ ovlivní: ⟨C⟩

Formální zápis inst.	Adres.	Činnost
SEC	IMPL	1→⟨C⟩

TAB. 56 Instrukce SEC

Instrukce CLD

Provádí nastavení příznaku D /Decimal/ na log. 0, tj. nastavuje standardní binární /hexadecimální/ režim pro matematické operace.

Schéma:

0→⟨D⟩ ovlivní: ⟨D⟩

Formální zápis inst.	Adres.	Činnost
CLD	IMPL	0⟶⟨D⟩ /nastavení standardního binárního režimu/

TAB. 57 Instrukce CLD

Instrukce SED

Provádí nastavení příznaku D /Decimal/ na log. 1, tj. nastavuje dekadický režim pro matematické operace.

Schéma:

1⟶⟨D⟩ ovlivní: ⟨D⟩

Formální zápis inst.	Adres.	Činnost
SED	IMPL	1⟶⟨D⟩ /nastavení dekadického režimu/

TAB. 58 Instrukce SED

Pozn.: Dekadický režim je nutno používat opatrně. Nezruší-li se, může to vést k havárii systému.

Instrukce CLI

Provádí nastavení příznaku I /Interrupt/ na log. 0, tj. povoluje zpracování žádosti o maskovatelné přerušení typu $\overline{IRQ}$.

Schéma:

0⟶⟨I⟩ ovlivní: ⟨I⟩

Formální zápis inst.	Adres.	Činnost
CLI	IMPL	0⟶⟨I⟩ /povolení zpracování přerušení od IRQ/

TAB. 59 Instrukce CLI

Proveďte opačnou operaci než v následujícím příkladu. Povolte zpět přerušení od všech zdrojů přerušení /tj. $\overline{\text{IRQ}}$, $\overline{\text{NMI}}$/, povolte DMA.

```
LDA#$C0
STA $D40E          ; povolí NMI /VBI, DLI/
CLI                ; povolí IRQ
```

Tento postup předpokládá, že jsme neměnili původní hodnotu masky přerušení $\overline{\text{IRQ}}$ pro obvod POKEY na adrese $10 a původní hodnotu proměnné pro řízení DMA na adrese $22F.

Jelikož při prvním zpracování VBI dojde ke kopírování obsahu buňky $10 do sledujícího hardwarového registru $D20E a kopírování obsahu buňky $22F do hardwarového registru $D400. To zapříčiní nastavení interní masky pro obvod POKEY a povolení zpracování DMA pro obvod ANTIC.

Pokud jsme provedli zásahy v datových strukturách, je vhodnější použít např. postup:

```
LDA#%11000000   ; nastavení MASKY pro IRQ POKEY
STA $10
STA $D20E       ; sledující registr
LDA#%11000000   ; maska pro NMI ANTIC
STA $D40E       ; povolení NMI
LDA#%00100010   ; maska pro řízení DMA ANTIC
STA $2FF
STA $D400       ; sledující registr, povolení DMA
CLI             ; povolení IRQ
```

Instrukce SEI

Provádí nastavení příznaku I /Interrupt/ na log. 1, tj. zakazuje zpracování přerušení od $\overline{IRQ}$.

Schéma:
$1 \longrightarrow \langle I \rangle$ ovlivní: $\langle I \rangle$

Formální zápis inst.	Adres.	Činnost
SEI	IMPL	$1 \longrightarrow \langle I \rangle$ /zákaz zpracování přerušení od $\overline{IRQ}$/

TAB. 60 Instrukce SEI

Příklad:

Zakažte přerušení od všech systémových zdrojů přerušení /tj. $\overline{NMI}$, $\overline{IRQ}$/, zakažte činnost mikroprocesoru ANTIC /tj. potlačte DMA - přímý přístup do paměti/.

Řešení:

```
        SEI             ; zákaz přerušení všech zdrojů
                          IRQ /POKEY, PIA/
        LDA#0
        STA $D40E       ; zákaz přerušení NMI /ANTIC:
                          VBI, DLI/
        STA $D400       ; zákaz HALT přímého přístupu
                          do paměti DMA /ANTIC/
```

Tento postup se používá, chceme-li:

- provádět časovou synchronizaci na základě času spotřebovaného instrukcemi /např. synchronizace čtení a zápisu u programu TURBO 2000/,
- aby náš program byl maximálně rychlý, /zrychlí se cca 2x/,
- mít úplnou nadvládu nad operačním systémem a operační pamětí,

- provádět hazardní operace s operačním systémem /například kopirování programů operačního systému z ROM na stínovou paměť RAM, vystavění RAMDISKu na stínové paměti RAM, kontrola stínové paměti RAM, viz program RAMTEST V1.0 ve "Sbírce"/.

Instrukce CLV

Provádí nastavení příznaku V /Overflow/ na log. 0.

Schéma:

0 ⟶ ⟨V⟩ ovlivní: ⟨V⟩

Formální zápis inst.	Adres.	Činnost
CLV	IMPL	0 ⟶ ⟨V⟩

TAB. 61 Instrukce CLV

Pozn.: Příznak V nelze přímo nastavit na log. 1. Nastavení je třeba provést některou jinou vhodnou instrukcí /například pomocí PLP/ nebo hardwarovým signálem S.O. /Set Overflow/.

Příklad:

Nastavte programově příznak V na hodnotu log. 1.

Řešení 1:

```
        CLD             ; hex. režim
        LDA#$40
        ADC#1           ; ⟨C⟩ lib.
```

Řešení 2:

```
        PHP
        PLA
        ORA#%01000000
        PHA
        PLP
```

Instrukce CMP

Provádí komparaci /porovnání/ obsahu akumulátoru s daty na efektivní adrese. Porovnání je provedeno tak, že od obsahu akumulátoru se odečte obsah efektivní adresy. Výsledek komparace se nikam neukládá, pouze ovlivňuje příznaky N, Z a C. Obsah akumulátoru i efektivní adresy se zachovává.

Schéma:
⟨A⟩ - ⟨M⟩ ovlivní: ⟨N⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
CMP#op_8	IMM	komparace: ⟨A⟩-op_8
CMP op_8	ZP	komparace: ⟨A⟩-⟨op_8⟩
CMP op_8,X	ZP,X	komparace: ⟨A⟩-⟨op_8+⟨X⟩⟩
CMP op_{16}	ABS	komparace: ⟨A⟩-⟨op_{16}⟩
CMP op_{16},X	ABS,X	komparace: ⟨A⟩-⟨op_{16}+⟨X⟩⟩
CMP op_{16},Y	ABS,Y	komparace: ⟨A⟩-⟨op_{16}+⟨Y⟩⟩
CMP (op_8,X)	(IND,X)	komparace: ⟨A⟩-⟨⟨op_8+⟨X⟩⟩⟩
CMP (op_8),Y	(IND),Y	komparace: ⟨A⟩-⟨⟨op_8⟩+⟨Y⟩⟩

TAB. 62 Instrukce CMP

Používá se:
- pro testování,
- pro následné větvení,
- pro ovlivnění příznaků.

Princip nastavení příznaků je stejný jako u instrukce SBC.
Instrukce CMP na rozdíl od SBC:
- neměni obsah akumulátoru,
- příznak C nevstupuje do operace,
- příznak V není ovlivněn.

Příznak C je nastaven /stejně jako SBC/ inverzním způsobem než u ekvivalentní instrukce CMP u mikroprocesoru I-8080, Z-80.

Příklad 1:

Sledujte použití instrukce CMP pro následné větvení programu.

```
LDA $CC
CMP $CD          ; ovlivní ⟨Z⟩, ⟨N⟩, ⟨C⟩
B.. ADRESA
```

	skok je proveden, je-li:	
BEQ	⟨$CC⟩ = ⟨$CD⟩	⟨Z⟩=1
BNE	⟨$CC⟩ ≠ ⟨$CD⟩	⟨Z⟩=0
BPL	0 ≤ [⟨$CC⟩ - ⟨$CD⟩] ≤ $7F	⟨N⟩=0
BMI	$80 ≤ [⟨$CC⟩ - ⟨$CD⟩] ≤ $FF	⟨N⟩=1
BCC	⟨$CC⟩ < ⟨$CD⟩	⟨C⟩=0
BCS	⟨$CC⟩ ≥ ⟨$CD⟩	⟨C⟩=1

TAB. 63 Podmínky k provedení skoku

Příklad 2:

Jestliže obsah akumulátoru má hodnotu 5, skoč na návěští SHODA, má-li jinou hodnotu, pokračuj.

Řešení:

```
CMP#5
BEQ SHODA        ; skok, jestliže ⟨A⟩=5
...              ; pokračuje, jestliže ⟨A⟩≠5
```

Příklad 3:

Je-li obsah akumulátoru menší než obsah buňky paměti $8000, převeď řízení na návěští CY0, jinak převeď řízení na CY1.

Řešení:

```
CMP $8000
BCC CY0          ; skok na CY0, je-li ⟨A⟩ <$8000
BCS CY1          ; skok na CY1, je-li ⟨A⟩≥ $8000
```

Instrukce CPX

Provádí komparaci /porovnání/ obsahu registru X s daty na efektivní adrese. Porovnání se provádí odečtením dat z efektivní adresy od obsahu registru X. Výsledek se nikam neukládá, pouze ovlivňuje příznaky N, Z a C. Obsah registru X i efektivní adresy zůstává zachován.

Schéma:
⟨X⟩ - ⟨M⟩ ovlivní: ⟨N⟩, ⟨Z⟩, ⟨C⟩

Formální zápis inst.	Adres.	Činnost
CPX#op_8	IMM	komparace: ⟨X⟩-op_8
CPX op_8	ZP	komparace: ⟨X⟩-⟨op_8⟩
CPX op_{16}	ABS	komparace: ⟨X⟩-⟨op_{16}⟩

TAB. 64 Instrukce CPX

Příklad:

Nechť na adresách $CC, $CD je uložena hodnota 16-ti bitového čítače /ukazatele/, na adresách $CE, $CF je uložena referenční adresa. Na nižších adresách jsou méně významné byte. Převeďte řízení na adresu ZPET,

a/ pokud hodnota ukazatele je menší než hodnota referenční adresy,

Řešení a/

```
        LDX $CC
        LDA $CD
        CPX $CE
        SBC $CF
        BCC ZPET        ; skok, je-li ⟨$CC.D⟩ <⟨$CE.F⟩
```

b/ pokud hodnota ukazatele je rovna nebo větší než hodnota referenční adresy,

Řešení b/

```
        LDX $CC
        LDA $CD
        CPX $CE
        SBC $CF
        BCS ZPET        ; skok, je-li ($CC.D) ≥ ($CE.F)
```

c/ pokud hodnota ukazatele je menší nebo rovna hodnotě referenční adresy,

Řešení c/

```
        LDX $CE
        LDA $CF
        CPX $CC
        SBC $CD
        BCS ZPET        ; skok, je-li ($CC.D) ≤ ($CE.F)
```

d/ pokud hodnota ukazatele je větší než hodnota referenční adresy,

Řešení d/

```
        LDX $CE
        LDA $CF
        CPX $CC
        SBC $CD
        BCC ZPET        ; skok, je-li ($CC.D) > ($CE.F)
```

Instrukce CPY

Provádí komparaci obsahu registru Y s daty na efektivní adrese. Porovnání se provádí odečtením dat z efektivní adresy od obsahu registru Y. Výsledek se nikam neukládá, pouze ovlivňuje příznaky N, Z a C. Obsah registru Y i efektivní adresy zůstává zachován.

Schéma:
(Y) - (M) ovlivní: (N), (Z), (C)

Formální zápis inst.	Adres.	Činnost
CPY#op_8	IMM	komparace: $\langle Y \rangle - op_8$
CPY op_8	ZP	komparace: $\langle Y \rangle - \langle op_8 \rangle$
CPY op_{16}	ABS	komparace: $\langle Y \rangle - \langle op_{16} \rangle$

TAB. 65 Instrukce CPY

Instrukce BIT

Provádí zdánlivý logický součin obsahu akumulátoru s daty na efektivní adrese. Výsledek však není nikam uložen, pouze ovlivní příznak Z. Příznak N je nastaven na hodnotu nejvýznamnějšího bitu b_7 dat z efektivní adresy. Příznak V je nastaven na hodnotu b_6 dat z efektivní adresy. Obsah akumulátoru i efektivní adresy zůstává zachován.

Schéma:
$\langle A \rangle$.AND.$\langle M \rangle$ ovlivní: $\langle Z \rangle$, $\langle b_7 \rangle \rightarrow \langle N \rangle$, $\langle b_6 \rangle \rightarrow \langle V \rangle$

Formální zápis inst.	Adres.	Činnost
BIT op_8	ZP	zdánlivý log. součin $\langle A \rangle$.AND.$\langle op_8 \rangle$
BIT op_{16}	ABS	zdánlivý log. součin $\langle A \rangle$.AND.$\langle op_{16} \rangle$

TAB. 66 Instrukce BIT

5.5 Instrukce NOP

Instrukce NOP neprovádí nic. Pouze zabírá 1 byte v paměti a spotřebuje 2 hodinové cykly. Neovlivňuje žádné příznaky a neměni obsah žádného registru ani buňky paměti.

Formální zápis inst.	Adres.	Činnost
NOP	IMPL	nic neprovádí

TAB. 67 Instrukce NOP

Používá se například v časových smyčkách, pro rezervování místa pro předpokládané vkládání instrukcí v budoucnosti, pro rezervování místa pro data, pro "přemazání" instrukcí na úrovni strojového kódu atd.

6. Pseudoinstrukce

Pseudoinstrukce slouží k řízení překladače během překladu. Některé pseudoinstrukce /například definice dat DFB, DFW/ se bezprostředně projeví v generovaném strojovém kódu. Jiné /například definice jmen EQU, EPZ, pseudoinstrukce pro řízení počítadla adres ORG/ se ve vlastním generovaném kódu projevují zprostředkovaně.

Pro jazyk symbolických adres existují různé překladače, které se liší v komfortu práce s nimi, poskytovanými možnostmi, ale i různou sadou pseudoinstrukcí a jejich formálním zápisem.

Dále popsané pseudoinstrukce jsou používány v překladači ATMAS-II. Přehled odlišností mezi zápisy pseudoinstrukcí u různých překladačů JSA pro mikropočítače ATARI poskytuje tabulka 2.

Ve zdrojovém textu se pseudoinstrukce zapisují stejně jako instrukce, tzn. jejich návěští se zapisují do pole návěští, pseudoinstrukce do pole typu operace, operandy do pole operandů a komentáře do pole komentářů.

Pseudoinstrukce ORG

Slouží k nastavení hodnoty aktuálního počítadla adres. Používá se pro určení logické a fyzické adresy překladu. Touto pseudoinstrukcí začíná zdrojový text programu.

Formální zápis pseudoinstrukce	Činnost
ORG log.adr.,fyz.adr.	určení logické a fyzické adresy překladu, jestliže se od sebe liší
ORG adr.	určení logické a fyzické adresy překladu, jestliže jsou totožné

TAB. 68 Pseudoinstrukce ORG

Fyzická adresa určuje místo, od kterého bude uložen přeložený strojový kód.

Logická adresa určuje místo, kam by měl být uložen přeložený strojový kód, jestliže jej na toto místo dočasně nelze uložit /například je tam zatím uložen překladač nebo jiný program/. Všechny absolutní vazby budou pak odvozeny od logické adresy. Před spuštěním programu ve strojovém kódu musí být program nejdříve zaveden z fyzické adresy na logickou.

Pro zápis operandů log.adr., fyz.adr. a adr. platí stejná pravidla jako pro zápis operandu op_{16}, může tam být matematický výraz, symboly atd.

Pseudoinstrukce ORG může být v programu použita vícekrát.

Je-li uvnitř zdrojového textu ORG *+N, kde N ke libovolné číslo ⟨0,65535⟩, dojde k rezervaci N byte paměti. Například pro proměnné, vyrovnávací buffery a pod. Je to náhrada za pseudoinstrukci rezervace místa DSN, kterou překladač ATMAS-II nemá.

Pseudoinstrukce EQU

Přiřazuje symbolu /jménu/ uvedenému v poli návěští konkrétní numerickou hodnotu výrazu uvedeného v poli operandu. Výraz může mít dvoubajtovou hodnotu /$0000 až $FFFF/.

Formální zápis pseudoinstrukce	Činnost
symbol EQU op_{16}	uvedenému symbolu přiřazuje hodnotu op_{16}

TAB. 69 Pseudoinstrukce EQU

Pseudoinstrukci EQU lze použít kdekoli v programu.

Příklad na použití pseudoinstrukce EQU

```
AKTUALPC EQU *
A1       EQU $340
HODNOTA  EQU (AKTUALPC+A1-$A1 ) /256*256
DELKA    EQU PEND-AKTUALPC
         LDY#DELKA:L
         LDX#DELKA:H
         LDA A1
         STA HODNOTA
PEND     EQU *
```

Pseudoinstrukce EPZ

Přiřazuje symbolu /jménu/ uvedenému v poli návěští konkrétní numerickou hodnotu výrazu uvedeného v poli operandu. Na rozdíl od EQU přiřazuje EPZ jménu pouze jednobajtovou hodnotu /$00 až $FF/.

Formální zápis pseudoinstrukce	Činnost
symbol EPZ op_8	uvedenému symbolu přiřazuje hodnotu op_8

TAB. 70 Pseudoinstrukce EPZ

Pseudoinstrukci EPZ je možno použít kdekoli v programu.

Pseudoinstrukce DFB

Definuje jednobajtové konstanty. Umožňuje vkládat data do generovaného strojového kódu.

Formální zápis pseudoinstrukce	Činnost
DFB op_8	definice jedné konstanty
DFB op_8,op_8...,op_8	definice více konstant

TAB. 71 Pseudoinstrukce DFB

Používá se k:
- vkládání konstant,
- rezervování místa pro proměnné,
- přímému vkládání operačních kódů nebo operandů instrukcí.

Příklad:

Vložení operačního kódu instrukce pomocí matematického přepínače.

```
VERZE   EPZ 0                              ; 0=MONITOR, 1=BASIC
START   DFB (1-VERZE)*$EA+VERZE*$68        ; 0...NOP,   1...PLA
        ...                                ; instrukce těla pprg.
        DFB VERZE*$60                      ; 0...BRK,   1...RTS
```

Příklad:

Rezervování místa v programu pro proměnnou.

```
        JMP *+4
        DFB 0                  ; REZERVE 1 BYTE
```

Pseudoinstrukce DFW

Definuje slovo, tj. dvoubajtovou konstantu. Méně významný byte je uložen jako první /na nižší adresů/, významnější jako druhý /na vyšší adresu/.

Formální zápis pseudoinstrukce	Činnost
DFW op_{16}	definice jednoho slova
DFW op_{16},op_{16}...,op_{16}	definice více slov

TAB. 72 Pseudoinstrukce DFW

Používá se k:
- vkládání dvoubajtových konstant,
- vkládání vektoru skoku,
- rezervování místa pro proměnné.

Příklad:

Definování vektorů skoků.

```
            JMP (ADRVEK)
            JMP (ADRVEK+2)
            JMP (ADRVEK+4)
            ...
ADRVEK      DFW $5000
            DFW $708
            DFW $E471
```

Pseudoinstrukce ASC

Umožňuje vkládat textové řetězce v ASCII kódu.

Formální zápis pseudoinstrukce	Činnost
ASC 'text' ASC "text"	vloží přímo ASCII hodnoty textu /liší se pouze formálním zápisem/
ASC /text/	vloží SCII znaky, ale nejvýznamnější bit je u všech znaků nastaven na log. 1 /přičte $80 ke všem znakům textu/
ASC \text\	vloží SCII znaky, ale u posledního znaku nastaví nejvýznamnější bit na log. 1 /přičte $80 k poslednímu znaku/

TAB. 73 Pseudoinstrukce ASC

Jak je patrno, stejný text je uložen různě podle použitého oddělovače.

Pseudoinstrukce ASC se používá pro vypisování zpráv na displej nebo tiskárnu.

Příklad:

Výpis zprávy "READY:?" na obrazovku.

```
JSR SMESSAGE
DFB $0D          ; nový řádek
ASC \READY:?\
```

Pozn.: Podprogram SMESSAGE je uveden ve "Sbírce".

Pseudoinstrukce OUT

Vyvolává třetí průchod překladače, který generuje zprávu o překladu /tzv. listing/. Listing může obsahovat:

a/ zdrojový text JSA, strojový kód, logické adresy, definice makroinstrukcí,

b/ tabulku použitých symbolů, jmen, návěští včetně lokálních proměnných makroinstrukcí a jejich skutečných hodnot,

c/ opis rozvoje maker.

Formální zápis pseudoinstrukce	Činnost
OUT L	výpis a) a c) na displej
OUT N	výpis b) na displej
OUT LM	výpis a) na displej
OUT LNM	výpis a), b) na displej
OUT LPn	výstup typu L na tiskárnu, n udává typ tiskárny: n=1 ... RS 232 C n=2 ... paralelní n=3 ... ELCOMP
OUT LNMPn	výstup typu LNM na tiskárnu, n udává typ tiskárny.

TAB. 74 Pseudoinstrukce OUT

Pseudoinstrukce OUT se zpravidla uvádí jako první ve zdrojovém textu. Může být použita opakovaně. Lze tedy měnit skladbu výpisu listingu. Nejčastěji se používá OUT LN.

Pseudoinstrukce MACRO

Pseudoinstrukce MACRO slouží k definování makroinstrukce. Makroinstrukce umožňují generovat opakující se části zdrojového textu, které se od sebe mohou mírně lišit. Za tím účelem je třeba nejdříve požadovanou část zdrojového textu definovat jako makroinstrukci. V definici makroinstrukce jsou proměnné položky nahrazeny formálními parametry.

Formálním parametrem může být pouze symbol /např. ALFA, B1 a pod./. Nezaměňovat se skutečným parametrem, který může být:

- číslo /decimální, hexadecimální, binární/,
- řetězec,

- symbol,
- aritmetický výraz ohraničený kulatými závorkami.

Jakmile byla makroinstrukce jednou definována, může být kdykoli volána svým jménem uvedeným v poli typu operace. V poli operandů se uvede seznam skutečných parametrů.

Formální zápis pseudoinstrukce	Činnost
jméno MACRO	definice makroinstrukce s názvem jméno bez formálních parametrů
jméno MACRO par_1	definice makroinstrukce s názvem jméno a jedním formálním parametrem par_1
jméno MACRO par_1,par_2,...., par_n	definice makroinstrukce s názvem jméno a s formálními parametry par_1 až par_n

TAB. 75 Pseudoinstrukce MACRO

V definici (tělu) makroinstrukce lze použít:
- instrukce JSA 6502,
- jiné, již definované makroinstrukce,
- definice nových makroinstrukcí /tzv. hnízdění/,
- formální parametry makroinstrukce /mají pouze lokální význam v rámci makroinstrukce/,
- globální jména /symboly, návěští, proměnné/, které mají význam v celém programu /stejné jméno nesmí být deklarováno dvakrát/,
- lokální jména, za nimiž bezprostředně následuje znak @; lokální jména mohou být definována pouze uvnitř makroinstrukce a jejich platnost je omezena pouze na definovanou makroinstrukci, takže v definici jiné makroinstrukce může být použito stejné lokální jméno.

Srovnání makroinstrukce s podprogramem:

- Podprogram i makroinstrukci používáme například tam, kde se opakuje stejná posloupnost instrukcí.
- V přeloženém programu se strojový kód podprogramu objeví pouze jedinkrát, zatímco tělo makroinstrukce /ve strojovém kódu/ bude vygenerováno všude, kde byla makroinstrukce volána.
- Použitím podprogramu se snižuje počet strojových instrukcí, použitím makroinstrukce nikoli.
- Makroinstrukce urychlují provedení programu tím, že nevyžadují instrukce volání podprogramu JSR a návratu z podprogramu RTS.
- Použitím makroinstrukce se zvýší přehlednost programu.
- Použitím podprogramů ve zdrojovém textu programu se vylučuje přemístitelnost programu na úrovni strojového kódu.

Pseudoinstrukce MEND

Ukončuje definici makroinstrukce. Musí následovat za poslední instrukcí či pseudoinstrukcí zdrojového textu makroinstrukce. Ke každé pseudoinstrukci MACRO náleží právě jedna pseudoinstrukce MEND. Mezi pseudoinstrukcemi MACRO a MEND je umístěno tělo makroinstrukce.

Formální zápis pseudoinstrukce	Činnost
MEND	ukončení definice makroinstrukce

TAB. 76 Pseudoinstrukce MEND

Volání makroinstrukce

Jakmile je jednou makroinstrukce definována, může být ve zdrojovém textu vícenásobně volána. Volání makroinstrukce spočívá v uvedení jména makroinstrukce v poli typu operace a příslušného seznamu skutečných parametrů v poli operandů. Během překladu je každé volání makroinstrukce nahrazeno tělem definice makroinstrukce, ve kterém jsou formální parametry nahrazeny skutečnými.

Makroinstrukce musí být definována ještě před svým voláním. Počet typ a pořadí skutečných parametrů musí odpovídat počtu, typu a pořadí formálních parametrů v definici makroinstrukce.

Formální zápis volání makroinstrukce	Činnost
jméno	volání makroinstrukce pod uvedeným jménem (bez parametrů)
jméno val	volání makroinstrukce pod uvedeným jménem s dosazením skutečného parametru val
jméno val_1,val_2,...val_n	volání makroinstrukce pod uvedeným jménem s dosazením skutečných parametrů val_1 až val_n

TAB. 77 Volání makroinstrukce

Skutečným parametrem může být:

- číslo /decimální, hexadecimální, binární/,
- řetězec,
- symbol /jméno, návěští/ globálního významu,
- aritmetický výraz s operátory + - x / ohraničený kulatými závorkami

Používání makroinstrukcí vychází ze zásad strukturovaného programování, což přináší tyto výhody:

- umožňuje podobně jako podprogram rozložit úlohu na dílčí části /moduly - makroinstrukce/,

- používání lokálních jmen vylučuje možnost konfliktu při použití stejných lokálních jmen v jiných makroinstrukcích,
- předávání informací pomocí parametrů zvyšuje bezpečnost a přehlednost,
- snižuje únavné přepisování stejných či podobných částí (zdrojových textů) programů a tím snižuje možnost výskytu chyb,
- chyby v makroinstrukci se odstraní pouze jednou opravou bez ohledu na to, kolikrát je makroinstrukce v programu použita, což urychluje ladění programu,
- makroinstrukce zvyšují přehlednost, čitelnost, opravovatelnost a modifikovatelnost programů.

Příklady definic a použití makroinstrukcí jsou uvedeny ve "Sbírce".

Příklad:

Definice makroinstrukce, která opakovaně tiskne vybraný ASCII znak na obrazovku.

ASCII znak, počet opakování tisku jsou vstupními parametry makroinstrukce.

```
OUTCH       EQU $F2B0       ; globální jm., rutina OS pro
                              XL/XE
* Definice makroinstrukce
MREPEAT     MACRO CHAR,POCET
CITAC @     EPZ $CE         ; lokální jm.
            LDA #CHAR       ; 1.parametr
            LDX #POCET      ; 2.parametr
            STX CITAC @
LOOP @      PHA             ; uchování CHAR
            JSR OUTCH
            PLA             ; výběr CHAR
            DEC CITAC @
            BNE LOOP @
            MEND
```

```
* Použití makroinstrukce
          ORG $5000
          LDA#$9B          ; nový řádek
          JSR OUTCH
          MREPEAT  '?,38
          MREPEAT $41,$10
          RTS
```

7. Makroasembler ATMAS-II

Překladač ATMAS-II /Atari Macro Asembler/ byl vyvinut pro osmibitové mikropočítače firmy ATARI postavené na bázi mikroprocesoru 6502 s minimální pamětí 48 KB RAM a lze ho tedy provozovat i na dovážených mikropočítačích ATARI 800XL, ATARI 130XE a ATARI 800XE.

ATMAS-II vychází z předcházejících verzí, na kterých se podílelo několik významných autorů /H. Ch. Wagner, W. Hofacker, Finzel/.

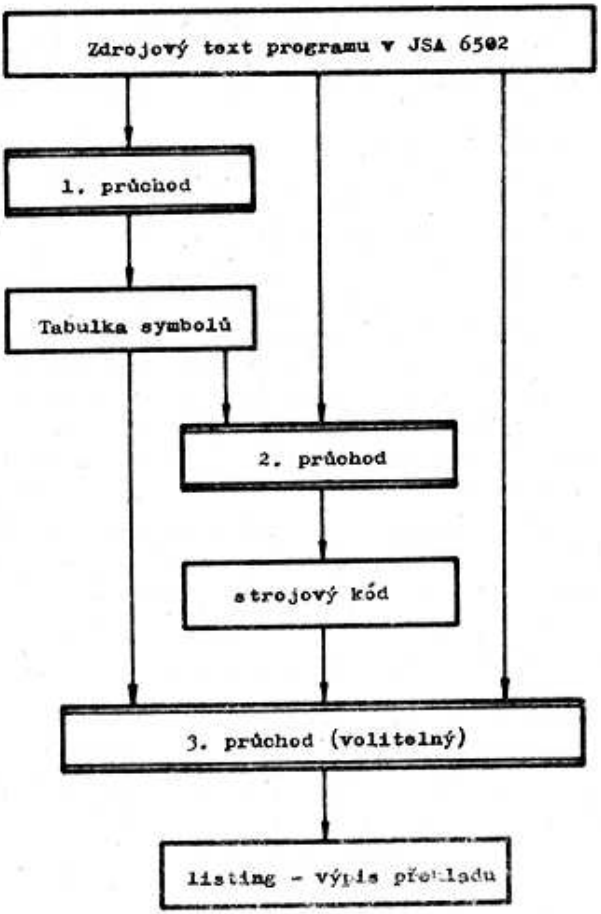
Charakteristika ATMAS-II

Makroasembler ATMAS-II lze stručně charakterizovat následovně:

- Je to dvouprůchodový, resp. tříprůchodový vlastní překladač JSA mikroprocesoru 6502.
- Kromě překladače obsahuje komfortní editor na pořizování a opravování zdrojového textu programu v JSA. /popsán v kap. 7.1/
- Obsahuje monitor se základními funkcemi. /popsán v kap. 7.6/
- Nemá zabudované ladicí prostředky. Má pouze možnost nastavení tzv. ladicího bodu /Break Point/. Pozn.: Podle zahraničních pramenů existuje kompatibilní ladicí modul, který může být v paměti zároveň s programem ATMAS-II, avšak v době přípravy tohoto materiálu nebyl k dispozici.
- Všechny programové produkty, tj. překladač, editor, monitor /popřípadě i ladicí program/, zdrojový text vytvářeného programu, přeložený cílový modul /tj. strojový kód/, mohou být zároveň v paměti, což usnadňuje a urychluje ladění.
- Mezi překladačem, editorem, monitorem, ladicím programem a strojovým kódem vytvořeného programu lze předávat řízení.

- Dovoluje definovat a volat makroinstrukce s parametry, makroinstrukce lze do sebe hnizdit. /popis v kap. 6./
- Parametricky lze generovat třeti průchod překladače a ovlivňovat skladbu výpisu překladu.
- Výpis překladu lze tisknout na obrazovku, na ATARI tiskárnu nebo na paralelni tiskárnu.
- Celý zdrojový text programu nebo jeho část lze zapsat a uchovat na disketě nebo na mg. kazetě.
- Do zdrojového textu uloženého v paměti počitače lze přihráním z diskety či mg. kazety vložit /insertovat/ jiný zdrojový text, což umožňuje modulárni skládáni zdrojového textu.
- Neobsahuje linker, překládá přimo do cilového strojového kódu. Neumožňuje tedy linkovat program s moduly jiných překladačů.
- Cilový strojový kód lze pomoci monitoru zapsat na mg. kazetu nebo disketu a takto uložený soubor lze dále zpracovávat jiným, vyšším jazykem /např. ATARI Basicem/. Soubor lze zavést a připojit k programu v Basicu /viz. kap. 10./.
- Umožňuje volit logickou a fyzickou adresu překladu.
- Monitor, který je v překladači zabudován, umožňuje vykonávat základni funkce /výpis paměti, modifikace paměti, disasembler, start strojového programu, přesun paměti a pod./.

Překlad se provádi podle následujiciho obrázku:



obr. 34 - Princip překladu zdrojového textu překladačem ATMAS-II

Popis ATMAS-II

Makroasembler ATMAS-II se před zahájením práce zavede obvyklým způsobem z mg. kazety nebo diskety.

Je-li uložen na mg. kazetě, skládá se ze dvou částí, a to ze zavaděče BL/C V2.0 a z vlastního strojového kódu makroasembleru ATMAS-II V1.0. Zavede se následujícím postupem:

1/ Zapnout počítač a přidržet tlačítka START a OPTION nebo jsme-li v ATARI Basicu zapsat povel BYE a stisknout RETURN, pak přidržet současně START a OPTION a lehce zavadit o tlačítko RESET.

2/ Přetočit kazetu na začátek záznamu, stisknout tlačítko PLAY na magnetofonu a potvrdit stisknutím libovolné klávesy mimo BREAK.

3/ Po natažení zavaděče /asi 6 bloků/ stisknout klávesu A, volba: A binary load/EXECUTE from "C:" a potvrdit libovolnou klávesou mimo BREAK.

4/ Po natažení souboru ATMAS-II vypnout magnetofon STOP/EJECT.

Jestliže byl program ATMAS-II úspěšně natažen do paměti, pak ho zaváděcí program BL/C automaticky odstartuje.

ATMAS-II obsahuje tři samostatné programové moduly:

- editor,
- monitor,
- překladač.

Rozdělení operační paměti při implementaci ATMAS-II

Program ATMAS-II přiděluje, využívá a spravuje 64 KB přímo adresovatelné paměti následujícím způsobem:

$0000 až $00FF ... zápisníková paměť, datové struktury operačního systému
$0100 až $01FF ... zásobníková paměť
$0200 až $05FF ... datové struktury operačního systému a ATMAS-II
$0600 až $06FF ... volné - vhodné místo pro uložení dat a programů uživatele

$0700 až $09FF ... při použití magnetofonu je zde zavaděč BL/C, při použití diskety je zde DOS

$0A00 až $27FF ... volné - vhodný prostor pro data a programy uživatele; při použití diskety je zde DOS

$2800 až $4CF5 ... program ATMAS-II /editor, překladač, monitor/

$4CF6 až $63FF ... volné - vhodný prostor pro data a programy uživatele

$6400 až $A7FF ... textový buffer pro editor

$A800 až $AA41 ... volné, vhodné pro uložení programu a dat

$AA42 až $BC1F ... pracovní prostor ATMAS-II

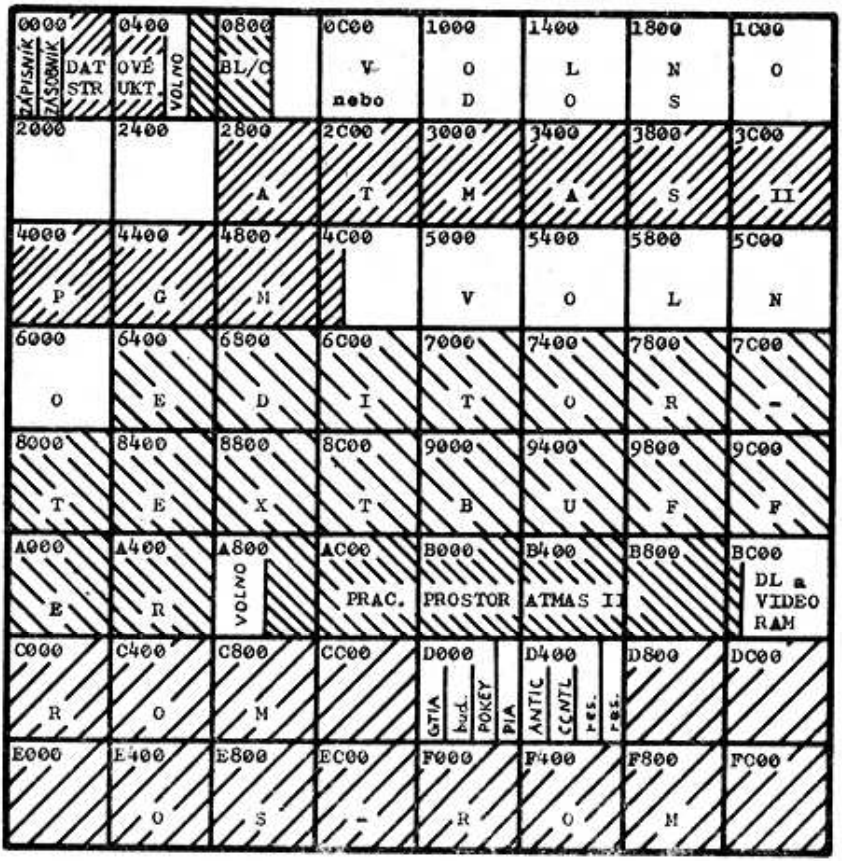
$BC20 až $BC3F ... program pro procesor ANTIC, tzv. Display List

$BC40 až $BFFF ... video RAM

$C000 až $CFFF ... OS ROM

$D000 až $D7FF ... V/V kanály /GTIA, POKEY, PIA, ANTIC, CCNTL/

$D800 až $FFFF ... OS ROM



obr. 35 - Rozdělení operační paměti při implementaci ATMAS-II

7.1 Editor ATMAS-II

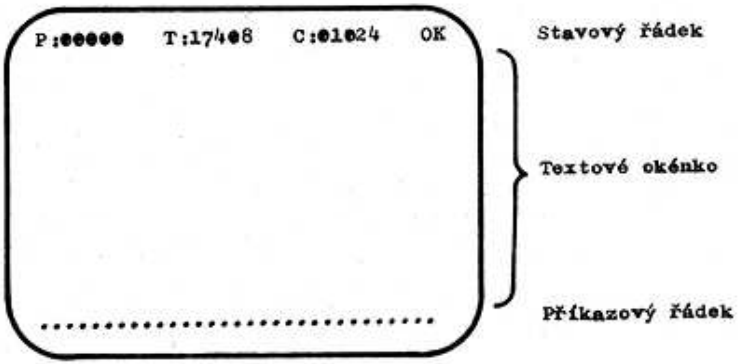
Program editor slouží k vytváření, opravování a uchování zdrojového textu programu. Má dva režimy práce:

- základní neboli přímý povelový režim,
- nepřímý příkazový režim.

Zaváděcí program BL/C předává řízení editoru, který je nastaven do základního režimu. Do základního režimu se dostaneme vždy, když stiskneme klávesu RESET*.

Obrazovka je rozdělena na tři části:

- stavový řádek /první řádek obrazovky/,
- textové okénko /druhý až předposlední řádek obrazovky/,
- příkazový řádek /poslední =vytečkovaný= řádek obrazovky/.



obr. 36 - Rozdělení obrazovky pro editor

Setkal jsem se s verzí ATMASu-II, kde stisknutí klávesy RESET vedlo ke zhroucení systému. Stačilo opravit obsahy buněk BOOT ⟨$09⟩←1, DOSINI ⟨$0C.D⟩←$283C, CASINI ⟨$02.3⟩←$070B, popřípadě i DOSVEC ⟨$0A.B⟩←$0708. Ale pozor, v této verzi byly i další chyby.
Je-li ATMAS-II zaveden z disku, pak Reset vrací do DOS.

Stavový řádek editoru

Ve stavovém řádku /Status Line/ jsou uvedeny důležité informace o stavu editoru:

P:nnnnn ... zobrazuje počet znaků /byte/ od začátku textu k aktuální pozici kurzoru

T:nnnnn ... zobrazuje aktuální počet volných byte ve vyhrazeném paměťovém prostoru pro textový buffer /max. 17 408 byte/ = 17 kB

C:nnnnn ... zobrazuje aktuální počet volných byte v kopírovacím bufferu /max. 1024 byte/ = 1 kB

Poslední dva znaky na stavovém řádku zobrazují buď stavové informace o kopírovacím bufferu nebo mají význam chybového hlášení.

Stavová hlášení editoru

Stavová hlášení se nacházejí na posledních dvou pozicích stavového řádku. Může nabývat dvou hodnot:

OK ... indikuje normální stav, kopírovací buffer je uzavřen,

CR ... informuje o otevřeném kopírovacím bufferu.

Chybová hlášení editoru

Chybová hlášení jsou zobrazována na posledních dvou pozicích stavového řádku. Pozor, neplést si s chybovým hlášením překladače!

Chybové hlášení editoru reaguje na nesprávnou práci s editorem, s příkazovým řádkem či vstupními/výstupními zařízeními.

RW ... nepřípustný příkaz READ nebo WRITE, například volání neexistujícího zařízení či souboru nebo chyba, ke které došlo při operaci čtení nebo zápisu

CO ... příkazový řádek je přeplněn

E? ... nepřípustný nebo žádný příkaz na příkazovém řádku

H? ... nesprávný nebo žádný hexadecimální argument

I? ... textový buffer vymezený pro zdrojový text programu je přeplněn /T:00000/

L? ... nesprávné výstupní zařízení pro tisk, neúspěšný tisk nebo není přítomna tiskárna

S? ... hledání řetězce ve zdrojovém textu je dokončeno a řetězec nebyl nalezen

T? ... nesprávná hodnota tabulátoru

C? ... kopírovací buffer je přeplněn /C:000000/

#? ... nesprávný argument pro opakované provádění příkazu

Chybová hlášení lze odstranit a "uvolnit" systém pro další operování několika způsoby:

1/ přechodem do základního přímého režimu stisknutím RESET, /pro ATMAS-II zavedený z mg. kazety/,

2/ přechodem do základního přímého režimu dvojnásobným stisknutím ESCAPE,

3/ návratem do příkazového režimu stisknutím ESCAPE a zopakováním příkazu.

Textové okénko

Textové okénko obsahuje 21 řádků a 38 sloupců střední části obrazovky. Slouží k zobrazování zdrojového textu programu. Okénkem můžeme posouvat /rolovat/ nahoru i dolů po celém zdrojovém textu.

Při editaci textu je výsledek činnosti ihned zobrazován v textovém okénku.

Editor pracuje v tzv. insertovacím režimu. To znamená, že na aktuální pozici kurzoru lze vkládat znak nebo část textu, aniž by se přepsal následující text. Takto lze vkládat i řídící znaky /např. CR, TAB/.

Kurzor lze přemísťovat pomocí kláves pro pohyb kurzoru nahoru, dolů, doleva a doprava, na začátek textu a na konec textu. Je možné i rolování nahoru a dolů.

Na rozdíl od editoru ATARI Basicu, kde se kurzorem může pohybovat po celé obrazovce, lze v editoru ATMASu-II pohybovat kurzorem pouze po zdrojovém textu.

Řádek textového okénka obsahuje sice pouze 38 sloupců, logický řádek může však mít až 127 znaků. Znaky za 38. pozicí nejsou viditelné. Prvních 76 znaků lze zviditelnit přepínačem režimů zobrazení CTRL-V.

Editovat lze v přímém režimu pomocí kurzoru, textového okénka a editovacích povelů editoru.

Příkazový řádek

Příkazový řádek editoru se nachází na poslední řádce obrazovky. Pro zvýraznění je celý vytečkován. Umožňuje řetězit nepřímé příkazy editoru a zobrazovat je na tomto řádku.

Do příkazového řádku lze zapisovat a pak vykonávat příkazy v tzv. příkazovém režimu editoru. Do tohoto režimu se lze dostat ze základního režimu editoru stisknutím klávesy ESCAPE.

7.2 Základní přímý mód editoru

V základním přímém módu editoru může být pořizován zdrojový text programu. Text je zapisován do textového okénka a uchováván v textovém bufferu, který se nachází v paměti na adrese $6400 až $67FF.

Každý znak je vkládán na aktuální pozici kurzoru na obrazovce. Jestliže držení klávesy bude delší než 1 sec, funkce klávesy bude opakována asi 8 krát za sekundu. Opakování funkce klávesy se vykonává tak dlouho, dokud se kurzor nachází v rozsahu textového okénka.

Pro pohyb kurzoru, manipulaci s textem a pod. existuje řada různých povelů, všechny jsou inicializovány stisknutím tlačítka CONTROL a příslušné klávesy. Povely jsou vykonávány ihned po jejich zadání.

Seznam přímých povelů editoru

Povely pro pohyb kurzoru a mazání textu:

CTRL-A, CTRL-→ ... kurzor o jednu pozici vpravo

CTRL-Q, CTRL-← ... kurzor o jednu pozici vlevo

CTRL-S, CTRL-→ ... kurzor na začátek dalšího řádku

CTRL-W, CTRL-↑ ... kurzor na začátek téže nebo předcházející linky

CTRL-I, TAB ... kurzor na další tabelační pozici

CTRL-M, RETURN ... ukončení řádku, kurzor na začátek dalšího řádku

CTRL-H, DEL ... vymazání jednoho znaku vlevo

CTRL-U, CTRL-DEL ... vymazání jednoho znaku vpravo

CTRL-X ... vymazání řádku od začátku až po kurzor nebo celého předešlého řádku, je-li kurzor umístěn na první pozici řádku

Pozn.: CTRL-key znamená stisknutí klávesy CONTROL a příslušné klávesy. Některé povely jsou zdvojené, což umožňuje pohodlnější operování. Například pohyb kurzoru lze ovládat jednou rukou.

Povely pro práci s kopírovacím bufferem:

CTRL-R ... otevření kopírovacího bufferu
/po otevření lze pro kopírování použít pouze CTRL-→, CTRL-Q, CTRL-↑, CTRL-W, CTRL-E, CTRL-H, DEL, CTRL-X/

CTRL-F ... uzavření kopírovacího bufferu

CTRL-J ... vložení obsahu kopírovacího bufferu na místo aktuální pozice kurzoru

CTRL-K ... vymazání obsahu kopírovacího bufferu
/C:01024/

Povely pro předávání řízení:

ESC ... otevření příkazové řádky, přechod do příkazového režimu
/ESC - oddělení příkazů,
ESC ESC - ukončení a odeslání příkazové řádky, návrat do základního režimu/

CTRL-Y ... start překladu, přechod do asembleru
/libovolná klávesa - po ukončení překladu návrat do základního režimu editoru/

CSTRL-P ... skok do monitoru
/E - návrat do základního režimu editoru/

RESET ... návrat ze všech režimů a stavů do základního režimu editoru /u disk. verze vrací do DOS/, nemaže zdrojový text

Povely pro přepínání režimů zobrazení:

CTRL-T ... režim zviditelnění kontrolních znaků ve zdrojovém textu
například TAB se zobrazí jako inverzní I, RETURN jako inverzní M

CTRL-V ... režim plného řádkového zobrazení, zviditelňuje znaky, které byly vloženy do řádku mimo textové okénko; zobrazuje prvních 76 znaků logických řádků

Pozn.: Opakováním povelu se lze vrátit do původního režimu zobrazení.

Povely pro nastavení značek ve zdrojovém textu:

CTRL-L ... nastavení značky FORM FEED - nový formulář; má význam pro tiskárnu vybavenou traktorem, která v tomto místě textu nastaví novou stránku

CTRL-Z ... nastaví značku "konec překladu"; má význam pro překladač, který v tomto místě zdrojového textu ukončí svoji činnost; jedná se o obdobu instrukce END u jiných překladačů; není-li použito, překládá se celý zdrojový text

Povely ostatní:

CTRL-G ... povel pro opakování příkazů v příkazovém řádku

7.3 Nepřímý příkazový režim editoru

Pomocí klávesy ESCape je možné přejít ze základního režimu do příkazového režimu editoru. Příkazový režim se ohlásí znakem $ na první pozici příkazového řádku. Tím je řádek přístupný pro zápis příkazů z klávesnice.

Příkazy je možné řetězit, takže lze do příkazového řádku zapsat více příkazů najednou. Jednotlivé příkazy se

odděluji stisknutím klávesy ESCape /nikoli SPACE, RETURN nebo TAB/, což se na příkazovém řádku projeví zobrazením znaku $.

Pro opravování lze použít klávesu DEL nebo CTRL-H. Pro rušení celého řádku a návrat do základního režimu editoru lze použít CTRL-X.

Připravené povely na příkazovém řádku lze spustit dvojnásobným stisknutím klávesy ESC. Příkazy se vyhodnocují a provádějí postupně zleva doprava. Po ukončení činnosti nebo při chybě dochází k automatickému přechodu do základního režimu. Místo posledního znaku $ se dosazuje znak ↵! Odtud lze kdykoli povelem CTRL-G zopakovat činnost příkazového řádku.

Seznam nepřímých příkazů editoru

Příkazy pro editování:

@n	... nastavení tabulační šířky, $n \in \{0,1,\ldots,9\}$
B	... kurzor o jednu pozici zpět
F	... kurzor o jednu pozici vpřed
D	... vymazání znaku vlevo od kurzoru
T	... vymazání všech znaků mezi aktuální pozicí kurzoru a prvním znakem dalšího řádku
K	... vymazání celého textu!
G	... vložení obsahu kopírovacího bufferu na místo aktuální pozice kurzoru
J	... automatické opakování příkazové řádky
H byte	... vložení hexadecimálního znaku na místo aktuální pozice kurzoru, $byte \in \{00,01,\ldots,FF\}$
I string	... vložení řetězce do textu na místo aktuální pozice kurzoru; string ... ASCII řetězec bez oddělovačů
S string	... hledání ASCII řetězce v textu od aktuální polohy kurzoru do konce textu; kurzor se nastaví bezprostředně za hledaný řetězec
ESC	... oddělení příkazů na jedné řádce; zobrazí se $

Příkazy V/V operací:

```
!          ... výpis zdrojového textu na obrazovku;
               výpis lze přerušit a opět spustit pomocí
               CTRL-1
L x        ... výpis na tiskárnu:
               x=0 ... tiskárna RS 232 Port 1
                 1 ... paralelní tiskárna
                 2 ... ELCOMP
R dev      ... čte zdrojový text z určitého zařízení
               /z diskety nebo kazety/ a ukládá ho na místo
               aktuální polohy kurzoru
               dev = C:
                     D:NAME.EXT
                     Dn:NAME.EXT
W dev      ... zapisuje zdrojový text od aktuální polohy
               kurzoru do konce textu na disketu nebo
               kazetu
               dev = C:
                     Dn:NAME.EXT
```

Příkazy pro předávání řízení:

```
U          ... příkaz pro odstartování uživatelského pro-
               gramu na adrese $A800 nebo cartridge na
               adrese $8000;
               RTS vrací do základního režimu editoru
               BRK vrací do monitoru s výpisem obsahů
               registrů a příznaků
ESC ESC    ... uzavření příkazového řádku a vykonání příka-
               zů; po provedení vrací do základního režimu
               editoru
CTRL-X     ... vymazání příkazové řádky, návrat do základ-
               ního režimu editoru aniž by se příkazová
               řádka interpretovala
RESET      ... návrat do základního režimu editoru, pří-
               kazový řádek se neprovede
```

Pozn.: Pokud před příkaz na příkazovém řádku napíšeme desetinální číslo m, kde m∈ {1,2,...,255}, bude se příkaz m-krát opakovat.
Například: $3B$$ ... kurzor o 3 pozice zpět.

Příklady použití příkazů a povelů editoru

Příklad 1:

Ve zdrojovém textu nastavte kurzor za první návěští se jménem LABEL.

Řešení:

```
a/ CTRL-E            ; kurzor na začátek textu
b/ $SLABEL$$         ; vyhledá první LABEL a nastaví
                       za něj kurzor
```

Příklad 2:

Ve zdrojovém textu nastavte kurzor za druhé návěští, které se jmenuje LABEL.

Řešení A:

```
a/, b/ stejné jako v příkladu 1
c/ CTRL-G            ; opakuje činnost naposledy
                       zadané příkazové řádky,
                       tedy od polohy kurzoru hledá
                       další řetězec LABEL
```

Řešení B:

```
a/ CTRL-E
b/ $2SLABEL$$
```

Příklad 3:

Ve zdrojovém textu nastavte kurzor před první návěští se jménem LABEL.

Řešení:

a/ CTRL-E

b/ $SLABEL$5B$$; zřetězení dvou příkazů:
vyhledej řetězec LABEL a posuň
kurzor o 5 pozic vzad

Příklad 4:

Nahraďte ve zdrojovém textu první řetězec ALFA za BETA.

Řešení:

a/ CTRL-E

b/ $SALFA$4D$IBETA$$

Příklad 5:

V celém zdrojovém textu nahraďte všechna slova LABEL za NAVESTI.

Řešení:

a/ CTRL-E

b/ $SLABEL$5D$INAVESTI$J$$

Příklad 6:

Ve zdrojovém textu nahraďte každou druhou instrukci LDA ZDROJ za STA $F831.

Řešení:

a/ CTRL-E

b/ $2SLDA ZDROJ$ISTA $H24$IF831$J$$
/$H24 nahrazuje ASCII "$"/

Příklad 7:

Zapište celý zdrojový text programu na mg. kazetu.

Řešení:

a/ CTRL-E kurzor na začátek programu!

b/ $WC:$$

Příklad 8:

Přečtete celý zdrojový text programu z mg. kazety.

Řešení:

a/ $K$$ nejprve vymaže celý textový buffer a pak zavede

b/ $RC:$$ zavede

Příklad 9:

Zaveďte zdrojový text programu. Program je uložen na disketě a skládá se ze dvou částí:

- DEFMACRO.MAC,
- MAINPRG.MAC,.

Řešení:

a/ $K$$

b/ $RD:DEFMACRO.MAC$$

c/ $RD:MAINPRG.MAC$$

Příklad 10:

Zaveďte zdrojový text programu z magnet. kazety. Program se skládá ze dvou souborů TEXT1 a TEXT2 uložených za sebou na mg. pásce.

Řešení:

a/ $K$$

b/ $RC:$$

c/ $RC:$$

Příklad 11:

Podobně jako v příkladu 10 zaveďte zdrojový text programu z mg. kazety, ale v opačném pořadí tak, aby v textovém bufferu počítače byl nejprve TEXT2 a za ním TEXT1.

Řešení A:

a/ KRC:$$

b/ CTRL-E

c/ $RC:$$

Řešení B:

a/ KRC:$RC:$$

b/ pomocí kopírovacího bufferu překopírovat TEXT1 za TEXT2

c/ nastavit kurzor za původní TEXT1
d/ vymazat TEXT1 dopředným mazáním držením CTRL-X

Řešení C:
a/ KRC:$RC:$$
b/ pomocí kopírovacího bufferu překopírovat
TEXT2 před TEXT1
c/ nastavit kurzor na začátek původního TEXT2
d/ vymazat první řádek příkazem $T$$
e/ opakovat mazání až do konce držením CTRL-G

Řešení D:
a/ nastavit magnetofon na začátek TEXT2
b/ $K
c/ $RC:$$
d/ nastavit magnetofon na začátek TEXT1
e/ $RC:$$

Příklad 12:
Zapište na mg. kazetu část programu, která začíná poznámkou "* HLAVNÍ PROGRAM".

Řešení:
a/ CTRL-E
b/ S* HLAVNÍ PROGRAM$16B$WC:$$

7.4 Práce s kopírovacím bufferem

Takzvaný kopírovací buffer je pomocný textový buffer, který může obsahovat maximálně 1024 znaků /1 KB RAM/. Jeho obsah lze vložit /insertovat/ na libovolné místo v hlavním textovém bufferu editoru.

Před vkládáním textu do kopírovacího bufferu musí být tento buffer otevřen povelem CTRL-R. Po otevření je spojen s hlavním textovým bufferem a editovací příkazy jsou vykonávány paralelně na obou bufferech.

Zkopírování textu z hlavního bufferu do kopírovacího se provádí pomocí editovacích povelů, které pohybují kurzorem po vybraném textu směrem zpět /CTRL-Q,CTRL-→,CTRL-E, CTRL-↑,CTRL-H,DEL,CTRL-X/. Na status řádku je možno na počítadle C:xxxxx sledovat aktuální počet volných byte kopírovacího bufferu.

Na závěr kopírování textu musí být kopírovací buffer uzavřen povelem CTRL-F. Text v kopírovacím registru je nyní chráněn proti obyčejným editovacím operacím. Kopírovací buffer je automaticky uzavírán a chráněn při volání příkazového režimu a při skoku do asembleru.

Obsah kopírovacího bufferu může být opakovaně kopírován do textového bufferu povelem CTRL-J na místo aktuální pozice kurzoru v textovém bufferu.

S kopírovacím bufferem lze pracovat i v příkazovém režimu. Například sekvencí: §SJMP DAL§7D§G§J§§ nahradí v celém textu instrukci JMP DAL obsahem kopírovacího bufferu.

7.5 Práce s překladačem

Asembler může být odstartován přímo ze základního režimu editoru povelem CTRL-Y. Překládá zdrojový text programu /source code/ uložený v textovém bufferu editoru ve dvou resp. třech průchodech podle volby dané pseudoinstrukcí OUT. Zdrojový text se překládá od začátku až po znak CTRL-Z. Není-li tento znak v textu uveden, provádí se překlad až do konce textu. Provádí-li asembler druhý průchod, jsou známky činnosti indikovány změnou zobrazeného znaku na poslední pozici status řádku.

Pomocí parametrů pseudoinstrukce OUT lze volit skladbu výstupní sestavy pro třetí průchod:

- L ... výstup překládaného zdrojového textu doprovázeného logickými adresami překladu a strojovým kódem
- N ... výstup seznamu použitých návěští, případně: informace o jejich nevyužití /UNUSED/

M ... potlačení výpisu zdrojového textu v rozvoji použité makroinstrukce

Pn ... výstup na tiskárnu; n zadává tiskárnu:
1 ... RS 232 C Port 1
2 ... paralelní
3 ... ELCOMP
Není-li zadána volba tiskárny, je výstup prováděn na obrazovku.

Jestliže během překladu není nalezena žádná syntaktická chyba, asembler generuje cílový strojový kód a uloží ho na fyzickou adresu určenou pseudoinstrukcí ORG. O správnosti překladu informuje hlášením:

PHYSICAL END ADDRESSING $....,

kde hexadecimální číslo udává první volnou adresu. Stlačením kterékoli klávesy se můžeme vrátit do základního režimu editoru.

Byla-li nalezena syntaktická chyba, asembler ukončí svoji činnost a vypíše chybové hlášení. Stisknutím kterékoli klávesy se vrátíme do základního režimu editoru. Kurzor je nastaven na místo první nalezené chyby, což zrychluje opravování programu.

Chybová hlášení překladače

LINE TOO LONG	příliš dlouhý řádek /řádek přesáhl 127 znaků/
TOO MANY LABELS	příliš mnoho návěští
DIVISION BY ZERO	nedovolené dělení nulou
NO ASCII	není ASCII znak
UNDEFINED EXPRESION	nedefinovaný výraz
NUMBER ERROR	numerická chyba
SYNTAX ERROR	syntaktická chyba
NAME UNKNOWN	neznámé jméno
ORG ERROR	chyba v pseudoinstrukci ORG
SAME LABEL TWICE	stejné návěští dvakrát
WRONG DELIMITER	nesprávný oddělovač
MACRO ERROR	chyba v makroinstrukci

IMPOSSIBLE BRANCH ..... nemožný skok

ADDRESSING ERROR ..... chyba v adresování

WARNING OPERAND OVERFLOW ... pokus o umístění dvoubajtového výrazu do jednobajtové lokace; POZOR! Tato chyba nemusí být v některých případech hlášena, pak se bere v úvahu pouze nižší byte výrazu.

OPCODE DIFFERENT ..... rozpor v překladu operačních kódů; Například: jsou-li dvě pseudoinstrukce ORG v rozporu a překlad má být proveden na překrývající se oblasti nebo rozpor ve vyhodnocení návěští v následujících průchodech překladu.

Předávání řízení

Jak již bylo naznačeno, program ATMAS-II se může nacházet v těchto režimech:

1/ v přímém, tj. základním režimu editoru,
2/ v příkazovém režimu editoru,
3/ v monitoru,
4/ v překladači,
5/ v uživatelském programu.

Tabulka ukazuje, jak lze předávat řízení mezi jednotlivými režimy:

do režimu / z režimu	základní režim editoru	příkazový režim editoru	monitor	překladač	uživat. program
základní režim editoru	=	ESCAPE	CTRL-P	CTRL-Y	-
příkazový režim editoru	2 * ESC	=	-	-	U ($A800)
monitor	E X,E RESET** G283C		=		Gstart.adr.
překladač	lib. kláv. RESET**	-	-	=	-
uživatelský program	JMP $283C* RESET**		RTS* BRK*		-

TAB. 78 Předávání řízení

* zařadit jako poslední instrukci v programu
** u disketové verze vrací do DOSu

Pozn.: Z monitoru je možné přímo startovat a využívat některé rutiny operačního systému. Pokud jsou ukončeny instrukcí RTS, vracejí řízení monitoru. Z některých však není návratu. Například:
G E477 ... studený start Basicu
G E471 ... SELF TEST
G 0700 ... start BL/C zavaděče
G E474 ... teplý start (RESET)

7.6 Monitor ATMAS-II

Monitor je program, který pomáhá při statickém sledování programu a dat.
Každý monitor by měl splňovat tři základní funkce:
1/ sledování obsahu pamětí a V/V registrů,

2/ modifikace paměti, V/V registrů,
3/ start programu z libovolného místa paměti
(předávání řízení).

Monitory se pak od sebe liší tím, do jaké míry rozvíjejí základní funkce. Pak hovoříme o úrovni neboli komfortu monitoru. Všimněte si, že tyto tři základní funkce jsou obsaženy v příkazech PEEK, POKE a USR v ATARI BASIC.

Implementovaný monitor v ATMAS-II je méně konfortní než speciální monitory. Obsahuje téměř všechny základní povely, na které jsme u monitorů zvykli. Pro naše účely plně postačí. Na tento monitor se musíme dívat jako na nadstavbu -doplněk překladače a editoru ATMAS-II. Proto jsme autorům vděčni za každý nabídnutý povel.

Ze základního přímého editovacího režimu lze do monitoru vstoupit povelem CTRL-P. Zpět se lze vrátit povelem monitoru E (Editor).

Všechny parametry povelů monitoru jsou zadávány výhradně v hexadecimálním tvaru.

Seznam povelů monitoru ATMAS-II

Povely pro sledování paměti:

M ..... DUMP /od, do,ASCII ?, PRINT ?/ - prohlížení paměti; stisknutí libovolné klávesy - pokračování výpisu

D ..... DISASEMBLER /od, PRINT?/ - zpětný překlad; stisknutí libovolné klávesy - pokračování výpisu

Povely pro modifikaci paměti

C ..... CHANGE /od/ - změna paměti

F ..... FILL /od, do, čím/ - naplnění konstantou

B ..... BLOCK TRANSFER /od, do, kam/ - přesun bloku

S ..... SAVE /od, do, kam, zařízení/ - zápis bloku paměti; k binárnímu obrazu paměti přihrává 6 byte záklaví: FF FF poč. adr., kon. adr.;

nahrávka na magnetofon je v normálním tvaru, tj. s dlouhými meziblokovými mezerami

L ..... LOAD /zařízení/ - čtení bloku paměti, uložení od adresy INTO /kam/ vyplněné při SAVE

Povely pro předávání řízení

X ..... EXIT - návrat do monitoru, ukončení povelů M, D, C; lze použít i k ukončení libovolného povelu monitoru, je-li očekáván vstup parametru

G ..... GOTO /start. adr./ - start libovolného programu ve strojovém kódu;
POZOR! Program je odstartován bezprostředně po zadání poslední hexadecimální cifry adresy. Je-li program ukončen instrukcí RTS, vrací se řízení do monitoru. Je-li program ukončen instrukcí BRK, vrací se řízení do monitoru a navíc je vypsán aktuální stav registrů mikroprocesoru.

E ..... EDITOR - návrat z monitoru do editoru

RESET .... návrat do editoru u kazetové verze, návrat do DOS u disketové verze

7.7 Přehledy syntaxe

Syntax adresovacích režimů

Mikroprocesory 65xx mají 13 druhů adresování, které mají odlišný formální zápis:

1/ IMPL - Implied - zahrnuto v sobě
[Label] ␣ Ins [␣com] ⟨Ret⟩

2/ ACC - Accumulator - přímé adresování akumulátoru
[Label] ␣ Ins [␣com] ⟨Ret⟩

3/ IMM - Immediate - bezprostřední adresování
[Label] ␣ Ins ␣ #op$_8$ [␣com] ⟨Ret⟩

4/ ABS - Absolute - absolutní adresování
[Label]␣[ns␣op$_{16}$ [␣com] ⟨Ret⟩

5/ ABS,X - Absolute X-indexed - X-indexové absolutní adresování
[Label]␣[ns op$_{16}$,X [␣com] ⟨Ret⟩

6/ ABS,Y - Absolute Y-indexed - Y-indexové absolutní adresování
[Label]␣[ns op$_{16}$,Y [␣com] ⟨Ret⟩

7/ ZP - Zero Page - nulostránkové adresování
[Label]␣[ns op$_{8}$ [␣com] ⟨Ret⟩

8/ ZP,X - Zero Page X-indexed - X-indexové nulostránkové adresování
[Label]␣[ns op$_{8}$,X [␣com] ⟨Ret⟩

9/ ZP,Y - Zero Page Y-indexed - Y-indexované nulostránkové adresování
[Label]␣[ns op$_{8}$,Y [␣com] ⟨Ret⟩

10/ REL - Relative - Relativní adresování
[Label]␣[ns op$_{16}$ [␣com] ⟨Ret⟩

11/ IND - Indirect - nepřímé adresování
[Label]␣[ns (op$_{16}$) [␣com] ⟨Ret⟩

12/ (IND,X) - X-indexed Indirect - X-indexové nepřímé adresování
[Label]␣[ns (op$_{8}$,X) [␣com] ⟨Ret⟩

13/ (IND),Y - Indirect Y-indexed - nepřímé Y-indexové adresování
[Label]␣[ns (op$_{8}$),Y [␣com] ⟨Ret⟩

Příklad:

V uvedeném příkladu je použito všech 13 způsobů adresování.

Návěští	Instrukce	Operand	Komentář
ZACATEK	TAX		IMPL
	ROR	A	ACC
	LDA	#$FF	IMM
	STA	$B7FF	ABS
	STA	$B7FF,X	ABS,X
	STA	$B7FF,Y	ABS,Y
	LDY	$0	ZP
	STY	$0B,X	ZP,X
	LDX	$3A,Y	ZP,Y
	BNE	DALE1	REL
	JMP	($0248)	IND
DALE1	ADC	($30,X)	(IND,X)
	STA	($DE),Y	(IND),Y

Syntax pseudoinstrukcí

1/ EQU - Equal - přiřazení 2 byte symbolu návěští

label ␣ EQU ␣ op_8 [␣com] ⟨Ret⟩

2/ EPZ - Equal Page Zero - přiřazení 1 byte symbolu návěští

label ␣ EPZ ␣ op_8 [␣com] ⟨Ret⟩

3/ DFB - Define Byte - definování byte

[label] ␣ DFB ␣ op_8 [␣com] ⟨Ret⟩

4/ DFW - Define Word - definování slova

[label] ␣ DFW ␣ op_8 [␣com] ⟨Ret⟩

5/ ASC - ASCII String - definování řetězce

[label]␣ ASC ⟨del⟩ [str] ⟨del⟩ [,⟨del⟩ [str] ⟨del⟩]″ [␣com] ⟨Ret⟩

6/ ORG - Origin - definování logické a fyzické adresy překladu

[label]␣ ORG␣log [,fyz] [␣com] ⟨Ret⟩

7/ OUT - Output - definování skladby výpisu překladu

[label]␣ OUT [L] [N] [N] [P[n]] [␣com] ⟨Ret⟩

Syntax makroinstrukcí

1/ Definice makroinstrukce:

name MACRO [par_1 [,par_2 [,par_3..]]] [␣com] ⟨Ret⟩

2/ Ukončení definice makroinstrukce:

[label] MEND [␣com] ⟨Ret⟩

3/ Volání makroinstrukce:

[label] name [val_1 [,val_2 [,val_3..]]] [␣com] ⟨Ret⟩

Použité symboly

[ ]	... výrazy uvedené v závorkách jsou nepovinné
[ ]″	... výraz umístěný v závorce může být opakován
␣	... oddělovač: SPACE /mezerník/, TAB, CTRL-I
label	... návěští skládající se maximálně z osmi alfanumerických znaků, přičemž první znak musí být písmeno
name	... jméno makroinstrukce
⟨del⟩	... omezovač - kterýkoli nealfanumerický znak /např. ', "/; zvláštní význam mají: - znak \ , který automaticky zvyšuje hodnotu posledního znaku řetězce o $80 /tj. 128/, - znak / , který ke všem ASCII znakům řetězce přičítá $80 /tj. 128/
str	... řetězec ASCII znaků - maximálně 250 znaků; musí být otevřen i uzavřen stejným omezovačem
⟨Ret⟩	... omezovač řádku, "Carriage Return" /CR/, vykonává se prostřednictvím klávesy RETURN nebo STRL-H
⟨com⟩	... komentář
Ins	... název instrukce /viz. Příloha 1/
op_8	... osmibitový operand /viz kapitola 3.1/
op_{16}	... šestnáctibitový operand /viz kapitola 3.1/
log	... logická adresa přeloženého programu
fyz	... fyzická adresa překladu
L	... volba výpisu překladu
N	... volba výpisu tabulky symbolů
M	... potlačení výpisu zdrojového textu makroinstrukci v místě volání
par_i	... formální parametr makroinstrukce /platí stejné zásady jako pro label/
val_i	... skutečná hodnota parametru volané makroinstrukce; může to být: - binární, hexadecimální či decimální číslo,

- definovaná symbolická proměnná,
- ASCII řetězec omezený znaky ', " ,\, / .
- aritmetický výraz v kulatých závorkách.

8. Zásady programování v JSA

Dodržování jakýchkoli zásad při programování vede vždy k určitému omezení programátora. Na druhé straně to však přináší mnohé výhody.

Každý program lze v určitém smyslu zefektivnit /tj. optimalizovat/. Nejčastěji se program optimalizuje podle těchto hledisek:

a/ podle spotřeby času,

b/ podle spotřeby paměti,

c/ podle přehlednosti.

Současné mikropočítače mají dostatečně velkou paměť a jsou i dosti rychlé. Proto se dnes nejvíce uplatňuje kriterium přehlednosti. To pak platí dvojnásob pro programy v JSA, které jsou samy o sobě méně přehledné než programy ve vyšších jazycích.

Přehlednosti lze dosáhnout dodržováním zásad strukturovaného programování, které přináší:

- zvýšení čitelnosti programu,
- snadnější verifikovatelnost /opravitelnost/ a modifikovatelnost /rozšiřitelnost, přenositelnost, pružnost/ programu,
- možnost modulární výstavby programu,
- usnadnění kolektivní tvorby programů,
- získání dovedností a návyků vedoucích k rychlejší tvorbě programů,
- snadnější popisování programů,
- atd.

8.1 Zásady strukturovaného programování

Strukturované programování je popsáno v mnoha publikacích. V některých se autoři omezují na popis nejrůznějších strukturovaných konstrukcí. Jinde je strukturované programování chápáno jako metodika programování.

Strukturované programování v JSA má své specifické rysy. Odlišuje se od programování v TURBO BASICu či PASCALu, kde existují speciální řídicí konstrukce /například IF...THEN...ELSE...ENDIF a pod./.

8.1.1 Zásada rozkladu složitější úlohy na moduly

Každou úlohu lze rozložit na části /moduly/, které mohou být řešeny postupně a samostatně. Modulem může být v JSA poprogram nebo makroinstrukce. Prakticky každý modul lze ještě dále rozložit na menší celky /podmoduly/.

Jak by měl být modul v JSA rozsáhlý?

Tak, aby uceleně řešil danou část úlohy a zároveň aby programátor neztratil přehled o tom, co dělá. V praxi to znamená, že by modul neměl přesáhnout rozsah dvou až tří obrazovek.

Pozor! Především začátečníci se snaží řešit úlohu najednou. Získávají tak často nepřehledný kolos, ve kterém se snadno sami ztratí.

Při analýze programů profesionálních programátorů zjistíte, že rozsah zdrojového textu modulu bývá obvykle menší než jedna stránka formátu A4.

8.1.2 Zásada minimalizace toku informací mezi moduly

Jednotlivé moduly si předávají potřebné informace. Při analýze úlohy je vhodné navrhnout moduly tak, aby si předávaly co nejméně informací.

K přenosu informací používáme přednostně:

- příznakové bity registru P,
- registry A, X, Y.

Dále můžeme používat:

- zápisníkovou paměť /volné jsou buňky $CB až $D1, operační systém ani BASIC je nepoužívají, avšak

ATMAS-II využívá v softwarovém přerušení, vyvolané instrukcí BRK, k uložení aktuálních hodnot registrů mikroprocesoru/,

- zásobník /pozor na úroveň otevření!/,
- paměť RAM:
  - uvnitř programu,
  - mimo program.

Chceme-li komunikovat s operačním systémem, musíme respektovat jeho způsob předávání informací. Nejčastěji jsou využívány:

- příznakové bity registru P,
- registry procesoru 6502 - A, X, Y,
- datové struktury OS na RAM /systémové proměnné, ukazatele, buffery, příznaky,.../,
- registry ostatních programovatelných obvodů /GTIA, ANTIC, POKEY, PIA/.

8.1.3 Zásada jednoho vstupního a jednoho výstupního bodu modulu

Každý modul by měl mít pouze jeden vstupní a jeden výstupní bod.

Vstupním bodem /entry point/ je adresa volání. Rozvětvení programu na jednotlivé větve by mělo být realizováno uvnitř modulu podle zadaného vstupního parametru.

Pokud z nějakého důvodu požadujeme více vstupních bodů, pak by měl modul obsahovat tabulku vstupních bodů. Například:

```
ENTER1   JMP SLUZBA1
ENTER2   JMP SLUZBA2
ENTER3   JMP SLUZBA3
SLUZBA1  ...
         ...
SLUZBA2  ...
         ...
SLUZBA3  ...
         ...
```

V žádném případě nejsou při strukturovaném programování přípustné skoky doprostřed modulu!

Není chybou, je-li uprostřed modulu volán jiný modul, jestliže po skončení práce volaného modulu se vracíme zpět za místo volání. Tento postup zcela vychází z principů strukturovaného programování.

Za jeden výstupní bod můžeme považovat i ten případ, kdy je každá větev uvnitř modulu samostatně ukončena instrukcí RTS /nebo RTI/, protože tyto instrukce vracejí řízení do jednoho bodu - za místo volání.

8.1.4 Zásada používání konstrukcí strukturovaného programování

a/ Lineární sekvence

Lineární sekvencí se rozumí řazení instrukcí či modulů postupně za sebou bez jakéhokoli větvení. Je považována za nejpřehlednější typ konstrukce.

b/ Uzavřený cyklus

Cyklus má mít jeden vstup a jeden výstup. Nelze vstupovat a vystupovat "uprostřed" cyklu.

Pomocí instrukcí podmíněných skoků, inkrementací či dekrementací, popřípadě komparací, lze simulovat konstrukce typu:

```
     REPEAT...UNTIL
     WHILE... ...WEND
     FOR...NEXT

     LDX#$5
LOOP STA $600,X
     DEX
     BLP LOOP
```

c/ Podmíněný přeskok

Podmíněný přeskok lze realizovat například takto:

```
            B.. DAL
            ...
DAL         ...
```

Je to podobné konstrukci IF...THEN..., ale zde je negovaná podmínka. Odpovídá to tedy spíše IF NON...THEN... .

d/ Rozvětvení

Jedná se o simulaci konstrukce IF...THEN...ELSE...ENDIF. Pro JSA je v tomto případě povoleno použít instrukce nepodmíněného skoku JMP k vyvedení programu ze slepé větve do společného uzlu obou větví.

```
            CMP ...
            B.. THEN        ; IF...THEN...ELSE
ELSE        ...             ; podmínka není splněna
            ...
            JMP SPOL
THEN        ...             ; podmínka je splněna
            ...
SPOL        ...             ; společné pokračování
```

e/ Multirozvětvení

Jde o simulaci konstrukce typu přepínač. Například rozřazení programu podle hodnoty akumulátoru, kde $\langle A \rangle \in \{0,1,2,3,...\}$ /ordinární soubor/, lze provést:

komparací:

```
MULTI       CMP#0
            BEQ VEKTOR1
            CMP#1
            BEQ VEKTOR2
            CMP#2
            BEQ VEKTOR3
            ...
```

dekrementací:

```
MULTI       TAX
            BEQ VEKTOR1
            DEX
            BEQ VEKTOR2
            DEX
            BEQ VEKTOR3
```

Paulova metoda:

```
TABULKA     DFW VEKTOR1-1
            DFW VEKTOR2-1
            DFW VEKTOR3-1
            ...
MULTI       ASL A               ; 2* (A)→(A)
            TAX
            LDA TABULKA+1,X
            PHA                 ; Hi
            LDA TABULKA,X
            PHA                 ; Lo
            RTS                 ; skok na vektor_n
```

8.1.5 Zásada přednosti jednodušší konstrukce před složitější

Řešení pomocí jednodušší konstrukce jsou zpravidla průhlednější a v mnohých případech efektivnější i z hlediska spotřeby paměti a času.

Příklad:

Porovnejte obsahy buněk $CC a $CD, ve kterých jsou uložena čísla v rozsahu 0 až 255. Je-li ($CC) menší než ($CD), nastavte obsah akumulátoru na samé jedničky /tj. $FF→(A)/, jinak na samé nuly /tj. 0→(A)/.

Řešení A: rozvětvením - konstrukcí IF...THEN...ELSE...

```
            LDA $CC
            CMP $CD        ; je-li (CC)<(CD), pak 0→(C)
            BCC THEN       ; skok, je-li (C)=0
ELSE        LDA#0
            BEQ *+4        ; vždy
THEN        LDA#$FF
```

Spotřeba paměti 12 byte, času 14 taktů.

Řešení B: jednodušší a efektivnější řešení s podmíněným skokem

```
            LDA $CC
            CMP $CD        ; je-li (CC)<(CD), pak 0→(C)
            LDA#$FF
            BCC *+4        ; skok, je-li (C)=0
            LDA#0
```

Spotřeba paměti 10 byte, času 12 taktů.

Řešení C: nejefektivnější lineární řešení

```
            LDA $CC
            CMP $CD        ; je-li (CC) (CD), pak 0→(C)
            SBC $CC        ; 0-(C̄)→(A)
```

Spotřeba paměti 6 byte, času 9 taktů.

Pozn.: Uvědomte si, že každé větvení programu je v podstatě řešení nějakého matematicko-logického problému, který lze velmi často řešit lineárním způsobem použitím matematicko-logických instrukcí.

8.1.6 Zásada omezení skokových instrukcí

Každá skoková instrukce znepřehledňuje řešení. Strukturované programování nepřipouští:

- bezprostředně skákat kamkoli do programu,
- skákat z jedné větve do druhé,

- skákat z jedné úroveně podprogramů na druhou,
- skákat doprostřed modulu, tedy předávat řízení jinak, než přes vstupní bod /body/ modulu,
- vyskočit uprostřed modulu /mimo výstupní bod/.

Pro JSA je povoleno použít skoku k:
- vyvedení programu ze slepé větve do společného uzlu větví,
- přemostění programu přes data,
- předání řízení jinému modulu.

8.1.7 Zásada návratu větví do jediného bodu

Jestliže je program rozvětven do několika větví, pak podle zásad strukturovaného programování se všechny větve vracejí do jediného společného bodu.

Není chybou, je-li každá větev podprogramu samostatně ukončena instrukcí RTS (nebo RTI u podprogramů zpracovávajících přerušení), kde společným bodem je adresa návratu.

8.1.8 Zásada přehledné organizace dat

Analýza úlohy a návrh datových struktur patří k nejdůležitějším momentům v programování.

Data organizujeme přehledně podle významu, určení a typu. Data společného typu řadíme k sobě. Například ukazatele polí, proměnné, konstanty, pracovní prostory, tabulky, vektory, zprávy, ...

Podle významu rozlišujeme:
- data lokálního významu,
- data globálního významu.

Lokální data mají význam pouze pro příslušný modul. Proto se nejčastěji umisťují uvnitř modulu nebo v jinak vymezeném prostoru.

Globální data jsou přístupná celému programu. Umisťují se buď uvnitř programu nebo mimo prostory programu, avšak vždy na takovém místě, aby případná modifikace programu nenarušila jejich strukturu. Viz, například ekvivalentní datové struktury u různých verzí programu CENTRONICS ve "Sbírce řešených úloh v JSA 6502" /lit 14/

Data mohou být organizována staticky, kdy prostor je předem určen a vymezen, nebo dynamicky, kdy poloha a především velikost prostoru jsou dimenzovány podle aktuálních potřeb.

8.1.9 Zásada přiměřeného a výstižného komentování

Ke komentování lze přistupovat různě. Například známý profesionální programátor mikroprocesoru 6502 Hans Wagner nekomentuje své programy vůbec. A přesto, díky strukturovanému řešení a vhodné volbě názvů návěští, jsou jeho programy v JSA jasné a srozumitelné. Naproti tomu programátoři firmy ATARI komentují každou instrukci.

Je-li komentování příliš bohaté, je nebezpečí, že se v množství poznámek ztratí důležité informace. A je-li naopak příliš chudé či žádné, pak je velmi pravděpodobné, že programu za čas neporozumí ani sám autor.

Způsob komentování je závislý na tom, jak rozsáhlý a složitý je program, zda bude sloužit pouze autorovi nebo zda na něj bude navazovat někdo další.

Začátečníci by měli komentovat raději více než méně.

I v komentování je třeba dodržovat určité zásady. Vyhneme se komentářům, které popisují činnost instrukce, protože je známá a neměnná. Pokud ji programátor zapomene, může si ji oživit nahlédnutím do tabulek a manuálů. Takže místo:

```
LDA #$00      ; $00 ---> (A)
JSR TISKZNAK  ; volání podprogramu TISKZNAK
```

je lepší:

```
          LDA#$0D          ; řídicí kód: návrat vozíku
          JSR TISKZNAK     ; tisk (A) na obrazovku
```

nebo pomocí běžně používaných zkratek a symbolů:

```
          LDA#$0D          ; CR
          JSR TISKZNAK     ; (A)---->SCREEN
```

K popisu činnosti instrukce v komentáři se můžeme uchýlit, chceme-li zdůraznit některou druhotnou vlastnost instrukce.

```
          TAX              ; ovlivní (N), (Z)
          STA ALFA
          BEQ ZERO
```

Pokud chceme zvýraznit určitou skupinu instrukcí, můžeme použít technicky odsazení /identace/, nebo v poli komentáře použít stejný symbol, který upozorní, že instrukce patří k sobě. Například:

```
          LDA#0            ;& potlačení zpracování DMA
          STA $02FF        ;&
          STA $D400        ;&
```

Program můžeme komentovat pomocí syntaktického zápisu některého z vyšších jazyků. Nejlépe strukturovaného. Například Pascal, PL/1, kanadský Fortran atd. Program je pak čitelnější a srozumitelnější.

Příklad komentování jazykem PASCAL:

```
*          .......          REPEAT
REP       INC ALFA             ALFA : = ALFA + 1
          DEX                  x : = x - 1
          BNE REP          UNTIL X=0
```

8.1.10 Zásada členění programu

Při psaní programu bychom měli dodržovat zásadu, že společné informace patří k sobě. Například umisťujeme těsně k sobě definice jmen, deklarace proměnných, vektorů, polí a pod., definice makroinstrukcí, podprogramy atd.

Vždy bychom měli dodržovat zásadu: "Nejprve definuj, pak teprve použij".

Místo:

```
            LDA BETA
    BETA    EQU $37AC
```

což však není syntakticky chybně, je lepší přirozený postup:

```
    BETA    EQU $37AC
            LDA BETA
```

Takže struktura celého programu může vypadat následovně:

Makroinstrukce:

- definice makroinstrukcí nejnižší úrovně
- ...
- definice makroinstrukcí nejvyšší úrovně

Hlavní program:

- popis - záhlaví
- definice umístění strojového kódu: ORG
- definice globálních jmen: EPZ, EQU
- tabulka služeb: JMP
- datové struktury /tj. proměnné, konstanty, ukazatele, pole,.../: DFB, DFW, ORG *+N, ASC
- vlastní program složený z instrukcí JSA, volání podprogramů a makroinstrukcí nejvyšší /tj. první/ úrovně
- instrukce ukončující hlavní program: JMP, JMP (), JSR, BRK, RTS, RTI

Podprogramy:

- podprogramy nejvyšší úrovně
- ...
- podprogramy nejnižší úrovně

Vnitřní struktura podprogramů či makroinstrukcí může být podobná jako struktura hlavního programu.

Příklady členění programu jsou ve "Sbírce" - lit /14/.

8.1.11 Zásada popisu modulu

Platí zásada, že vnitřní činnost modulu /podprogramu nebo makroinstrukce/ je složitější, než se projevuje navenek.

Na modul můžeme pohlížet jako na černou schránku, kde nás nezajímá, jakým způsobem bylo dosaženo cíle, ale jak se projevuje navenek.

Z těchto principů vycházíme při popisu modulu. V záhlaví by se měly objevit tyto informace:

1/ výstižný název modulu
   například PREVOD a pod.

2/ stručná charakteristika činnosti či návod k použití modulu
   například Převod ASCII kódu na EBDIC nebo ISO-7

3/ popis vstupních informací
   například: I: ⟨A⟩ ... ASCII kód

4/ popis výstupních informací
   například: O: ⟨A⟩ ... EBDIC či ISO-7 kód

Dále je možné uvést:

5/ zda jde o makroinstrukci nebo podprogram
   například: MACRO PREVOD či SUBR. PREVOD

6/ úroveň vnoření modulu
   například v programu MOZART ve "Sbírce" - lit/14/.
   je úroveň vnoření podprogramu vyjádřena počtem hvězdiček v záhlaví

7/ seznam podmodulů, které modul používá; stačí uvádět jména modulů o jednu úroveň nižší, než je popisovaný modul
   například: EXTERNAL: SOUCET, ERROR

8/ seznam jmen vstupních bodů modulu
   například ENTRY POINT: ASCEBD, ASCISO

9/ seznam modulů, které popisovaný modul používají
   například: PUBLIC: ZPRAVA, ERRMESS

Jako je společným jazykem lékařů latina, je angličtina jazykem programátorů. Proto je možné popisovat moduly jak v národním, tak i v anglickém jazyce. Případně i kombinovaně, kdy symboly a návěští vyjadřujeme ustálenými anglickými slovy nebo zkratkami. Například CIO, INPUT, WRITE, LOOP, LABEL,... Komentáře pak píšeme v jazyce národním.

Příklad záhlaví modulu typu podprogram v češtině:

```
*================================================
* Podprogr. jméno: TISKTXT      úroveň: 3
* Činnost: Tisk textu na obrazovku
* Vstup: (SCC.D)..... ukazatel, poč. adr.
*        (SCE.F)..... počet znaků
* Výstup: obrazovka /Video RAM/, aktuální poloha kurzoru
* Volá: RIZENI, JEDENZN
*------------------------------------------------
TISKTXT     JSR RIZENI
            JSR JEDENZN
            ...
            RTS
*================================================
```

Totéž anglicky:

```
*-------------------------------------------------
* Subr. name: PRINTTXT     Level: 3
* Activity: Print text to screen
* Input: ($CC.D)..... pointer, start addr.
*        ($CE.F)..... number of characters
* Output: screen /Video RAM/, current possition of cursor
* External: CONTROL, ONECHAR
*-------------------------------------------------
PRINTTXT    JSR CONTROL
            JSR ONECHAR
            ...
            RTS
*-------------------------------------------------
```

8.1.12 Zásada čistého programování

V operačním systému je celá řada užitečných programů ve strojovém kódu, které může programátor využívat ve svých programech -viz. lit /8/.

Zásada čistého programování pak spočívá v tom, že tyto podprogramy /tzv. služby/ voláme dohodnutým způsobem, například přes tabulku vstupních bodů. Parametry předáváme přes vymezené lokace. Tím je zajištěna přenositelnost programového vybavení mezi verzemi osmibitových počítačů ATARI.

Operační systémy počítačů ATARI obsahují centrální tabulku vstupních bodů nejdůležitějších služeb systému. Vstupní body těchto služeb jsou u všech typů na stejných adresách /od adresy $E450 do $E48C včetně/. Využívá-li programátor pouze služeb uložených na těchto adresách, jsou jeho programy přenositelné mezi jednotlivými typy.

Tabulka služeb OS ATARI

Adr. vstup. bodu	ATARI 800 ver.B	ATARI 800XL 130XE ver.C	Význam
E450	JMP $EDEA	JMP $C6A3	Disk handler initiation routine
E453	JMP $EDF0	JMP $C6B3	Disk handler vector
E456	JMP $E4C4	JMP $E4DF	Central Input/Output vector
E459	JMP $E959	JMP $C933	Serial Input/Output vector
E45C	JMP $E8ED	JMP $C272	Set system timers routine vector
E45F	JMP $E7AE	JMP $C0E2	System vertical blank interrupt processing
E462	JMP $E9B5	JMP $C282	Exit from vertical blank processing
E465	JMP $E944	JMP $E95C	Serial Input/Output initialization
E468	JMP $EBF2	JMP $EC17	Serial bus send enable routine
E46B	JMP $E6D5	JMP $C00C	Interrupt handler routine
E46E	JMP $E4A6	JMP $E4C1	Central Input/Output initialization
E471	JMP $F223	JMP $F223	Blackboard mode vector to memo pad mode
E474	JMP $F11B	JMP $C290	Warm start entry /follows SYSTEM RESET/
E477	JMP $F125	JMP $C2C8	Cold start entry point /follows power-up/
E47A	JMP $EFE9	JMP $FD8D	Cassette read block routine vector
E47D	JMP $EF5D	JMP $FCF7	Cassette open for input vector

TAB. 79 Tabulka služeb OS ATARI

9. Přerušení

K přerušení běhu programu může dojít z mnoha důvodů. Jak pravidelně se opakujících tak i nepravidelných.

Princip přerušení spočívá v tom, že na základě nějakého podnětu dojde k žádosti o přerušení. Je-li jí vyhověno, právě probíhající program je přerušen a spustí se program obsluhující přerušení. Po skončení obsluhy přerušení se řízení vrací do přerušeného programu. Výjimkou je přerušení typu RESET.

U mikrorprocesoru 6502 lze vyvolat tři typy hardwarového /IRQ, NMI, RESET/ a jeden typ softwarového /BRK/ přerušení:

1/ HW maskovatelné přerušení IRQ /Interrupt Request/ /vektor $FFFE.F/ /pro OS dále $216.7/,
2/ HW nemaskovatelné přerušení NMI /Non Maskable Interrupt/ /vektor $FFFA.B/,
3/ HW inicializační přerušení RESET /vektor $FFFC.D/,
4/ SW programové přerušení instrukcí BRK /vektor $FFFE.F/ /pro OS dále $216.7/.

Nejnižší prioritu má signál IRQ, nejvyšší RESET. Přijdou-li tedy dva nebo tři požadavky na přerušení najednou, je přednostně zpracováno přerušení s vyšší prioritou.

Zdrojem signálu přerušení může být u ATARI 800XL a ATARI 130XE obvod uvnitř mikropočítače nebo externí signál přivedený z vnějšku přes konektor systémové sběrnice /ATARI XL/, či přes konektor ECI /ATARI XE/ nebo přes konektor sériové sběrnice /ATARI XL/XE/.

9.1 Zdroje přerušení IRQ, NMI, RESET

Zdrojem signálu IRQ může být:

1/ obvod POKEY,
2/ obvod PIA 6520 /na základě vnějšího podnětu; pin 9 - Procced a pin 13 - Interrupt na konektoru seriové

sběrnice u ATARI XL/XE/

3/ externí zdroj /u ATARI XL: pin 35 na konektoru systémové sběrnice a u ATARI XE: pin 8 na konektoru ECI/

Důvodem, proč obvod POKEY generuje signál $\overline{IRQ}$, je žádost o zpracování přerušení vyvolaná:

a/ stisknutím klávesy BREAK - (b_7)
/vektor $236.7/,

b/ stisknutím libovolné klávesy kromě START, OPTION, SELECT, RESET - (b_6)
/vektor $208.9/,

c/ seriovým přijímačem - (b_5)
/vektor $20A.B/,

d/ seriovým vysílačem - (b_4)
/vektor $20C.D/,

e/ ukončením seriového přenosu - (b_3)
/vektor $20E.F/,

f/ časovačem TIMER3 - (b_2)
/vektor $214.5/,

g/ časovačem TIMER2 - (b_1)
/vektor $212.3/,

h/ časovačem TIMER1 - (b_0)
/vektor $210.1/.

Jednotlivé důvody přerušení lze maskovat /tj. potlačit, zakázat/ nastavením příslušného bitu adresy $10(↑ $D20E) na hodnotu log. 0.

Důvodem, proč obvod PIA generuje signál $\overline{IRQ}$, je žádost o zpracování přerušení vyvolaná:

a/ signálem $\overline{IRQA}$
/vektor $202.3/,

b/ signálem $\overline{IRQB}$
/vektor $204.5/.

Zdrojem signálu $\overline{NMI}$ je obvod ANTIC. Důvodem generování signálu $\overline{NMI}$ může být:

a/ VBI /Vertical Blank Interrupt/ - žádost o zpracování přerušení po vykreslení jednoho televizního snímku v době tzv. vertikálního zatmění /pro normu PAL každých 20 ms/ - (b_7)
/vektor $222.3/,

b/ DLI /Display List Interrupt/ - žádost o zpracování přerušení programem procesoru ANTIC - (b_6)
/vektor $200.1/

Tyto důvody lze potlačit nastavením příslušného bitu adresy $D40E na hodnotu log. 0.
Nezaměňovat s řízením DMA $22F, $D400.

Zdrojem signálu RESET může být:

a/ zapnutí zdroje počítače,

b/ stisknutí klávesy RESET,

c/ externí zdroj.

9.2 Maskovatelné přerušení IRQ

Je-li signál IRQ aktivní /log. 0/, znamená to, že některý zdroj signálu žádá o přerušení. Pak mikroprocesor testuje příznakový bit masky přerušení I.

Je-li ⟨I⟩=1, pak žádosti o přerušení není vyhověno a mikroprocesor pokračuje dál ve své činnosti další instrukcí probíhajícího programu.

Je-li ⟨I⟩=0 /přerušení je povoleno/, pak mikroprocesor zahájí přerušovací sekvenci:

1/ Dokončí poslední zpracovávanou instrukci.

2/ Přeruší právě probíhající program.

3/ Do zásobníkové paměti uloží návratovou adresu, tj. adresu příští instrukce /v pořadí PCH, PCL/.

4/ Do zásobníkové paměti uloží obsah příznakového registru P /příznakový bit I obsahuje log. 0!/.

5/ Nastaví příznakový bit I na log. 1, čímž zamezí případnému dalšímu zpracování žádosti o přerušení IRQ.

6/ Naplní programový čítač PC následujícím způsobem: ⟨$FFFE⟩→⟨PCL⟩, ⟨$FFFF⟩→⟨PCH⟩. To znamená, že spustí program, jehož počáteční adresa /vektor/ je uložena na posledních dvou buňkách paměti.

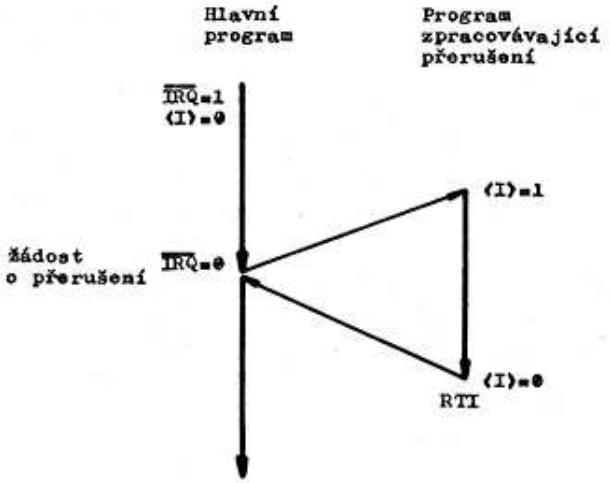
Je-li přítomna paměť ATARI OS ROM 16 KB, pak program pokračuje společnou větví /přes vektor $216.7/, ve které se zjišťuje zdroj a důvod přerušení. Podle toho pak dojde k větvení. Jednotlivé větve pokračují na adrese, která je uložena na buňkách příslušného vektoru /podle důvodu přerušení - viz kap. 9.1/.

Každá větev ošetřující rutiny je ukončena instrukcí RTI, která způsobí:

a/ vybrání příznakového registru P ze zásobníku ⟨I⟩=0, takže je povoleno zpracování další žádosti o přerušení/,

b/ vybrání návratové adresy ze zásobníku PCL, PCH a uložení do programového čítače.

Po této instrukci se program vrací z rutiny ošetřující přerušení a pokračuje v původním programu.



obr. 37 - Princip přerušení IRQ

9.3 Nemaskovatelné přerušení NMI

Je-li signál NMI aktivní /log. 0/, pak mikroprocesor zahájí přerušovací sekvenci bez ohledu na stav příznakového bitu I:

1/ Dokončí poslední zpracovávanou instrukci.
2/ Přeruší právě probíhající program.
3/ Do zásobníku uloží návratovou adresu v pořadí PCH, PCL.
4/ Do zásobníku uloží obsah příznakového registru P.
5/ Nastaví příznakový bit I na log. 1.
6/ Naplní obsah programového čítače PC následujícím způsobem: ⟨$FFFA⟩→⟨PCL⟩, ⟨$FFFB⟩→⟨PCH⟩. Spustí se tedy program, jehož vektor spuštění je na adrese $FFFA.B.

Je-li aktivní ATARI OS ROM 16 KB, pak program pokračuje společnou větví a teprve dále se větví podle důvodu přerušení.

Například VBI je generováno každých 20 ms. Program zpracovávající toto přerušení zajišťuje mimo jiné přenos informací mezi HW registry obvodů ANTIC, GTIA, POKEY, PIA na adresách $D000 až $D7FF a datovými strukturami /tzv. systémovými proměnnými/ organizovanými v paměti RAM. Tedy každých 20 ms HW registry přehrávají aktuální informace do tzv. sledujících systémových proměnných /například o poloze joysticku, o poloze světelného pera, o A/D převodu a pod./. A obráceně, ze systémových proměnných se přenášejí informace do sledujících HW registrů /například o barvě a jasu obrazu, o síle, výšce a zkreslení zvuku a pod./.

Přerušení VBI může být používáno pro práci v reálném čase.

Pomocí vektoru $222.3 můžeme celou systémovou rutinu zaměnit za uživatelskou rutinu. Pak je samozřejmě na uživateli, do jaké míry zabezpečí výměnu informací mezi HW registry a datovými strukturami.

Pomocí tzv. post vektoru $224.5 můžeme systémovou rutinu rozšířit o uživatelskou část. Tento postup se používá

častěji. Po ukončení uživatelské části můžeme s výhodou použít rutiny EXITVBL /tj. JMP $E462/, která vrátí původní hodnoty registrům mikroprocesoru a vyrovná zásobník, a instrukce RTI, která, podobně jako u přerušení typu IRQ, vrací řízení do přerušeného programu.

Pokud by byl celkový čas zpracování obslužné rutiny VBI delší, než je interval mezi jednotlivými požadavky na přerušení, došlo by ke zhroucení systému zacyklením programu. Uvádí se, že by uživatelská část programu měla být kratší než 3000 instrukcí.

Nastavení a zrušení uživatelského post vektoru $224.5, ale i vektoru $222.3 a programových časovačů nelze provést kdykoli. Tuto část je nutno synchronizovat s přerušením VBI. Jinak vzniká nebezpečí tzv. programového hazardu. Ten například nastane, dojde-li mezi instrukcí nastavení nižšího byte post vektoru STA $0224 a instrukcí nastavení vyššího byte post vektoru STA $0225 k přerušení VBI. K synchronizaci nám poslouží rutina SETVBL /tj. JSR $E45C/, kde do akumulátoru připravíme parametr určující vektor či časovač, který budeme nastavovat. Do registru X uložíme vyšší byte adresy a do registru Y nižší byte adresy.

Příklad:

Na adrese MAINPRG=$5000 je uložen hlavní program, jehož součástí je nainstalování post vektoru uživatelské rutiny VBI, která je uložena na adrese USRHW20=$6000 a je volána každých 20 ms.

```
EXITVBL     EQU $E462
SETVBL      EQU $E45C
MAINPRG     EQU $5000
USRHW20     EQU $6000
            ORG MAINPRG
            ...             ; instrukce hlavního programu
            LDA#7           ; % nastavení vektoru $224.5
            LDX#USRHW20:L   ; %
            LDY#USRHW20:H   ; %
            JSR SETVBL      ; % instalační rutina
```

```
...             ; instrukce hlavniho programu
RTS             ; návrat do monitoru /pro
                  ATMAS-II/

OGR USRHW2Ø
...             ; instrukce uživatelské rutiny
...             ; zpracovávající přerušení VBI
...             ; která je provedena bezpro-
                  středně
...             ; po systémové rutině VBI
JMP EXITVBL     ; zajistí návrat do přerušeného
                ; programu
```

Konkrétní využití tohoto přerušení je ukázáno ve "Sbírce" v příkladu MOZART.

9.4 Inicializační přerušení RESET

Je-li signál RESET aktivní /log. Ø/, pak je vyvoláno přerušení, které není nijak maskovatelné /ani příznakem I/. Během doby trvání úrovně log. Ø je v mikroprocesoru blokována funkce čtení a zápisu. Je nutné, aby signál RESET byl aktivní nejméně 2 takty /při 1,79 MHz je 1 takt zhruba 66Ø nanosekund/.

Při přechodu signálu RESET z log. Ø do log. 1 je generována vnitřní inicializační přerušovací sekvence. Trvá 6 taktů, nastaví výchozí stav mikroprocesoru.

Přerušovací sekvence:

1/ přeruší právě probíhající program,

2/ nastaví příznakový bit I na log. 1,

3/ naplní obsah programového čítače PC následujícím způsobem:

⟨$FFFC⟩→⟨PCL⟩, ⟨$FFFD⟩→⟨PCH⟩

Jelikož sekvence neuchovává návratovou adresu, není řízení vráceno do místa přerušení, jako je tomu u ostatních druhů přerušení.

Je-li aktivní ATARI OS ROM 16 KB, pak je spuštěn program, který testuje, zda má probíhat tzv. studený start, tj. stav po spuštění zdroje, nebo tzv. teplý start, tj. stav

vyvolaný stisknutím tlačítka RESET či externím zdrojem ze systémové sběrnice za provozu počítače.

Program studeného startu mimo jiné provádí:

- mikrotest paměti,
- nastavení datových struktur operačního systému /to jsou systémové proměnné, konstanty, vektory, tabulky, příznaky, ukazatele, buffery, čítače, program pro ANTIC - tzv. Display List, VíčeoRAM, atd./,
- inicializaci a nastavení režimů ostatních obvodů v mikropočítači /ANTIC, GTIA, POKEY, PIA/,
- inicializaci všech aktivních periferií; je-li aktivní disketová jednotka, zavádí se DOS,
- inicializaci cartridge, je-li zapojena do systému,
- předání řízení /dle volby/; není-li zapojena Cartridge ROM a nebylo-li stisknuto tlačítko START a OPTION, je řízení předáno BASIC ROM 8 KB, kde dochází k instalaci dalších datových struktur a k průběhu dalších inicializačních rutin.

Program teplého startu provádí redukovanou činnost studeného startu. Staví pouze některé datové struktury, provádí některé instalace a inicializace.

Pomocí příslušných vektorů v datových strukturách je možné se "vrátit" do přerušeného programu.

10. Ukázka překladu programu do strojového kódu a připojení k Basicu

Sestavte program v JSA, který při každém spuštění změní v grafickém režimu 0 barvu a odstín písma /adresa systémové proměnné 709, tj. $2C5/ za barvu a odstín pozadí /adresa systémové proměnné 710, tj. $2C5/. Program odlaďte a přeložte do strojového kódu. Odladěný a přeložený program pak připojte k programu v Basicu.

Na tomto příkladu je prakticky předvedena práce s programem ATMAS-II a spojení programu ve strojovém kódu s programem v Basicu.

Postup:

1/ Zaveďte program ATMAS-II /START+OPTION/ pro zavaděč BLC 2.0, stlačte klávesu A a Return.

2/ Po nahrání ATMAS-II se nacházíte v editovacím režimu. Můžete přímo zapsat následující zdrojový text programu:

```
            OUT LN
            ORG $600
*ZAMENA COL1 s COL2
COL1        EQU $2C5        ; 709
COL2        EQU $2C6        ; 710

BASIC       PLA             ; vstup pro Basic
MONITOR     LDA COL1        ; vstup pro monitor
            LDX COL2
            STA COL2
            STX COL1
            RTS
```

3/ Přeložte program do strojového kódu stisknutím CTRL-Y. Je-li v programu syntaktická chyba, překladač vypíše chybové hlášení. Stisknutím libovolné klávesy se vraťte do editoru, opravte chybu a opakujte překlad.

Není-li v programu syntaktická chyba, pak překladač vypíše zprávu o překladu /listing/. Výpis můžete pozastavovat stisknutím CTRL-1.

```
                        ORG $600

                * ZAMENA COL1 s COL2

                COL1    EQU 709         ; $2C5
                COL2    EQU 710         ; $2C6

0600:68         BASIC   PLA             ; vstup
0601:ADC502     MONITOR LDA COL1        ; vstup
0604:AEC602             LDX COL2
0607:8DC602             STA COL2
060A:8EC502             STX COL1
060D:60                 RTS

PHYSICAL ENDADDRESS:$060E

COL1            $02C5
COL2            $02C6
BASIC           $0600           UNUSED
MONITOR         $0601           UNUSED
```

V horní části obrazovky vlevo jsou logické /v našem případě i skutečné, tj. fyzické/ adresy překládaných instrukcí JSA. Za oddělovací dvojtečkou je strojový kód instrukcí. Adresy i strojový kód jsou v hexadecimálním tvaru. Vedle adres a strojového kódu je zdrojový text JSA. /V našem případě MONITOR je návěští, LDA instrukce, COL1 operand, ; vstup je komentář./ Zprávou PHYSICAL ENDADDRESS:$060E nás překladač informuje o první volné adrese. Pod touto zprávou je seznam použitých symbolických jmen, návěští a jejich skutečné hodnoty v hexadecimálním tvaru. Zprávou UNUSED /nevyužíváno/ nás překladač informuje o tom, že příslušné návěští /v našem

případě BASIC, MONITOR/ nejsou v programu využívána. Není na ně v programu orientován žádný odkaz. Mají pouze informační význam.

4/ Stiskněte libovolnou klávesu, řízení se vrátí do editoru.

5/ Nahrajte zdrojový text JSA na mg kazetu /disketu/ pro možnou budoucí modifikaci. Je to zároveň pojistka proti případnému zhroucení systému při odstartování chybného strojového programu.
Nejprve umístěte kurzor na začátek zdrojového textu pomocí CTRL-E. Potom zadejte sekvenci: [ESC][W][C][;][ESC][ESC]
Stiskněte PLAY + WRITE na magnetofonu a potvrďte klávesou Return. Zdrojový text se zapíše na pásku.
O ukončení činnosti nás systém informuje hlášením:
$WC:$ # .... v povelovém řádku.
Kurzor je přemístěn na konec zdrojového textu.

6/ Předejte řízení monitoru stisknutím CTRL-P. Monitor se ohlásí výpisem MONITOR. Zkontrolujte správnost uložení strojového programu implementovaným zpětným překladačem stisknutím klávesy D. Monitor vypíše:
DISASEMBLER
START:

Napište počáteční adresu překladu v hexadecimálním tvaru, tedy: 0600. Na dotaz PRINT? odpovězte N nebo stisknutím klávesy Return.
Systém vypíše první obrazovku disasemblovaného programu:

```
0600:68          PLA
0601:ADC502      LDA $2C5
0604:AEC602      LDX $2C6
0607:8DC602      STA $2C6
060A:8EC502      STX $2C5
060D:60          RTS
060E:00          BRK
060F:00          BRK
   .               .
   .               .
   .               .
```

Opište si strojový kód pro budoucí použití v programu napsaného v Basicu /od aresy $0600 do $060D včetně/. Stiskněte klávesu X,. Řízení je vráceno MONITORu. Ověřte správnost programu jeho spuštěním. Stiskněte klávesu G. Monitor vypíše GOTO a čeká na zadání čtyř znaků hexadecimální adresy startu. Napište 0601.
Po stisknutí posledního znaku adresy se ihned spustí strojový program od adresy 0601. Pokud je program v pořádku, provede se záměna barvy pozadí a textu. Jelikož je program ukončen instrukcí RTS, vrací se řízení do monitoru.
Zopakováním GOTO 0601 uvedeme barvy do původního stavu.

7/ Zapište strojový kód /binární obraz paměti od adresy $0600 do adresy $060D včetně/ příkazem S v monitoru. Vyplňte následujícím způsobem:

```
SAVE
FROM:0600
TO  :060D
INŤO:0600
FILENAME (D:FN.EXT)? C:      (Return)
```

Po akustickém signálu stiskněte PLAY+WRITE na magnetofonu, potvrďte Returnem.
Na mg. kazetě je vytvořen datový soubor se záhlavím:
FF,FF,00,06,0D,06.
Pak následují data strojového kódu:
68,AD,C5,02,.....,60.
Soubor je v normálním tvaru, tzn. s dlouhými mezibloky mezerami (AUX2=0).

8/ Předejte řízení interpretu BASICu postupem:
GOTO E477 nebo GOTO E471 a Reset.

9/ V Basicu napište a spusťte následující program.
Po akustickém signálu nastavte mg. pásku na začátek datového souboru, stiskněte PLAY a Return. Řádek:

40 ..... otevře kanál#1 na čtení souboru z mg kazety v povelovém režimu

60-80 ..... přečte a ignoruje prvních 6 byte nahrávky /tzv. header - záhlaví/

110-130 ..... přečte a zavede za začátek volné šesté stránky strojový kód

140 ..... uzavře kanál#1

200 ..... odstartuje strojový kód, který vymění barvu pozadí a textu, pak vrací řízení do Basicu.

Příkazem GOTO 200 docílíme původního zobrazení barev.

```
1     REM Inverze barev RESENI 1
10    REM LOADER PRO ZAMENU COL1 S COL2
20    BEG=1536:REM $600 ... POC. ADR.
30    PND=1549:REM $060D ... KON. ADR.
40    OPEN#1,4,0,"C:"
50    REM * CTENI 6 BYTE ZAHLAVI
60    FOR I=1 TO 6
70    GET#1,BYTE
80    NEXT I
100   REM * CTENI A ZAVEDENI STROJOVEHO KODU
110   FOR ADRESA=BEG TO PND
115   GET#1,BYTE
120   POKE ADRESA,BYTE
130   NEXT ADRESA
140   CLOSE#1
200   X=USR(BEG)
999   END
```

Zhodnocení ŘEŠENÍ 1:

Použitý postup je vhodný pro zavádění středně dlouhých programů ve strojovém kódu, cca 1/4 KB až 4 KB (1/4 KB = 1 stránka, tj. 256 byte).

Výhoda spočívá v tom, že se strojovým kódem nepřijde programátor do styku. Pohybuje se na úrovni JSA a Basicu.

Nevýhoda je do jisté míry v tom, že již předem musíme znát prostor /rozsah adres/, kam bude strojový kód směrován. Do tohoto prostoru nesmí zasahovat ani Basic ani systém /např. Video RAM/. Potřebný prostor se zpravidla uvolňuje snížením ukazatele vrcholu volné paměti RAM. Například postupem:

```
RAMTOP=106:A=PEEK(RAMTOP):A=A-4*KB:POKE RAMTOP,A:GR.0,
```

kde KB je počet rezervovaných KByte.
Popřípadě zvýšením ukazatele spodní hranice volné paměti RAM. To je třeba provést před instalací interpretu Basic, resp. jeho datových struktur /text. buffer a pod./. Jako je tomu například v samostartující rutině u programu CENTRONICS V2.1 uvedené ve "Sbírce".

Programy o délce 2 KB až 8 KB /16 KB/ je vhodné přetransformovat na mg. kazetě na tvar s krátkými meziblokovými mezerami /AUX2=$80/. K tomu nám poslouží soubor makroinstrukcí MCIO uvedený ve "Sbírce". Pak je nutné tento program zavádět LOADERem ve strojovém kódu, který je umístěn do Basic programu, například způsobem jako v ŘEŠENÍ 4. Tím se podstatně urychlí zavádění strojového programu z mg. kazety.

10/ Převeďte strojový kód z hexadecimálního tvaru, který jsme si opsali z výpisu překladu nebo který jsme pomocí tabulek z přílohy "ručně" přeložili, do decimálního tvaru a do ATASCII znaků pro další zpracování.

HEX.	DEC.	ATASCII	INSTRUKCE
68	104	h	PLA
AD	173	-	LDA $02C5
C5	197	E	
02	2	H	
AE	174	.	LDX $02C6
C6	198	F	
02	2	H	
8D	141	M	STA $02C6
C6	198	F	

```
02      2
8E    142        STX $02C5
C5    197
02      2
60     96        RTS
```

```
1   REM Inverze barev ŘEŠENÍ 2
10  PAGE6=1536
20  FOR I=PAGE6 TO PAGE6+13
30  READ A:POKE I,A
40  NEXT I
50  X=USR(PAGE6)
60  DATA 104,173,197,2,174,198,2
70  DATA 141,198,2,142,197,2,96
```

Zhodnocení ŘEŠENÍ 2:

Tento postup se používá pro krátké programy ve strojovém kódu /do 1 KB/. Má tyto nevýhody.

1/ Programátor musí "ručně" vložit strojový kód do dat.

2/ Spotřeba paměti je více než dvojnásobná, jelikož je strojový kód uložen jednak neúsporným způsobem v datech v textovém bufferu Basicu /viz ř. 60-70/, jednak v místě zavedení /viz ř. 10-40/.

3/ Zavádění strojového kódu z dat na příslušné místo v paměti trvá poměrně dlouho /1 KB strojového kódu trvá cca 10 sec/.

I přes své nevýhody je programátory často používán, jelikož je nenáročný, poměrně snadno kontrolovatelný, nevyžaduje dvojí nahrávání z mg. kazety.

```
1   REM Inverze barev RESENI 3
10  CLR:DIM INVERZE$(14)
20  INVERZE$="h-E!.F!‾F!_E!◆"
30  X=USR(ADR(INVERZE$))
```

Zhodnocení ŘEŠENÍ 3:

Tento postup se používá pro velmi krátké programy ve strojovém kódu /do 1/4 KB/. Je úspornější, zavedení strojového kódu je mnohem rychlejší než ŘEŠENÍ 2. Vyžaduje dvojnásobný prostor než je délka strojového programu, jelikož je strojový kód uložen v textovém bufferu Basicu /viz Ř. 20/ a také je uložen v datových strukturách /v tomto případě v řetězcové proměnné/ Basicu /viz Ř. 10-20/.

Hlavní nevýhoda spočívá v tom, že programátor musí strojový kód ručně překódovat do ATASCII znaků, program je těžko dokumentovatelný, strojový kód musí být přemístitelný, tj. startovatelný z libovolného místa paměti. To vede k jistému omezení programátora při výběru instrukcí.

```
1   REM Inverze barev RESENI 4
10  INVERZE=ADR("hEI.FI FI EI◆")
20  X=USR(INVERZE)
```

Zhodnocení ŘEŠENÍ 4:

Je to nejúspornější postup zavedení strojového kódu. Strojový kód je uložen pouze jedenkrát v paměti - v textovém bufferu Basicu /viz Ř. 10/. I zdrojový text Basic programu je oproti ŘEŠENÍ 1, 2 mnohonásobně kratší. Jinak platí totéž, co bylo uvedeno u ŘEŠENÍ 3.

Použitá literatura

/1/ De Jong, M., L.:
Programming Interfacing the 6502 With Experiments,
Howard, W., Sams Co.,Inc.,
Indianopolis, Indiana, USA, 1980

/2/ Zaks, R.:
Programming the 6502,
Sybex, Inc., USA, 1978

/3/ Leventhal, A., L.:
Microcomputer Experimentation With the MOS Technology
KIM-1,
Prentice-Hall, Inc., USA, 1982

/4/ Chasin, M.:
Assembly Language Programming for the ATARI Computers,
McGraw-Hill, Inc., USA, 1984

/5/ Douglas, W., M.:
Apple Assembly Language a Course of Study Based on
LazerWare Software,
Computer Science Press, Inc., USA, 1984

/6/ An Introduction to Microprocessor Systems,
Macmillan Education, Ltd., London, GB, 1985

/7/ Poole, L.; McNiff, M.; Cook, S.:
Your ATARI Computer,
McGraw Hill, Inc., Berkeley, CA., USA, 1982

/8/ Eichler, Grohmann:
Intern,
Data Backer, Düsseldorf, NSR, 19

/9/ Profibuch, NSR (překlad bez bližších údajů).

/10/ Fajta, V.:
Řešené příklady v jazyku symbolických adres mikroprocesoru 8080,
Katedra jazyků, VŠST Liberec, 1987

/11/ Fajta, V.:
Sbornik přednášek - programování mikroprocesoru 650?,
Vývoj a projektování, ORGATEX, Hradec Králové, 1987

/12/ Hofacker, W.:
Program Descriptions I,
ELCOMP Publishing, Inc., Pomona, CA., USA, 1982

/13/ Petrov, P.:
Proektirane na sistemi, programirane na asembler i eksperimentirane s 6502
Rádio, televize, elektronika, časopis BLR, ročnik 1986

/14/ Fajta, V.:
Sbirka řešených úloh v jazyku symbolických adres mikroprocesoru 6502
487. ZO Svazarmu - ATARI KLUB, Praha, 1987

Příloha 1

PŘEHLED INSTRUKCÍ MIKROPROCESORU 6502

FAKTA v.

Poř. číslo	Inst-rukce	Prováděcí schéma	Způsob adresace	Operand	Počet period	Oper. kod	Počet byte	Příznaky NV.BDIZC	Čas. sch.
1 a	ADC	(A)+(M)+(C)→(A)	IMM	#OP8	2	69	2	NV....ZC	
b			ZP	OP8	3	65	2		
c			ZP,X	OP8,X	4	75	2		
d			ABS	OP16	4	6D	3		
e			ABS,X	OP16,X	4	7D	3		
f			ABS,Y	OP16,Y	4	79	3		
g			(IND,X)	(OP8,X)	6	61	2		
h			(IND),Y	(OP8),Y	5	71	2		
2 a	AND	(A) and (M)→(A)	IMM	#OP8	2	29	2	N.....Z.	
b			ZP	OP8	3	25	2		
c			ZP,X	OP8,X	4	35	2		
d			ABS	OP16	4	2D	3		
e			ABS,X	OP16,X	4	3D	3		
f			ABS,Y	OP16,Y	4	39	3		
g			(IND,X)	(OP8,X)	6	21	2		
h			(IND),Y	(OP8),Y	5	31	2		
3 a	ASL		ACC	A	2	0A	1	N.....ZC	
b			ZP	OP8	5	06	2		
c			ZP,X	OP8,X	6	16	2		
d			ABS	OP16	6	0E	3		
e			ABS,X	OP16,X	7	1E	3		

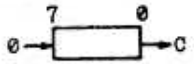
Poř. číslo	Inst-rukce	Prováděcí schéma	Způsob adresace	Operand	Počet period	Oper. kód	Počet byte	Příznaky NV.BDIZC	Čas. sch.
4	BCC	skok, jestliže (C)=0	REL	OP8	2 ··	90	2	.....	
5	BCS	skok, jestliže (C)=1	REL	OP8	2 ··	B0	2	.....	
6	BEQ	skok, jestliže (Z)=1	REL	OP8	2 ··	F0	2	.....	
7 a	BIT	(A) and (M)	ZP	OP8	3	24	2	M_7M_6....Z.	
b		$(M_7)\rightarrow(N)$	ABS	OP16	4	2C	3		
		$(M_6)\rightarrow(V)$							
8	BMI	skok, jestliže (N)=1	REL	OP8	2 ··	30	2	.....	
9	BNE	skok, jestliže (Z)=0	REL	OP8	2 ··	D0	2	.....	
10	BPL	skok, jestliže (N)=0	REL	OP8	2 ··	10	2	.....	
11	BRK	$1\rightarrow(B)$	IMPL		7	00	1	...1....	
		$(PL)+1\rightarrow(PC)$							
		$(PCH)\rightarrow(TOP)$							
		$(S)-1\rightarrow(S)$							
		$(PCL)\rightarrow(TOP)$							
		$(S)-1\rightarrow(S)$							
		$(P)\rightarrow(TOP)$							
		$(S)-1\rightarrow(S)$							
		$(\$FFFE)\rightarrow(PCL)$							
		$(\$FFFF)\rightarrow(PCH)$							

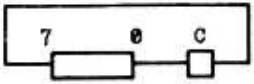
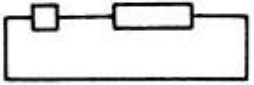
Poř. číslo	Inst- rukce	Provádějící schéma	Způsob adresace	Operand	Počet period	Oper. kod	Počet byte	Příznaky NV.BDIZC	Čas. sch.
12	BVC	skok, jestliže (V)=0	REL	OP8	2	50	2	.....	
13	BVS	skok, jestliže (V)=1	REL	OP8	2	70	2	.....	
14	CLC	0→(C)	IMPL		2	18	1	0	
15	CLD	0→(D)	IMPL		2	D8	1	....0...	
16	CLI	0→(I)	IMPL		2	58	1	.....0..	
17	CLV	0→(V)	IMPL		2	B8	1	.0.....	
18 a	CMP	(A)-(M)	IMM	#OP8	2	C9	2	N.....ZC	
b			ZP	OP8	3	C5	2		
c			ZP,X	OP8,X	4	D5	2		
d			ABS	OP16	4	CD	3		
e			ABS,X	OP16,X	4	DD	3		
f			ABS,Y	OP16,Y	4	D9	3		
g			(IND,X)	(OP8,X)	6	C1	2		
h			(IND),Y	(OP8),Y	5	D1	2		
19 a	CPX	(X)-(M)	IMM	#OP8	2	E0	2	N.....ZC	
b			ZP	OP8	3	E4	2		
c			ABS	OP16	4	EC	3		

Poř. číslo	Inst-rukce	Provádějící schéma	Způsob adresace	Operand	Počet period	Oper. kod	Počet byte	Příznaky NV.BDIZC	Čas. sch.
20 a	CPY	(Y)-(M)	IMM	#OP8	2	C0	2	N.....ZC	
b			ZP	OP8	3	C4	2		
c			ABS	OP16	4	CC	3		
21 a	DEC	(M)-1→(M)	ZP	OP8	5	C6	2	N.....Z.	
b			ZP,X	OP8,X	6	D6	2		
c			ABS	OP16	6	CE	3		
d			ABS,X	OP16,X	7	DE	3		
22	DEX	(X)-1→(X)	IMPL		2	CA	1	N.....Z.	
23	DEY	(Y)-1→(Y)	IMPL		2	88	1	N.....Z.	
24 a	EOR	(A)xor(M)→(A)	IMM	#OP8	2	49	2	N.....Z.	
b			ZP	OP8	3	45	2		
c			ZP,X	OP8,X	4	55	2		
d			ABS	OP16	4	4D	3		
e			ABS,X	OP16,X	4 *	5D	3		
f			ABS,Y	OP16,Y	4 *	59	3		
g			(IND,X)	(OP8,X)	6	41	2		
h			(IND),Y	(OP8),Y	5 *	51	2		

Poř. číslo	Inst-rukce	Provádějící schéma	Způsob adresace	Operand	Počet period	Oper. kód	Počet byte	Příznaky NV.BDIZC	Čas. sch.
25 a	INC	(M)+1→(M)	ZP	OP8	5	E6	2	N.....Z.	
b			ZP,X	OP8,X	6	F6	2		
c			ABS	OP16	6	EE	3		
d			ABS,X	OP16,X	7	FE	3		
26	INX	(X)+1→(X)	IMPL		2	E8	1	N.....Z.	
27	INY	(Y)+1→(Y)	IMPL		2	C8	1	N.....Z.	
28 a	JMP	OP16L→(PCL)	ABS	OP16	3	4C	3	.....	
b		OP16H→(PCH)	IND	(OP16)	5	6C	3		
29	JSR	(PC)+2→(PC) (PCH)→(TOP) (S)-1→(S) (PCL)→(TOP) (S)-1→(S) OP16L→(PCL) OP16H→(PCH)	ABS	OP16	6	20	3	.....	

Poř. číslo		Inst-rukce	Provádějící schéma	Způsob adresace	Operand	Počet period	Oper. kod	Počet byte	Příznaky NV.BDIZC	Čas. sch.
30	a	LDA	(M)→(A)	IMM	#OP8	2	A9	2	N.....Z.	
	b			ZP	OP8	3	A5	2		
	c			ZP,X	OP8,X	4	B5	2		
	d			ABS	OP16	4	AD	3		
	e			ABS,X	OP16,X	4 *	BD	3		
	f			ABS,Y	OP16,Y	4 *	B9	3		
	g			(IND,X)	(OP8,X)	6	A1	2		
	h			(IND),Y	(OP8),Y	5 *	B1	2		
31	a	LDX	(M)→(X)	IMM	#OP8	2	A2	2	N.....Z.	
	b			ZP	OP8	3	A6	2		
	c			ZP,Y	OP8,Y	4	B6	2		
	d			ABS	OP16	4	AE	3		
	e			ABS,Y	OP16,Y	4 *	BE	3		
32	a	LDY	(M)→(Y)	IMM	#OP8	2	A0	2	N.....Z.	
	b			ZP	OP8	3	A4	2		
	c			ZP,X	OP8,X	4	B4	2		
	d			ABS	OP16	4	AC	3		
	e			ABS,X	OP16,X	4 *	BC	3		

Poř. číslo	Inst- rukce	Prováděcí schéma	Způsob adresace	Operand	Počet period	Oper. kod	Počet byte	Příznaky NV.BDIZC	Čas. sch.
33 a	LSR		ACC	A	2	4A	1	0.....ZC	
b			ZP	OP8	5	46	2		
c			ZP,X	OP8,X	6	56	2		
d			ABS	OP16	6	4E	3		
e			ABS,X	OP16,X	7	5E	3		
34	NOP	(PC)+1→(PC)	IMPL		2	EA	1	.....	
35 a	ORA	(A) or (M)→(A)	IMM	#OP8	2	09	2	N.....Z.	
b			ZP	OP8	3	05	2		
c			ZP,X	OP8,X	4	15	2		
d			ABS	OP16	4	0D	3		
e			ABS,X	OP16,X	4 *	1D	3		
f			ABS,Y	OP16,Y	4 *	19	3		
g			(IND,X)	(OP8,X)	6	01	2		
h			(IND),Y	(OP8),Y	5 *	11	2		
36	PHA	(A)→(TOP) (S)-1→(S)	IMPL		3	48	1	.....	
37	PHP	(P)→(TOP) (S)-1→(S)	IMPL		3	08	1	.....	
38	PLA	(S)+1→(S) (TOP)→(A)	IMPL		4	68	1	N.....Z.	

Poř. číslo	Inst- rukce	Prováděcí schéma	Způsob adresace	Operand	Počet period	Oper. kód	Počet byte	Příznaky NV.BDIZC	Čas. sch.
39	PLP	⟨S⟩+1→⟨S⟩ ⟨TOP⟩→⟨P⟩	IMPL		4	28	1	NV.BDIZC	
40 a	ROL		ACC	A	2	2A	1	N.....ZC	
b			ZP	OP8	5	26	2		
c			ZP,X	OP8,X	6	36	2		
d			ABS	OP16	6	2E	3		
e			ABS,X	OP16,X	7	3E	3		
41 a	ROR		ACC	A	2	6A	1	N.....ZC	
b			ZP	OP8	5	66	2		
c			ZP,X	OP8,X	6	76	2		
d			ABS	OP16	6	6E	3		
e			ABS,X	OP16,X	7	7E	3		
42	RTI	⟨S⟩+1→⟨S⟩ ⟨TOP⟩→⟨P⟩ ⟨S⟩+1→⟨S⟩ ⟨TOP⟩→⟨PCL⟩ ⟨S⟩+1→⟨S⟩ ⟨TOP⟩→⟨PCH⟩	IMPL		6	40	1	NV.BDIZC	

Poř. číslo	Inst- rukce	Prováděcí schéma	Způsob adresace	Operand	Počet period	Oper. kód	Počet byte	Příznaky NV.BDIZC	Čas. sch.
43	RTS	⟨S⟩+1→⟨S⟩ ⟨TOP⟩→⟨PCL⟩ ⟨S⟩+1→⟨S⟩ ⟨TOP⟩→⟨PCH⟩ ⟨PC⟩+1→⟨PC⟩	IMPL		6	60	1	.....	
44 a	SBC	⟨A⟩-⟨M⟩-⟨C⟩→⟨A⟩	IMM	#OP8	2	E9	2	NV....ZC	
b			ZP	OP8	3	E5	2		
c			ZP,X	OP8,X	4	F5	2		
d			ABS	OP16	4	ED	3		
e			ABS,X	OP16,X	4	FD	3		
f			ABS,Y	OP16,Y	4	F9	3		
g			(IND,X)	(OP8,X)	6	E1	2		
h			(IND),Y	(OP8),Y	5	F1	2		
45	SEC	1→⟨C⟩	IMPL		2	38	1	1	
46	SED	1→⟨D⟩	IMPL		2	F8	1	....1...	
47	SEI	1→⟨I⟩	IMPL		2	78	1	.....1..	

Poř. číslo	Inst-rukce	Prováděcí schéma	Způsob adresace	Operand	Počet period	Oper. kód	Počet byte	Příznaky NV.BDIZC	Čas. sch.
48 a	STA	(A)→(M)	ZP	OP8	3	85	2	.....	
b			ZP,X	OP8,X	4	95	2		
c			ABS	OP16	4	8D	3		
d			ABS,X	OP16,X	5	9D	3		
e			ABS,Y	OP16,Y	5	99	3		
f			(IND,X)	(OP8,X)	6	81	2		
g			(IND),Y	(OP8),Y	6	91	2		
49 a	STX	(X)→(M)	ZP	OP8	3	86	2	.....	
b			ZP,X	OP8,X	4	96	2		
c			ABS	OP16	4	8E	3		
50 a	STY	(Y)→(M)	ZP	OP8	3	84	2	.....	
b			ZP,X	OP8,X	4	94	2		
c			ABS	OP16	4	8C	3		
51	TAX	(A)→(X)	IMPL		2	AA	1	N.....Z.	
52	TAY	(A)→(Y)	IMPL		2	A8	1	N.....Z.	
53	TSX	(S)→(X)	IMPL		2	BA	1	N.....Z.	
54	TXA	(X)→(A)	IMPL		2	8A	1	N.....Z.	
55	TXS	(X)→(S)	IMPL		2	9A	1	.....	
56	TYA	(Y)→(A)	IMPL		2	98	1	N.....Z.	

LEGENDA K TABULCE

Registry:

X	index-registr X
Y	index-registr Y
A	akumulátor
S	ukazatel vrcholu zásobníku /Stack/
PC	programový čítač /Program Counter/
PCH	významnější byte programového čítače
PCL	méně významný byte programového čítače
P	příznakový registr

Příznaky:

N	příznak znaménka - Negative
V	příznak přetečení mantisy (b_7 xor b_6) - Overflow
B	příznak programového přerušení - Break
D	příznak dekadického režimu - Decimal
I	příznak masky přerušení - Interrupt
Z	příznak nulovost - Zero
C	příznak přenosu - Carry

Symboly:

$\longrightarrow$	přesun
$\overline{C}$	inverze příznaku C
TOP	vrchol zásobníku
M	efektivní adresa
M_7	bit b_7 efektivní adresy
M_6	bit b_6 efektivní adresy
OP8	osmibitový /1 byte/ operand instrukce
OP16	šestnáctibitový /2 byte/ operand instrukce
( )	obsah
#	symbol označující bezprostřední adresování

Operátory:

+	součet
-	rozdíl

and	logický součin
or	logický součet
xor	exklusivní logický součet, nonekvivalence

Adresování:

IMPL	Implied - implicitní
ACC	Accumulator - přímé
IMM	Immediate - bezprostřední
ABS	Absolute - absolutní
ABS,X	Absolute X-indexed - absolutní přes index-registr X
ABX,Y	Absolute Y-indexed - absolutní přes index-registr Y
ZP	Zero Page - nulostránkové absolutní
ZP,X	Zero Page X-indexed - nulostránkové přes index-registr X
ZP,Y	Zero Page Y-indexed - nulostránkové přes index-registr Y
REL	Relative - relativní
IND	Indirect - nepřímé
(IND,X)	X-indexed Indirect - nepřímé přes index-registr X
(IND),Y	Indirect Y-indexed - nepřímé přes index-registr Y

Poznámky:

* Přičítá se jeden takt, jestliže dojde k přetečení do následující stránky

** +0 taktů, jestliže se skok neuskuteční
+1 takt, jestliže se skok uskuteční na tutéž stránku
+2 takty, jestliže se skok uskuteční na předcházející nebo následující stránku paměti

Příloha 2

SEZNAM NÁZVŮ INSTRUKCÍ 6502

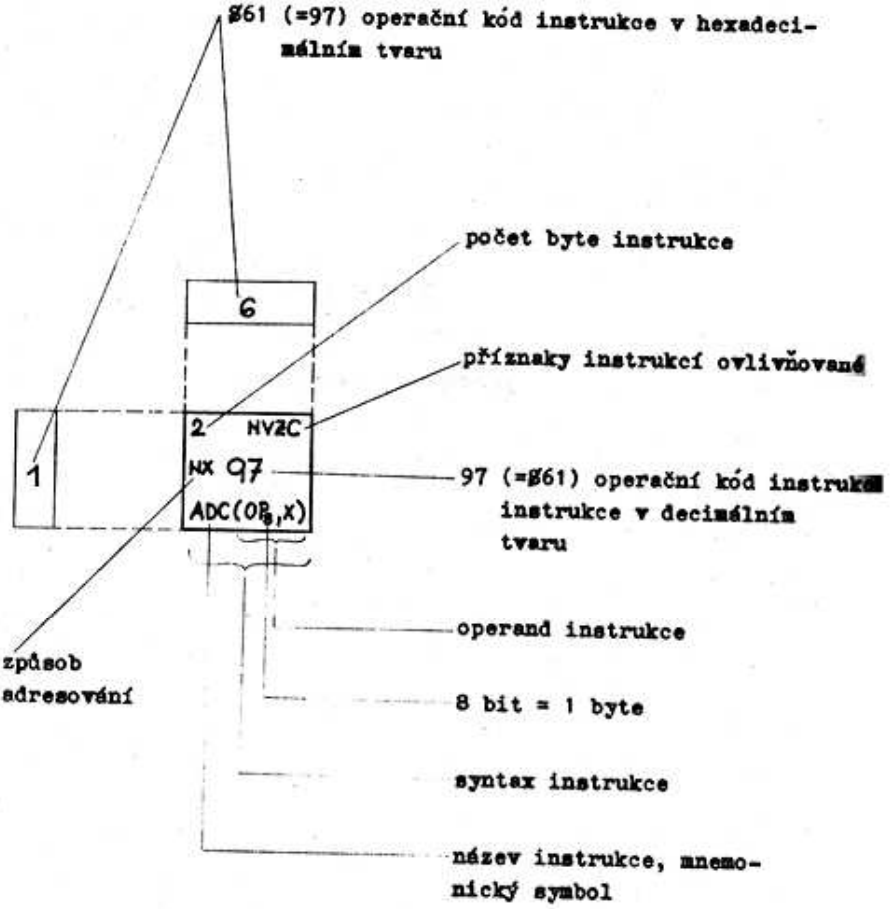
Přehled mnemonických názvů instrukcí JSA 6502

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift Left One Bit /Memory or Accumulator/
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memoty with Accumulator
NMI	Branch on Result Minus /Negative/
BNE	Branch on Result not Zero
BPL	Branch on Result Plus /Positive/
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode Flag
CLI	Clear Interrupt Disable Flag
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index Register X
CPY	Compare Memory and Index Register Y
DEC	Decrement Memory by One
DEX	Decrement Index Register X by One
DEY	Decrement Index Register Y by One
EOR	"Exclusive-Or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index Register X by One
INY	Increment Index Register Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index Register X with Memory
LDY	Load Index Register Y with Memory
LSR	Shift Right One Bit /Memory or Accumulator/
NOP	No Operation
ORA	"OR" Memory with Accumulator

PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left /Memory or Accumulator/
ROR	Totate One Bit Right /Memory or Accumulator/
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Substract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode Flag
SEI	Set Interrupt Disable Flag
STA	Store Accumulator in Memory
STX	Store Index Register X in Memory
STY	Store Index Register Y in Memory
TAX	Transfer Accumulator to Index Register X
TAY	Transfer Accumulator to Index Register Y
TSX	Transfer Stack Pointer to Index Register X
TXA	Transfer Index Register X to Accumulator
TXS	Transfer Index Register X to Stack Pointer
TYA	Transfer Index Register Y to Accumulator

Příloha 3

INSTRUKČNÍ SOUBOR 6502


$61 (=97) operační kód instrukce v hexadecimálním tvaru
počet byte instrukce
příznaky instrukcí ovlivňované
97 (=$61) operační kód instrukce v decimálním tvaru
operand instrukce
způsob adresování
8 bit = 1 byte
syntax instrukce
název instrukce, mnemonický symbol
6
1
2
NVZC
NX 97
ADC(OP,X)

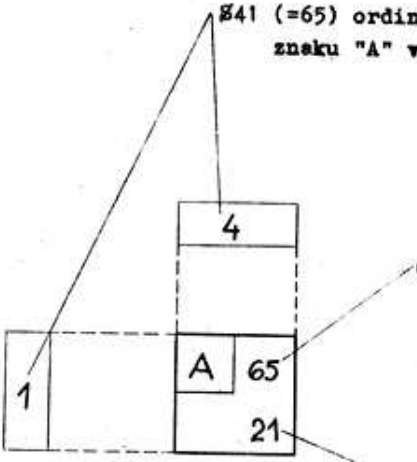
INSTRUKČNÍ SOUBOR 6502

H / L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	H / L
0	1 1→B P 0 BRK	2 – R 16 BPL OP$_8$	3 – A 32 JSR OP$_{16}$	2 – R 48 BMI OP$_8$	1 INV.BDIZC P 64 RTI	2 – R 80 BVC OP$_8$	1 – P 96 RTS	2 – R 112 BVS OP$_8$	128	2 – R 144 BCC OP$_8$	2 N,Z I 160 LDY #OP$_8$	2 – R 176 BCS OP$_8$	2 N,Z,C I 192 CPY #OP$_8$	2 – R 208 BNE OP$_8$	2 N,Z,C R 224 CPX #OP$_8$	2 – R 240 BEQ OP$_8$	0
1	2 N,Z NX 1 ORA (OP$_8$,X)	2 N,Z NY 17 ORA (OP$_8$),Y	2 N,Z NX 33 AND (OP$_8$,X)	2 N,Z NY 49 AND (OP$_8$),Y	2 N,Z NX 65 EOR (OP$_8$,X)	2 N,Z NY 81 EOR (OP$_8$),Y	2 NVZC NX 97 ADC (OP$_8$,X)	2 NVZC NY 113 ADC (OP$_8$),Y	2 NX 129 STA (OP$_8$,X)	2 NY 145 STA (OP$_8$),Y	2 N,Z NX 161 LDA (OP$_8$,X)	2 N,Z NY 177 LDA (OP$_8$),Y	2 N,Z,C NX 193 CMP (OP$_8$,X)	2 N,Z,C NY 209 CMP (OP$_8$),Y	2 N,V,Z,C NX 225 SBC (OP$_8$,X)	2 N,V,Z,C NY 241 SBC (OP$_8$),Y	1
2	2	18	34	50	66	82	98	114	130	146	2 N,Z I 162 LDX #OP$_8$	178	194	210	226	242	2
3	3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243	3
4	4	20	2 N↑→N,N6→Y Z 36 Z BIT OP$_8$	52	68	84	100	116	2 – Z 132 STY OP$_8$	2 – ZX 148 STY OP$_8$,Y	2 N,Z Z 164 LDY OP$_8$	2 N,Z ZX 180 LDY OP$_8$,X	2 N,Z,C Z 196 CPY OP$_8$	212	2 N,Z,C Z 228 CPX OP$_8$	244	4
5	2 N,Z Z 5 ORA OP$_8$	2 N,Z ZX 21 ORA OP$_8$,X	2 N,Z Z 37 AND OP$_8$	2 N,Z ZX 53 AND OP$_8$,X	2 N,Z Z 69 EOR OP$_8$	2 N,Z ZX 85 EOR OP$_8$,Y	2 N,V,Z,C Z 101 ADC OP$_8$	2 NVZC ZX 117 ADC OP$_8$,X	2 – Z 133 STA OP$_8$	2 – ZX 149 STA OP$_8$,X	2 N,Z Z 165 LDA OP$_8$	2 N,Z ZX 181 LDA OP$_8$,X	2 N,Z,C Z 197 CMP OP$_8$	2 N,Z,C ZX 213 CMP OP$_8$	2 N,V,Z,C Z 229 SBC OP$_8$	2 N,V,Z,C ZX 245 SBC OP$_8$,Y	5
6	2 N,Z,C Z 6 ASL OP$_8$	2 N,Z,C ZX 22 ASL OP$_8$,X	2 N,Z,C Z 38 ROL OP$_8$	2 N,Z,C ZX 54 ROL OP$_8$,X	2 8→N,8→C Z 70 Z LSR OP$_8$	2 8→N,8→C ZX 86 Z LSR OP$_8$,X	2 N,Z,C Z 102 ROR OP$_8$	2 N,Z,C ZX 118 ROR OP$_8$,X	2 – Z 134 STX OP$_8$	2 – ZX 150 STX OP$_8$,Y	2 N,Z Z 166 LDX OP$_8$	2 N,Z ZY 182 LDX OP$_8$,Y	2 N,Z Z 198 DEC OP$_8$	2 N,Z ZX 214 DEC OP$_8$,X	2 N,Z Z 230 INC OP$_8$	2 N,Z ZX 246 INC OP$_8$,Y	6
7	7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247	7
8	1 – P 8 PHP	1 8→C P 24 CLC	1 INV.BDIZC P 40 PLP	1 1→C P 56 SEC	1 – P 72 PHA	1 8→I P 88 CLI	1 N,Z P 104 PLA	1 1→I P 120 SEI	1 N,Z P 136 DEY	1 N,Z P 152 TYA	1 N,Z P 168 TAY	1 8→V P 184 CLV	1 N,Z P 200 INY	1 8→D P 216 CLD	1 N,Z P 232 INX	1 T→D P 248 SED	8
9	2 N,Z I 9 ORA #OP$_8$	3 N,Z AY 25 ORA OP$_{16}$,Y	2 N,Z I 41 AND #OP$_8$	3 N,Z AY 57 AND OP$_{16}$,Y	2 – I 73 EOR #OP$_8$	3 N,Z AY 89 EOR OP$_{16}$,Y	2 N,V,Z,C I 105 ADC #OP$_8$	3 NVZC AY 121 ADC OP$_{16}$,Y	137	3 – AY 153 STA OP$_{16}$,Y	2 N,Z I 169 LDA #OP$_8$	3 N,Z AY 185 LDA OP$_{16}$,Y	2 N,Z,C I 201 CMP #OP$_8$	3 N,Z,C AY 217 CMP OP$_{16}$,Y	2 N,V,Z,C I 233 SBC #OP$_8$	3 N,V,Z,C AY 249 SBC OP$_{16}$,Y	9
A	1 N,Z,C P 10 ASL A	26	1 N,Z,C P 42 ROL A	58	1 8→N,8→C P 74 Z LSR A	90	1 N,Z,C P 106 ROR A	122	1 N,Z P 138 TXA	1 – P 154 TXS	1 N,Z P 170 TAX	1 N,Z P 186 TSX	1 N,Z P 202 DEX	218	1 – P 234 NOP	250	A
B	11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251	B
C	12	28	3 M7→N,M6→V A 44 Z BIT OP$_{16}$	60	3 – A 76 JMP OP$_{16}$	92	3 – N 108 JMP (OP$_{16}$)	124	3 – A 140 STY OP$_{16}$	156	3 N,Z A 172 LDY OP$_{16}$	3 N,Z AX 188 LDY OP$_{16}$,Y	3 N,Z,C A 204 CPY OP$_{16}$	220	3 N,Z,C A 236 CPX OP$_{16}$	252	C
D	3 N,Z A 13 ORA OP$_{16}$	3 N,Z AX 29 ORA OP$_{16}$,Y	3 N,Z A 45 AND OP$_{16}$	3 N,Z AX 61 AND OP$_{16}$,Y	3 N,Z A 77 EOR OP$_{16}$	3 N,Z AX 93 EOR OP$_{16}$,X	3 NVZC A 109 ADC OP$_{16}$	3 NVZC AX 125 ADC OP$_{16}$,X	3 – A 141 STA OP$_{16}$	3 – AX 157 STA OP$_{16}$,X	3 N,Z A 173 LDA OP$_{16}$	3 N,Z AX 189 LDA OP$_{16}$,Y	3 N,Z,C A 205 CMP OP$_{16}$	3 N,Z,C AX 221 CMP OP$_{16}$,X	3 N,V,Z,C A 237 SBC OP$_{16}$	3 N,V,Z,C AX 253 SBC OP$_{16}$,X	D
E	3 N,Z,C A 14 ASL OP$_{16}$	3 N,Z,C AX 30 ASL OP$_{16}$,Y	3 N,Z,C A 46 ROL OP$_{16}$	3 N,Z,C AX 62 ROL OP$_{16}$,X	3 8→N,8→C A 78 Z LSR OP$_{16}$	3 8→N,8→C AX 94 Z LSR OP$_{16}$,X	3 N,Z,C A 110 ROR OP$_{16}$	3 N,Z,C AX 126 ROR OP$_{16}$,X	3 – A 142 STX OP$_{16}$	158	3 N,Z A 174 LDX OP$_{16}$	3 N,Z AX 190 LDX OP$_{16}$,Y	3 N,Z A 206 DEC OP$_{16}$	3 N,Z AX 222 DEC OP$_{16}$,X	3 N,Z A 238 INC OP$_{16}$	3 N,Z AX 254 INC OP$_{16}$,X	E
F	15	31	47	63	79	95	111	127	143	159	175	191	207	224	239	255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Příloha 4

ATASCII

Návod na použití tabulky ATASCII



ATASCII

L \ H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	H / L
0	♥ 0 á 40	♣ 16 Ü 50	32 00	3 48 10	e 64 20	P 80 30	♦ 96 ı 60	p 112 70	♥ 128 C0	♤ 144 D0	160 80	0 176 90	e 192 A0	P 208 B0	♦ 224 E0	p 240 F0	0
1	┣ 1 ù 41	┏ 17 â 51	! 33 01	1 49 11	A 65 21	Q 81 31	a 97 61	q 113 71	┣ 129 C1	┏ 145 D1	! 161 81	1 177 91	A 193 A1	Q 209 B1	a 225 E1	q 241 F1	1
2	┃ 2 N 42	━ 18 û 52	" 34 02	2 50 12	B 66 22	R 82 32	b 98 62	r 114 72	┃ 130 C2	━ 146 D2	" 162 82	2 178 92	B 194 A2	R 210 B2	b 226 E2	r 242 F2	2
3	┛ 3 É 43	╋ 19 î 53	# 35 03	3 51 13	C 67 23	S 83 33	c 99 63	s 115 73	┛ 131 C3	╋ 147 D3	# 163 83	3 179 93	C 195 A3	S 211 B3	c 227 E3	s 243 F3	3
4	┫ 4 ç 44	● 20 é 54	$ 36 04	4 52 14	D 68 24	T 84 34	d 100 64	t 116 74	┫ 132 C4	● 148 D4	$ 164 84	4 180 94	D 196 A4	T 212 B4	d 228 E4	t 244 F4	4
5	┓ 5 ô 45	▬ 21 è 55	% 37 05	5 53 15	E 69 25	U 85 35	e 101 65	u 117 75	┓ 133 C5	▬ 149 D5	⁄ 165 85	5 181 95	E 197 A5	U 213 B5	e 229 E5	u 245 F5	5
6	╱ 6 ò 46	▚ 22 ñ 56	& 38 06	6 54 16	F 70 26	V 86 36	f 102 66	v 118 76	╱ 134 C6	▌ 150 D6	& 166 86	6 182 96	F 198 A6	V 214 B6	f 230 E6	v 246 F6	6
7	╲ 7 ì 47	┳ 23 ê 57	' 39 07	7 55 17	G 71 27	W 87 37	g 103 67	w 119 77	╲ 135 C7	╈ 151 D7	' 167 87	7 183 97	G 199 A7	W 215 B7	g 231 E7	w 247 F7	7
8	◢ 8 £ 48	┻ 24 ä 58	(40 08	8 56 18	H 72 28	X 88 38	h 104 68	x 120 78	◢ 136 C8	┻ 152 D8	(168 88	8 184 98	H 200 A8	X 216 B8	h 232 E8	x 248 F8	8
9	▄ 9 ï 49	▒ 25 à 59	) 41 09	9 57 19	I 73 29	Y 89 39	i 105 69	y 121 79	▄ 137 C9	▊ 153 D9	) 169 89	9 185 99	I 201 A9	Y 217 B9	i 233 E9	y 249 F9	9
A	◣ 10 ü 4A	┗ 26 Ä 5A	* 42 0A	: 58 1A	J 74 2A	Z 90 3A	j 106 6A	z 122 7A	◣ 138 CA	┗ 154 DA	* 170 8A	: 186 9A	J 202 AA	Z 218 BA	j 234 EA	z 250 FA	A
B	■ 11 ä 4B	╞ 27 ß 5B	+ 43 0B	; 59 1B	K 75 2B	[91 3B	k 107 6B	♠ 123 7B	■ 139 CB	EOL 155 DB	+ 171 8B	; 187 9B	K 203 AB	[219 BB	k 235 EB	♠ 251 FB	B
C	■ 12 ö 4C	↑ 28 5C	, 44 0C	< 60 1C	L 76 2C	\ 92 3C	l 108 6C		124 7C	▀ 140 CC	↑ 156 DC	, 172 8C	< 188 9C	L 204 AC	\ 220 BC	┣ 236 EC	┃ 252 FC	C
D	▔ 13 ü 4D	↓ 29 5D	- 45 0D	= 61 1D	M 77 2D	] 93 3D	m 109 6D	↖ 125 7D	▔ 141 CD	↓ 157 DD	- 173 8D	= 189 9D	M 205 AD	] 221 BD	m 237 ED	↖ 253 FD	D
E	▁ 14 ö 4E	← 30 5E	. 46 0E	> 62 1E	N 78 2E	^ 94 3E	n 110 6E	◀ 126 7E	▁ 142 CE	← 158 DE	▒ 174 8E	> 190 9E	N 206 AE	^ 222 BE	n 238 EE	◀ 254 FE	E
F	■ 15 ö 4F	→ 31 5F	/ 47 0F	? 63 1F	O 79 2F	_ 95 3F	o 111 6F	▶ 127 7F	■ 143 CF	→ 159 DF	/ 175 8F	? 191 9F	O 207 AF	_ 223 BF	o 239 EF	▶ 255 FF	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Igi/2019