

FORTH

Ing. Rudolf Pecinovský, CSc.

V posledních letech začaly do mnoha odvětví národního hospodářství pronikat mikropočítače. Ale nejen do národního hospodářství. Mikropočítače již v současné době vlastní i v naší republice tisíce amatérů a počítačových nadšenců..

Prakticky jediným vyšším programovacím jazykem, který je mezi těmito počítači hojněji rozšířen, je jazyk BASIC. Tento informoval článek „Má FORTH naději?“ v AR 12/83.

V poslední době se začíná stále více prosazovat jazyk FORTH, který má pro uživatele mikropočítačů řadu předností.

V tomto kurzu bychom vás chtěli s jazykem FORTH nejen podrobně seznámit, ale částečně vás i přesvědčit, že možná právě tento jazyk vám pomůže realizovat všechny vaše nápadů.

1. ZÁSOBNÍK

Dříve, než si začneme vyprávět cokoliv o programování, musíme si vysvětlit pojmem zásobníku a s ním spojených operací. Tento pojem bude v celém dalším výkladu klíčový a bez jeho pochopení nemá smysl výklad vůbec začítat.

Zásobník je datová struktura, jejíž činnost lze modelovat např. pomocí šuplíku v psacím stole. Do šuplíku ukládáme papíry s poznámkami, a to jeden na druhý. Přístup máme vždy pouze k papíru naposledy uloženému. Potřebujeme-li papír dříve uložený, musíme napřed odebrat všechny papíry, které leží nad ním.

Takto pracující datová struktura se nazývá zásobník (stack). V literatuře se může také často setkat s názvem LIFO, což je zkratka z „Last In First Out“ neboli poslední tam, první ven.

Položky na zásobníku budeme v dalším textu označovat následujícimi zkratkami:

TOS — vršek zásobníku (Top Of Stack) — položka, kterou jsme na zásobníku dali naposledy, nebo kterou se tam právě chystáme dát.

NOS — následující za TOS (Next On Stack) — mohli bychom říci polohu pod TOS.

NNOS — následující za NOS.

Zásobník budeme v dalším textu zobrazovat dvěma způsoby, a to buď v řádce zleva doprava, takže TOS bude nejvíce vpravo

.....NNOS NOS TOS

a nebo pod sebou, a to tak, že TOS bude NEJSPODNĚJŠÍ položka. Uvědomuj si, že tento způsob zápisu si protiřečí s terminy „uložit na vršek“ nebo „první pod vrškem“, ale z řady dalších důvodů je tento způsob zápisu výhodnější, nehledě na to, že ve všech mikropočítačích roste zásobník směrem k nižším adresám (tedy dolů), takže je toto značení v literatuře obvyklé, pěstože si, jak jsem již pojmenoval, s terminologií protiřečí.

Pro názornou představu si ukážeme, jak bude vypadat zásobník při tomto způsobu zápisu. Budeme-li na něj postupně ukládat číslo od jedné do pěti. Zároveň bude u každé položky uvedena patřičná ze tří výše uvedených zkrátek:

1 TOS	1 NOS	1 NNOS	1	1
2 TOS	2 NOS	2 NNOS	2	
3 TOS	3 NOS	3 NNOS	3	
4 TOS	4 NOS	4 NNOS	4	
			5 TOS	

Chceme-li nyní položky odebrat, budeme tak činit v obráceném pořadí, než jsme je ukládali, tedy:

1	1	1 NNOS	1 NOS	1 TOS
2	2	2 NNOS	2 NOS	2 TOS
3	3 NNOS	3 NOS	3 TOS	
4	4 NOS	4 NNOS		
		5 TOS		

2. FILOSOFIE JAZYKA FORTH

První, co začátečníka na jazyku FORTH většinou zarází, je jeho naprostá odlišnost od všeho, s čím se doposud setkal (nebyl-li to zrovna jazyk LISP, kterému se alespoň vzdáleně podobá). Práce v jazyku FORTH je celá postavena na dvou zásobnících. Prvý je systémový a jsou do něj ukládány návratové adresy a některé další údaje a uživateli se nedoporučuje jej příliš využívat. Jelikož je tento zásobník často označován jako **zásoobník návratových adres**, budeme jej v textu značit **ZNA**.

Uživateli je určen druhý zásobník, který je plně pod jeho kontrolou a na kterém se provádí veškeré operace. Tento zásobník budeme označovat **UZ (uživatelský)**.

Úplný program jazyku FORTH je tvoren posloupnosti tzv. **slov**, oddělených mezerymi nebo znaky „Nová řádka“. Slovo může být označeno (pojmenováno) jakýmkoliv znakem kódů ASCII s výjimkou mezery a řídicích znaků nebo libovolnou posloupností těchto znaků neobsahující mezeru ani řídicí znaky. Každému slovu je jednoznačně přiřazena nějaká činnost a slovo tedy představuje obdobu procedury. Tato procedura očekává všechny své parametry na zásobníku, odkud je po převzetí vymaze a uloží tam případně výsledky.

Definice všech slov, neboli popisy ke slovům přiřazených činnosti, jsou uloženy v tzv. slovníku, který bývá někdy označován za třetí zásobník. Slouží k uchovávání definic nových slov, hodnot proměnných, polí, atd. Slova k němu můžeme přidávat a opět je z něj odmazávat obdobně, jako položky na zásobníku.

Interpret čte program slovo za slovem, každé slovo napřed najde ve slovníku a vzápětí vykoná patřičnou činnost. Pokud slovo nenajde ve slovníku, pokusí se jej interpretovat jako číslo, jehož hodnotu pak uloží na vrchol zásobníku. Pokud slovo nelze interpretovat ani jako číslo, ohláší chybou.

Před započetím programování v jazyku FORTH je zapotřebí provést důkladnou analýzu úlohy. Řešení problémů si rozložime na hlavní části, z nichž každou pak znova dále dělíme. Tento proces trvá tak dlouho, dokud jednotlivé části nejsou natolik jednoduché, že je můžeme naprogramovat pomocí známých slov. Takto provedená analýza se nazývá „analýzou shora dolů“ (top-down design).

Po provedené analýze pak naefinujeme slova, realizujici nejjednodušší činnosti.

Pomocí těchto slov a slov dříve známých naefinujeme realizaci činnosti složitějších atd., až na konec naefinujeme jedno slovo, řešící celý problém. Tento postup se nazývá „programování zdola nahoru“ (bottom-up programming).

Specifickou vlastností jazyka FORTH je, že tato postupně tvorená slova nejen definiujeme, ale také zároveň odladujeme. Okamžité odladění slov je umožněno tím, že všechna slova očekávají své vstupní parametry na zásobníku a tamtéž ukládají své výsledky. Není tedy nutné vytvářet zvláštní procedury, předávající testovaným slovům parametry a tisknoucí jejich výsledky, neboť tyto činnosti můžeme jednoduše reálnovat „ručně“.

Mnohé jiné jazyky, např. BASIC, nás nenutí provádět bezpodmínečně tuto teoretickou přípravu a svoji strukturu umožňují přímé programování, kdy provádíme zároveň analýzu a kódování. Pokud však po dokončení programu porovnáme časy potřebné k naprogramování úlohy oběma metodami, zjistíme, že první metoda je většinou neporovnatelně rychlejší a programy jí získané jsou i mnohem přehlednější. Nepovažujte proto nutnost teoretické přípravy za nevýhodu jazyka FORTH, ale za jeho výhodu, která nás ochrání před nekontrolovaným odladováním včetně chybných programů.

3. METODICKÉ POZNÁMKY

Základní interpret jazyka FORTH obsahuje podle implementace okolo 100 až 200 slov. Tato skutečnost vyžaduje poněkud odlišné pojetí výuky, než bývá u ostatních programovacích jazyků zvykem. Styl výuky by se měl spíše blížit výuce jazyků přirozených.

Kurz tedy nebude seznamem pravidel použití jednotlivých slov. Každá kapitola se bude zabývat některou z vlastností jazyka. Na jejím začátku budou uvedeny definice nových slov, použitých v této lekci spolu s vysvětlením jejich významu a funkce. Tato slova pak budou v dalším textu pokládána za známá.

V definicích bude u každého slova vpravo od jeho názvu uveden v závorkách očekávaný (vlevo od šipky) a výsledný (vpravo od šipky) stav uživatelského zásobníku.

V popisech stavu zásobníku budeme používat následující označení:

- A — Adresa.
- (A) — Obsah buňky na adrese A (16 bitů).

(1)

FORTH

- B(A)** — Obsah bajtu na adrese A.
B — Bajt je číslo od 0 do 255. Pokud chceme uložit bajt na TOS, uloží se šestnáctibitová položka, jež má v horních osmi bitech nuly. Chceme-li naopak uložit obsah TOS do bajtu, uloží se spodních 8 bitů TOS.
- D** — Celé číslo dvojnásobné přesnosti. Zaujmá dvě položky na zásobníku (tj. 4 bajty). Jeho rozsah je od -2 147 483 648 do 2 147 483 647.
- F** — Pravidlostní hodnota (flag). Nula je chápána jako „**FALSE**“ (nepravda), nenula hodnota jako „**TRUE**“ (pravda). Máme-li někam uložit pravidlostní hodnotu, je „**TRUE**“ ukládáno jako +1 nebo -1 a „**FALSE**“ jako 0.
- N** — Celé číslo v rozsahu od -32 768 do 32 767 (16 bitů).
- RA** — Návratová adresa (return address) z právě prováděného slova. Při vstupu do slova je (TOS) ZNA = RA (viz 8. lekce).
- U** — Celé číslo v rozsahu od 0 do 65535 (unsigned).
- X** — Obecná šestnáctibitová položka.
- xxx** — Jméno slova jazyka FORTH.
- .xxx.** — Adresa slova xxx.
- Z** — ASCII kód znaku — platí pro něj stejně zásady jako pro B.

V dalším textu budeme často hovořit o buňkách paměti nebo správněji o paměťových místech. Paměťovým místem (**PM**) budeme rozumět část paměti, kde je uložena nějaká informace. Tato část může mít různou velikost. Podle potřeby to jednou bude bit, jindy bajt, dvoubajt (slovo) atd. U vícebajtových PM budeme adresou paměťového místa myslit adresu jeho prvního bajtu.

Při práci s PM budeme potřebovat občas rozlišovat adresu místa a hodnotu na této adrese uloženou. Položky budeme značit identifikátorem (např. TOS, BASE). Pokud bude třeba zdůraznit, že se jedná o hodnotu, bude tento identifikátor uzavřen do kulatých závorek (např. (TOS), (BASE)); bude-li třeba zdůraznit, že se jedná o adresu, bude uzavřen mezi dvě tečky (např. .BASE je adresa proměnné BASE, .(BASE) je adresa, na níž je uložena hodnota proměnné BASE).

V některých programech (definicích) uvádíme pro snazší porozumění a větší názornost stav zásobníku po vykonání každého slova pod tímto slovem, a odděluji jej plnou čarou. U dvojteckových definic (budou vysvětleny) uvádíme stav zásobníku, který je očekáván před jejich plněním, pod jménem nově definovaného slova.

Pokud bude využíván i ZNA, budou zobrazeni obou zásobníků oddělena svislou čarou, přičemž UZ bude znázorněn vlevo a ZNA vpravo od této čáry. Zápis

3	4
7	

bude tedy znamenat, že na UZ je (TOS) = 7 a (NOS) = 3 a na ZNA je (TOS) = 4.

Pro lepší porozumění příkladům i pro odlaďování případných vlastních programů vám doporučujeme zhotovení tzv. RUP, což je zkratka názvu „ruční univerzální počítač“. Pro tento účel si opatřete sadu dominových kamenů nebo jiných, jim podobných kostek. V nouz postačí i z kartonu nastříhané obdélníky. Zásobník pak bude představovat řadu pod sebou vyrovnaných kamenů (ko-

stek, obdélníků), na nichž budou napsány hodnoty patřících položek. Uložení položky na vrchol zásobníku budeme realizovat tak, že její hodnotu nebo symbolické označení napišeme na volný kámen (kostku, karton) a tento přidáme na konec řady. Odebrání položky ze zásobníku bude prostým odebráním naposledy uloženého kamene. Všechna čísla, která budeme psát při zobrazování zásobníku, budeme zapisovat v desítkové soustavě. Budeme-li chtít číslo napsat v jiné soustavě, uvedeme první písmeno názvu soustavy v závorce za číslem. Tedy:

$$18 = 1010(B) = 22(O) = 18(D) = 12(H),$$

kde **B** značí binární (dvojkovou), **O** oktalovou (osmičkovou), **D** dekadickou (desítkovou) a **H** hexadecimální (šestnáctkovou) číselnou soustavu.

4. ARITMETICKÉ OPERACE

Nová slova:

- + — (N1 N2 → (N1 + N2))
Seče (TOS) a (NOS) a výsledek uloží na TOS.
- — (N1 N2 → (N1-N2))
Odeče (TOS) od (NOS), výsledek uloží na TOS.
- * — (N1 N2 → (N1*N2))
Vynásobi (TOS) a (NOS), výsledek uloží na TOS.
- / — (N1 N2 → (N1/N2))
Vydělí (NOS)/(TOS) a celou část podílu uloží na TOS (tedy 5/2 = 2)
- DUP** — (X → X X)
Zduplikuje TOS.
- (N →)
Vytiskne hodnotu TOS na obrazovku v dané bázi.

S ideou zásobníku je spojena tzv. postfixová notace, nazývaná též často obrácená polská notace (zkratka RPN tzn. Reverse Polish Notation). Tento způsob zápisu požaduje, abychom vždy uváděli napřed parametry (operandy) a teprve potom jméno procedury (operátor), která tyto parametry zpracuje. Pokud tedy potřebujeme spočítat „13×17“, musíme napsat „13 17 ×“. Obdobně zapisujeme i složitější výrazy. Např. výraz

$$((2+3)*(7-4))^2$$

bychom naprogramovali v jazyku FORTH následovně:

2	3	+	7	4	-	*	DUP	*	
2	2	5	5	5	5	15	15	225	
		3	7	7	3		15		
						4			

Zastavme se u tohoto příkladu podrobněji. Prvním slovem je slovo „2“. V základní verzi FORTH každé slovo, které je číslem, uloží na TOS svou hodnotu. Slovo „2“ uloží tedy na TOS číslo 2, jak je znázorněno pod příkladem.

Dalším slovem je slovo „3“ Při jeho vykonání se uloží na TOS číslo 3 a číslo 2 bude v NOS.

Slovo „+“ vezme (TOS) (=3) a (NOS) (=2), seče je a výsledek uloží na TOS. Po jeho vykonání je na TOS číslo 5. Toto číslo je také v danou chvíli jedinou položkou na UZ.

Slova „7“ a „4“ přidají své hodnoty postupně na TOS, takže po jejich vykonání budou na UZ tři položky.

Slovo „*“ — vezme (TOS) (=4) a (NOS) (=7), odeče (TOS) od (NOS) a výsledek (=3) uloží na TOS.

Následuje slovo „*“ které vezme (TOS)

(=3) a (NOS) (=5), vynásobi je a výsledek (=15) uloží na TOS.

Nyní již zbývá pouze tento součin umocnit. Jeníkž mocnění na druhou není mezi základními slovy jazyka, musíme je obejít. Jedná se o možnost, že zduplikovat TOS pomocí slova DUP a takté vzniklé dva operandy spolu vynásobit.

Posledním slovem v našem příkladu je slovo „. (tečka), které vezme (TOS) a vytiskne na obrazovku tuto hodnotu, následovanou jednou mezerou, která oddeláuje za sebou jdoucí čísla; pak odstraní svůj operand ze zásobníku.

Na příkladu aritmetických operací je názorně vidět, jak pracují všechny procedury jazyka FORTH — slova. Berou si své argumenty ze zásobníku, kam vzápětí uloží výsledek. Chtěl bych zde ještě jednou zdůraznit, že slovo „berou“ znamená odeberou, neboli, že se pak již tyto argumenty na zásobníku nevyskytují (viz stav zásobníku po vykonání slova).

Z uvedeného příkladu je také vidět, proč FORTH nepatří k nejvýhodnějším jazykům pro řešení problémů, v nichž zcela převažují numerické výpočty. Nejen, že výrazy jsou poněkud nepřehledné, což by ostatně po odhadění programu nemuselo být na závadu, ale jeho hlavním handicapem je neutrálné přesouvání položek do a ze zásobníku, což znatelně zpomaluje výpočet. Přesto však FORTH bývá i v numerických výpočtech 10 až 20krát rychlejší než BASIC a 3 až 5krát rychlejší než celočíselný BASIC. Dobrý kompilátor jazyka FORTRAN však dokáže tyto výpočty ještě rychleji. Již ne tak přesvědčivě jsou výsledky malých kompilátorů navržených pro mikropočítače a pokud opustíme pole numerické matematiky, najdeme nepřeberný počet aplikací, kde s jazykem FORTH mohou v rychlosti a efektivnosti využít paměti, soutěžit pouze programy, psané v assembleru nebo ve strojovém kódu. I zde však může nastat paradoxní situace, že program psaný v jazyku FORTH, tedy ve vyšším programovacím jazyku, zabere v paměti méně místa, než program psaný ve strojovém kódu.

FORTH ve své základní verzi počítá pouze s celými čísly v rozsahu od -32 768 do 32 767. Ve většině aplikací, v nichž se FORTH používá (řízení, hry), tato přesnost zcela vyhovuje a mimo to, výpočty v pevné čárci jsou mnohem rychlejší. V případě potřeby však lze nadefinovat i slova, která pracují s jakýmkoliv jinými typy dat (vice-násobná přesnost, plovoucí čárka, komplexní čísla, ...), která jsou v dané chvíli potřeba. O všech těchto možnostech se postupně dozvete v průběhu kurzu.

Aby se vám trochu zažila práce se zásobníkem, zkuste si naprogramovat výpočet a vytisknutí výsledku následujících příkladů. Průběh akcí na zásobníku si znázorněte pomocí „RUP“ nebo tak, jak to bylo uvedeno v příkladu v textu. Pro kontrolu je ještě jeden příklad takto vyřešen. U ostatních jsou jen na konci lekce kontrolní řešení.

Zadání: $(13 - 8)^3 / (2 + 4)$

Řešení:

$$13 \quad 8 \quad - \quad DUP \quad DUP \quad * \quad * \quad 2 \quad 4 \quad - \quad / \quad .$$

13	13	5	5	5	125	125	125	20
	8	5	5	25	2	2	6	
				5			4	

Další příklady:

1. $(3 - 2)^*(2^3 - 6)/(9 - 3)^**2$
 2. $(3 + 6)^*(3 - 6^*7)$
 3. $2 + 3^*(6 - 4^*7)^**2$
 4. $(3 - 1)^**4$
- Kontrolní řešení:
1. $3 \cdot 2 - 2 \cdot 3 \cdot 6 + 9 \cdot 3 - DUP \cdot / \cdot .$
 2. $3 \cdot 6 + 3 \cdot 6 \cdot 7 \cdot - \cdot .$
 3. $2 \cdot 3 \cdot 6 \cdot 4 \cdot 7 \cdot - \cdot DUP \cdot ** \cdot + \cdot .$
 4. $3 \cdot 1 - DUP \cdot DUP \cdot .$

5. DEFINICE NOVÝCH SLOV — I

Nová slova:

: **xxx** — (→)

Začátek dvoječkové definice slova **xxx**. Nastavení režimu **COMPILE**.

: — (→)

Konec dvoječkové definice. Nastavení režimu **EXECUTE**.

Slova v lekci nadefinovaná:

2NA	3NA	4NA	5NA
1+	1-	2+	2-
2*			

FORTH umožňuje, aby si každý uživatel nadefinoval svá vlastní slova. Slovem, které umožňuje definovat tato nová slova, je slovo „;“ (dvoječka). První slovo, které následuje za dvoječkou (popř. za jí následující mezerou), se chápá jako název nově definovaného slova. Po něm následuje seznam dříve definovaných slov, která se mají během jeho provádění postupně vykonat. Celá definice je pak ukončena slovem „;“ (středník).

Rekněme, že bychom chtěli nadefinovat slovo, které umocní svůj argument na druhou. Jedna z možností je

: **2NA** DUP * ;

N	N	N*2
N		

Zastavme se u této definice podrobněji a rozebereme si ji slovo za slovem.

Slovo „;“ (dvoječka). Ve slovníku jsme si přečetli, že označuje „začátek dvoječkové definice“ a mimo to že „nastaví režim **COMPILE**“. O co vlastně jde. Když jsme v minulé kapitole napsali v programu slovo **DUP**, tak se toto slovo ihned provedlo. Počítač byl v prováděcím režimu, nazývaném **EXECUTE**. Při definici našeho nového slova však nechceme slovo **DUP** ihned provést. Potřebujeme počítač pouze oznamit, že až bude provádět slovo **2NA**, tak že toto slovo provede tak, že nejprve provede slovo **DUP** a potom slovo „;“. Potřebujeme je tedy ne provést, ale přeložit do definice slova **2NA**. Proto slovo „;“ nastavuje komplikační režim (režim **COMPILE**), ve kterém určí posloupnost slov, jež se budou provádět, až bude počítač provést slovo právě definované.

Slova **DUP** a „;“ se pak zkompilují do nově definovaného slova **2NA**.

Slovo „;“ (středník) překompileje do slovníku ekvivalent známého **RETURN**, neboli činnost, kterou je třeba vykonat při návratu z procedury (slova), a ukončí komplikaci = nástavu zpět režimu **EXECUTE**.

K této definici bych chtěl ještě připomenout, že pod ní je zobrazena pouze ta část UZ, kterou slovo bezprostředně využívá. Slovo **2NA** očekává v TOS číslo, jehož druhou mocninu je třeba vypočítat. Je-li toto číslo jedinou položkou na UZ či ne, to pro dané slovo není důležité. Jeho činnost bude vždy stejná nezávisle na zbylém obsahu zásobníku.

Dále bych chtěl upozornit na to, že stavy zásobníku zobrazena pod definicí jsou stavy, ke kterým dojde teprve až se slovo bude vykonávat. Skutečný stav zásobníku se během definování nového slova nemění.

Jakmile jsme nějaké slovo jednou nadefinovali, můžeme je použít v dalších definicích. Na ukázku předvedeme definice slov, které umocní TOS na třetí, na čtvrtou a na pátou. V levém sloupci tabulky jsou definice, které se snaží o maximální úsporu paměti, v pravém sloupci definice, které naopak

FORTH

Ing. R. Pecinovský, CSc.

preferuji rychlosť výpočtu a proto se snaží omezit hloubku volání na minimum:

:	3NA	DUP	2NA	*	:	3NA	DUP	DUP	*	*	*
	N	N	N	N**3		N	N	N	N	N**3	
		N	N**2				N	N	N	N**2	
:	4NA	DUP	3NA	*	:	4NA	DUP	*	DUP	*	*
	N	N	N	N**4		N	N	N**2	N**2	N**4	
:	5NA	DUP	4NA	*	:	5NA	DUP	DUP	*	DUP	*
	N	N	N	N**5		N	N	N	N	N	N**5
		N	N**4				N	N	N**2	N**2	N**4

Abyste si definování nových slov trochu procvičili, zkuste si nadefinovat slova:

1+ 1- 2+ 2- 2*

která příčtou (odečtem) k TOS jedničku (dvojku), resp. vynásobi TOS dvěma.

Druhým vaším úkolem bude zobrazit si SKUTEČNÉ stavy UZ při vykonávání řádky

5 : 1+ 1 + ; 1+

Kontrolní řešení:

: 1+ 1 + ;	: 1- 1 - ;
: 2+ 2 + ;	: 2- 2 - ;
: 2* 2 * ;	

5 : 1+ 1 + ; 1+

5 5 5 5 5 6

6. DEFINICE NOVÝCH SLOV — II

Nová slova:

SWAP — (N1 N2 → N2 N1)

Zamění (TOS) a (NOS).

NOT — (F → F̄)

Neguje logickou hodnotu TOS.

AND — (F1 F2 → (F1 & F2)b)

Logický součin (TOS) a (NOS) bit po bitu.

OR — (F1 F2 → (F1|F2)b)

Logický součet (TOS) a (NOS) bit po bitu.

XOR — (F1 F2 → (F1|F2)b)

Exkluzivní OR (NOS) a (TOS) bit po bitu.

0 — (N → F(N< 0))

Uloží na TOS logickou hodnotu výrazu „(TOS)< 0“

FORGET **xxx** (→)

Vymaže ze slovníku nová slova počínaje slovem **xxx**.

Slova v lekci nadefinovaná:

(→ (N1 N2 → F(N1 < N2))

Uloží na TOS logickou hodnotu výrazu „(NOS) < (TOS)“

Další slova:

0 >	0 =	0 ()	0 > =	0 < =
>	=	()	> =	< =

Definování nových slov je nejběžnější činnost, kterou v jazyku FORTH provádíme. Měli bychom s ní být proto seznámeni důkladněji.

Každé nově nadefinované slovo je zařazeno do slovníku známých slov (na jeho konec). Přečte-li počítač nějaké slovo, pokouší se je napřed nalézt ve slovníku.

Slovík se zásadně prohledává ODZADU, tedy od naposledy nadefinovaných slov. Najde-li počítač hledané slovo a je-li v režimu **EXECUTE**, ihned toto slovo vykona. Je-li v režimu **COMPILE**, příkompiluje odkaz na toto slovo do slova právě definovaného.

Pokud počítač hledané slovo ve slovníku nenajde, pokusí se je interpretovat jako číslo. Pokud se mu to podaří, uloží toto číslo na TOS. (**EXECUTE**) popř. (**COMPILE**) zadí, aby se při vykonávání právě definovaného slova uložilo toto číslo v pravou chvíli na TOS.

Pokud se mu nepodaří dané slovo interpretovat ani jako číslo, ohláší počítač chybu.

Po nadefinování každého slova bychom měli ihned vyzkoušet, zda toto slovo pracuje správně. Pokud v něm zjistíme chybu, musíme je nadefinovat znova. Avšak ve slovníku zůstane a nové se pouze přidá na konec! Jelikož se slovník prohledává odzadu, bude dále pod názvem slova vždy mírně jeho nejnovější definice.

Nepříjemné však je, že stará definice zbytečně zabírá místo v paměti. Bylo by tedy výhodné, mít možnost slovo ze slovníku vymazat. K tomuto účelu se používá slovo **FORGET**. Avšak **Pozor!** Slovo **FORGET** vymaže ze slovníku nejen slovo označené, ale i všechna slova, nadefinována po něm. Napsali-li bychom např..

: **PRVNÍ** ; : **PRVNÍ** ; : **DRUHÝ** ; : **DRUHÝ** ; : **TŘETÍ** ; **FORGET DRUHÝ**,

zůstaly by ve slovníku obě definice slova **PRVNÍ** a starší definice slova **DRUHÝ**. Pokud bychom tato slova chtěli vymazat všechna, museli bychom napsat:

FORGET PRVNÍ FORGET PRVNÍ

Z uvedených vlastností slovníku jasné vyplývá, proč se o něm někdy hovoří jako o třetím zásobníku. Použijte se přečíst tuto lekci ještě jednou a operace se slovníkem interpretovat jako operace se zásobníkem. Zdůvodněte si vlastnosti slovníku pomocí vlastnosti zásobníku. Na konci této lekce si

(3)

opět uděláme malé cvičení v definování nových slov a používání zásobníku. Pokuste se sami nadefinovat slova uvedená v druhé části slovníčku z úvodu této lekce. Definice těchto slov, uvedené v kontrolních řešeních, nejsou jedině možné a ani optimální. Mají spíše sloužit jako vodítka těm, kdo na svoje vlastní řešení nemohou přijít.

Kontrolní řešení:

```
: 0= NOT ;
: 0> = 0< NOT ;
: 0<> 0= NOT ;
: 0<= DUP 0< SWAP 0= OR ;
: 0> 0<= NOT ;
: < - 0< ;
: = - 0= ;
: <= - 0<= ;
atd.
```

Pokud netrváme na „TRUE“ = 1, lze definovat i novat i

```
: 0<> ;
```

7. ZÁKLAD ČISELNÉ SOUSTAVY

Nová slova:

BASE — (→ .(BASE).) Proměnná, ve které je uložen základ soustavy, v níž jsou čtena (vyjadřována) vstupující (vystupující) čísla. Slovo BASE uloží na TOS adresu, na niž je uložena hodnota této proměnné.
DECIMAL — (→) Nastaví čtení a vyjadřování čísel v desítkové soustavě.
@ — (A → (A)) Uloží na TOS obsah na adrese A.
! — (X A →) Uloží na adresu A položku z NOS.
.” — (→) Na obrazovku vytiskne následující text až po znak ” (uvozovky).
CR — (→) Na obrazovce provede přechod na počátek další řádky.
(— Komentářová závorka. Následující text až po znak „ (uzavírací závorka) včetně počítač ignoruje (nejvýše však do konce řádky).

Slova v lekci nadefinovaná:

BIN HEX OCT PREV .B BASE?

Vraťme se zpět k číslům a operacím s nimi. V jazyku FORTH si, na rozdíl od většiny ostatních jazyků, můžeme zvolit základ číselné soustavy, v níž budou čtena vstupní data a tištěny výsledky. Základ této soustavy je uložen v proměnné **BASE** (slovo jazyka FORTH) a změnou obsahu této proměnné můžeme číselnou soustavu libovolně měnit.

Při každém čtení čísla ze vstupní jednotky (např. klávesnice), se testuje obsah této proměnné a vypočte se skutečná hodnota čísla, které je pak dále zpracováno již zásadně ve dvojkovém tvaru. Obdobně se hodnota proměnné **BASE** testuje před každým tiskem.

FORTH-79 STANDARD zná slovo **DECIMAL**, které zavádí čtení a zápis v desítkové

FORTH

ing. R. Pecinovský, CSc.

soustavě. Pokud se nám toto slovo nelibí, např. pro svou délku, není problém tentýž úkon pojmenovat jinak, např.:

```
: DEC DECIMAL ;
```

Od této chvíle je systému ihned, zda nápisem **DECIMAL** nebo **DEC**, činnost bude vždy stejná. Napíšeme-li nyní:

```
: DECIMAL 10 * ;
```

bude systém od této chvíle slovo **DECIMAL** provádět tak, že se TOS vynásobi deseti. Na činnosti slova **DEC** se však nic nezmění, neboť ve slovníku je u **DEC** odkaz na starou definici slova **DECIMAL**, a ta zůstala nedotčena. Nová definice slova **DECIMAL** se tak může uplatnit pouze u slov, která byla definována po ní. Vše opět plyne z charakteru slovníku jako zásobníku.

Prozatím umíme číst a psát pouze v soustavě desítkové. Nyní si ukážeme postup, jakým lze nadefinovat i jiné základy. Pro nastavení dvojkové soustavy si nadefinujeme slovo:

```
: BIN 2 BASE ! ;
```

```
2 2
.(BASE).
```

Jistě jste si všimli, že vykonání slova, jež je názvem proměnné, znamená, že se na TOS uloží adresa, na niž je uložena hodnota této proměnné. Pro ty hlbavější jenom dodám, že adresa slova **BASE**, označovaná **.BASE**, je něco jiného než adresa, na niž je uložena hodnota proměnné **BASE**. Podrobnejší se o tom dozvít ve 14. lekci.

Slovo „!“ (vykříčník) pak na adresu, kterou najde na TOS, uloží hodnotu NOS. Od této chvíle je tedy tato hodnota hodnotou proměnné, jejíž adresa byla na TOS, v našem případě proměnné **BASE**.

Zde bych chtěl ještě jednou připomenout dvě věci. Za prvé, že dvojková soustava se nestaví při definici slova **BIN**, ale až při jeho provedení. V tuto chvíli by tedy na našem hypotetickém počítači byla stále ještě nastavena soustava desítková.

Za druhé bych chtěl znova zdůraznit, že změna **BASE** platí pouze pro vstup a výstup a na vnitřní zobrazení čísla nemá nejmenší vliv.

Pro ilustraci obou těchto připomínek uvedeme ještě definice osmičkové a šestnáctkové soustavy:

```
: OCT 8 BASE ! ;
```

```
8 8
.(BASE).
```

BIN

```
: HEX 10000 BASE ! ;
```

```
16 16
.(BASE).
```

V tomto příkladu jsme mezi dvěma definicemi provedly slovo **BIN**. Proto jsme v druhé definici museli číslo 16 vyjádřit ve dvojkové soustavě.

Pomoci uvedených slov můžeme v jazyku FORTH jednoduše realizovat převody mezi různými číselnými soustavami. Nadefinuj-

me slovo **PREV**, které vyjádří TOS ve všech dosud nadefinovaných soustavách:

: PREV	(V TOS OČEKÁVÁ ČÍSLO, JEŽ SE MÁ VYTISKNOUT)
BASE @ SWAP	(ÚSCHOVA PŮvodního ZÁKLAdu)
DUP BIN . “ (B) = “	(TISK ČÍSLA V DVOJKOVÉ SOUSTAVĚ)
DUP OCT . “ (O) = “	(TISK ČÍSLA V OSMIČKOVÉ SOUSTAVĚ)
DUP DEC . “ (D) = “	(TISK ČÍSLA V DESÍTKOVÉ SOUSTAVĚ)
HEX . “ (H) “	(TISK ČÍSLA V ŠESTNÁCTKOVÉ SOUSTAVĚ)
BASE !	(OBNOVA PŮvodního ZÁKLAdu)

Zde bych chtěl ještě upozornit, že „(“ (otevřiací závorka) je slovo jazyka FORTH a jako takové musí být od dalšího textu odděleno alespoň jednou mezerou. Na rozdíl od něj je „)“ (uzavírací závorka) pouze příznakem konce komentáře a nemusí být od předchozího textu oddělena. Nezapomeňte však, že v komentáři se žádná jiná uzavírací závorka nesmí objevit, jinak bude počítač text, který po ní následuje, provádět jako posloupnost slov jazyka FORTH.

Vyzkoušejte si provádění slova **PREV**. Po vykonání posloupnosti:

DEC 23 PREV

by se na obrazovce mělo objevit

10111 (B) = 27 (0) = 23 (D) = 17 (H)

Jako další úkol si zkuste naprogramovat slovo **.B**, které vytiskne obsah NOS v soustavě, jejíž základ najde na TOS. Po vytisknutí tohoto čísla bude samozřejmě obnovena původní soustava.

Posledním úkolem této lekce bude malé zamýšlení. Zkuste si odvodit, co se vytiskne na obrazovce po vykonání řádku:

BASE

a nadefinujte slovo **BASE?**, které vytiskne **(BASE)** v dešifrovatelné podobě.

Kontrolní řešení:

: .B	(TISK ČÍSLA V ZADANÉ SOUSTAVĚ)
BASE @ SWAP	(ÚSCHOVA STARÉHO ZÁKLAdu)
BASE !	(NASTAVENÍ NOVÉHO ZÁKLAdu)
SWAP :	(TISK ČÍSLA V NOVÉ SOUSTAVĚ)
BASE !	(OBNOVENÍ STARÉHO ZÁKLAdu)

Po provedení uvedeného řádku se na obrazovce objeví číslo 10, protože základ jakékoli číselné soustavy se v jím definované soustavě zapíše jako 10. Možné řešení je

DEC : BASE? BASE @ CR
<“ BASE = “ 10 .B ;

Zde bych chtěl upozornit, že při používání různých základů číselných soustav se nemusíte omezovat pouze na soustavy běžně používané. Velmi oblíbenou je např. číselná soustava se základem 36 (= 10 číslic a 26 písmen). Čísel v této soustavě se používají zakódování různých textů. Např. programový systém fig-FORTH používá číslo ve dvojnásobné přesnosti k zakódování názvu použitého procesoru. K vyvolání tohoto názvu pak stačí provést slovo **.CPU**.

.CPU (→)

vypíše na zadáném výstupním zařízení název použitého procesoru.

8. ZÁSOBNÍK NÁVRATOVÝCH ADRES

Nová slova:

>R — ($X \rightarrow$) ZNA: ($\rightarrow X$)
Uloží (TOS) UZ na TOS ZNA.

R> — ($\rightarrow X$) ZNA: ($X \rightarrow$)
Uloží (TOS) ZNA na TOS UZ.

Slova v lekci nadefinovaná:

R@ — ($\rightarrow X$) ZNA: ($X \rightarrow X$)
Nedestruktivní čtení ZNA.

EXIT — (\rightarrow)
Předčasné ukončení slova — RETURN uprostřed definice.

OVER — ($N1\ N2 \rightarrow N1\ N2\ N1$)
Zkopíruje (NOS) na TOS.

DDUP — ($N1\ N2 \rightarrow N1\ N2\ N1\ N2$)
Zduplikuje dvě vrchní položky na UZ.

ROT — ($N1\ N2\ N3 \rightarrow N2\ N3\ N1$)
Rotuje vrchní tři položky UZ vlevo, tj. tak, aby nový (TOS) byl původní (NNOS).

2ROT — ($N1\ N2\ N3 \rightarrow N3\ N1\ N2$)
Rotuje vrchní tři položky na UZ vpravo.

3DUP — ($N1\ N2\ N3 \rightarrow N1\ N2\ N3\ N1\ N2\ N3$)
Zduplikuje tři vrchní položky na UZ.

3PICK — ($N1\ N2\ N3 \rightarrow N1\ N2\ N3\ N1$)
Zkopíruje (NNOS) na TOS.

Již v úvodu jsem hovořil o tom, že FORTH používá 2 zásobníky, přičemž první je systémový a druhý uživatelský. Dopsud jsme používali pouze UZ. V této lekci se naučíme používat i ZNA.

Zásobník návratových adres (ZNA) se používá — jak ostatně napovídá již jeho název — pro uschování návratové adresy z procedury — slova, které je právě vykonáváno, tedy pro uschování adresy slova, které se má provádět po slově právě vykonávaném. Tato adresa se během provedení daného slova nemění, mohli bychom téměř říci, že se zásobník nepoužívá. Lze jej tedy využít pro přechodné uložení některých hodnot.

Před ukončením slova však musíme všechny tyto položky odebrat, aby při ukončení slova byl ZNA ve stejném stavu, v jakém byl na počátku slova. Jinak by se systém chtěl vrátit na návratovou adresu, která ve skutečnosti návratovou adresou není.

Druhým omezením, které musíme mít na paměti, je, že některá slova již ZNA používají a jeho „neoborným“ užitím bychom mohli jejich činnost ohrozit. Ještě jednou tedy opakuji: PRÍ POUŽITÍ ZNA JE TŘEBA NEJVĚTŠÍ OPATRNOST!

ZNA se většinou používá k přechodnému uschování položek, které nám na UZ překážejí, ale o nichž víme, že je budeme za chvíli potřebovat.

Slova, umožňující ukládání na ZNA a výběr prvků zpět, jsou >R a R>. Pokusme se pomoci nich nadefinovat slovo OVER:

: OVER >R DUP R> SWAP ;

N1	N1	N2	N1	N2	N1	N1
N2		N1		N1	N2	N1

To, že používání ZNA není zcela bez problému, si můžete demonstrovat na následujícím příkladu. Zkuste si, dříve než se pustíte do dalšího čtení, nadefinovat slovo R@, jež zkopíruje položku, kterou máme uloženou v ZNA na TOS UZ, anž by ji v ZNA zrušilo.

Hотово? Předpokládám, že řada z vás nadefinovala slovo R@ následovně:

: R@ R> DUP >R ;

X	RA	X	RA	X	RA	X
RA		X	RA	X	RA	X

FORTH

ing. R. Pečinovský, ČSc.

Tato definice sice zkopiruje (TOS) ZNA na TOS UZ, avšak v okamžiku počátku vykonávání každého slova je na TOS ZNA návratová adresa z tohoto slova, takže případná v ZNA uložená položka může být nejvýše v NOS. Takto nadefinované slovo by nám na TOS UZ zkopiropovalo pouze svoji návratovou adresu. Správná definice slova R@ je — jak již asi sami dokážete odhadnout:

:	R	R>	R>	DUP	>R	SWAP	>R	;
X	RA	X	RA	RA	X	X	X	
RA		X		X	RA		RA	
		X						

Na závěr si ukážeme, jak lze nadefinovat slovo EXIT, které ukončí provádění právě vykonávaného slova:

:	EXIT	R>	DROP	;
RA1	RA2	RA1		RA1
RA2				

Co se stane? Při vstupu do slova EXIT je v TOS ZNA návratová adresa ze slova EXIT (RA2) a předpokládáme, že v NOS ZNA je návratová adresa ze slova, odkud bylo slovo EXIT vyvoláno; tedy ze slova, které chceme ukončit. Smažeme tedy TOS ZNA a po ukončení slova EXIT bude výpočet pokračovat od adresy RA1.

Zkuste nadefinovat zbylá slova z druhé části slovníku v úvodu této lekce.

Kontrolní řešení:

:	DDUP	OVER	OVER	;
:	ROT	>R	SWAP	R> SWAP
:	2ROT	ROT	ROT	;
:	2ROT	SWAP	>R	SWAP R>
:	3DUP	>R	DDUP	R@ 2ROT R>
:	3PICK	>R	OVER	R> SWAP

9. PODMÍNĚNÝ PŘÍKAZ „IF . . . ENDIF“

Nová slova:

IF . . . (TRUE) . . . ENDIF

IF — ($F \rightarrow$)

Testuje (TOS) na jeho pravdivostní hodnotu. V případě „TRUE“ se posloupnost slov mezi IF a ENDIF vykoná, v případě „FALSE“ se přeskocí a pokračuje se rovnou prvním slovem za ENDIF.

ENDIF — (\rightarrow)

Označuje konec konstrukce IF . . . ENDIF.

NEGATE — ($N \rightarrow -N$)

Vynásobi (TOS) číslem -1.

DROP — ($X \rightarrow$)

Smaže (TOS)

QUIT — (\rightarrow)

Havarijní ukončení výpočtu — obdoba příkazu STOP. Další příkazy čte z klávesnice.

Slova v lekci nadefinovaná:

ABC — ($N \rightarrow INI$)

Uloží na TOS absolutní hodnotu (TOS)

?DUP — ($X \rightarrow X\ ?X?$)

Zduplikuje (TOS) pouze je-li různý od nuly.

MAX — ($N1\ N2 \rightarrow \max(N1, N2)$)

Uloží na TOS hodnotu většího z obou argumentů.

MIN — ($N1\ N2 \rightarrow \min(N1, N2)$)

Uloží na TOS hodnotu menšího z obou argumentů.

Další slova: **MAX3 MEZE** .(B)

FORTH je jedním z mála jazyků, které principiálně nemají příkaz skoku (GOTO). Pro realizaci různých programových struktur je zavedena řada slov, o jejichž významu si postupně povíme.

Nejzákladnější z těchto struktur je konstrukce IF . . . ENDIF. Jelikož si myslím, že její vysvětlení ve slovníku je dostatečné, ukážeme si ihned na příkladech její použití.

První slovo, které si v této lekci nadefinujeme bude slovo ABS, které nahradí (TOS) jeho absolutní hodnotou. Jedna z možných definic je:

:	ABS	DUP	0	IF	NEGATE	ENDIF	;
N	N	N		-N	N		
		N (N<0)					

Druhou ukázkou bude slovo MAX, které na TOS zanechá větší z čísel v TOS a NOS. Toto slovo lze nadefinovat:

:	MAX	DDUP	<	IF	SWAP	ENDIF	DROP	;
N1	N1	N1		N2	N1	min(N1,N2)	max(N1,N2)	
N2	N2	N2		N1	N1			
		(N1<N2)		N2				

Posledním příkladem v této lekci bude definice slova MEZE, které můžeme použít ke kontrole, zda index do pole leží v daných mezích. Za předpokladu, že velikost indexu je v (NOS) a jeho horní mez v (TOS), otestujte slovo MEZE, zda platí nerovnost

$0 \leq (NOS) < (TOS)$, a v případě, že neplatí, vytiskne chybové hlášení a ukončí běh programu (HM značí horní mez). Pokud nerovnost platí, program pokračuje dalším slovem.

:	MEZE	OVER	<=	SWAP	0 <	OR	IF	;
N	N	N	(HMEN)	(HMEN)	N	(NO)	(HMEN)	
HM	HM	HM		N	N			
							nebo	
							N<0	

CR CR ." INDEX MIMO MEZE" QUIT THEN ;

Zkuste si nyní sami naprogramovat slova MIN, ?DUP a slovo MAX3, které zanechá na TOS největší z hodnot (NNOS), (NOS) a (TOS).

Vašim dalším úkolem bude nadefinovat slovo .(B), které vytiskne (TOS stejně jako slovo ." avšak navíc v případě (BASE) <> 10 vytiskne za slovo do závorky (BASE) v desítkové soustavě.

Kontrolní řešení:

:	MIN	DDUP	> IF	SWAP	ENDIF	DROP	;
:	?DUP	DUP	0 <>	IF	DUP	ENDIF	
:	MAX3	MAX	MAX				

DEC (JELIKOŽ V DALŠÍM TEXTU PÍSEMÉ 2KRÁT CÍSLO, MUSÍME MÍT ZARUČENOU SPRÁVNOU ČÍSELNOU SOUSTAVU)

:	(B)	BASE	@	10	<>	IF	
:	"(B=)"	BASE	@	10	.B	."	"
							THEN

10. PODMÍNĚNÉ VĚTVENÍ „IF . . . ELSE . . . ENDIF“

Nova slova:

IF . . . (TRUE) . . . ELSE . . . (FALSE) . . . ENDIF

IF — ($F \rightarrow$)

Testuje TOS na jeho pravdivostní

(5)

hodnotu. V případě „TRUE“ se vykoná posloupnost slov mezi IF a ELSE, v případě „FALSE“ se vykoná posloupnost slov mezi ELSE a ENDIF. V obou případech se po vykonání patřičné posloupnosti počítače prvním slovem za slovem ENDIF.

ELSE—(—)

Ukončuje první posloupnost. Počítače se prvním slovem za ENDIF.

ENDIF—(—)

Ukončuje konstrukci.

*— (N1 N2 N3 → (N1*N2)/N3)

Uloží na TOS hodnotu výrazu (NNOS)*(NOS)/(TOS), přičemž mezinásobek je počítán ve dvojnásobné přesnosti.

Slova nadefinovaná v této lekcii:

KRONDEL SIGN SIN COS TAN

Místo dalekosáhlého úvodu přejdeme rovnou k příkladům. Nadefinujme funkci známou pod názvem „Kroneckerovo delta“. Je to funkce dvou argumentů, jež nabývá hodnoty 1 v případě jejich rovnosti a hodnoty 0 v případě nerovnosti. Její možná definice je:

: KRONDEL = IF 1 ELSE Ø ENDIF ;

Tato „doslovňá“ definice však může sloužit pouze jako školní příklad, neboť uvedenou funkci lze naprogramovat efektivněji:

: KRONDEL <> 1+ ; nebo

: KRONDEL = NEGATE ;

Trochu složitějším příkladem je funkce SIGN, která má jeden argument a jejím výsledkem je +1 v případě kladného argumentu, —1 v případě záporného argumentu a Ø v případě argumentu nulového.

: SIGN DUP Ø IF DROP —1
ELSE Ø= IF Ø ELSE 1
ENDIF ENDIF ;

I tuto funkci lze realizovat několika alternativními způsoby, jejichž nalezení bude vás úkol.

Uvedený příklad budí zároveň názornou ukázkou, jak nemá program v jazyku FORTH vypadat. Nebudeme-li využívat komentáře a vhodné grafické úpravy, stanou se zcela nepřehlednými i takovéto jednoduché programy.

Dalším úkolem bude naprogramovat výpočet funkcí sin, cos a tg podle algoritmu uvedeného v AR 11/82 a popsaného následujícímu struktogramy:

x = 35	x = 35
sin = x/60	x = 90-x sin = 1- x*x/7000

x = 55	x = 55
cos = 1- x*x/7000	x = 90-x cos = x/60

x = 39	x = 51	x = 51
tg = 50/x	tg = (2*x-33)/57	tg = 50/(90-x)

FORTH

Ing. R. Pecinovský, CSc.

Upravte algoritmus tak, aby výsledek slov byl např. 100 krát větší než skutečná hodnota funkce, neboť jak si zajistěte pamatuje. FORTH umí v základní verzi pracovat pouze s celými čísly.

11. CYKLUS S PARAMETREM I

Nova slova v této lekcii:

DO ... LOOP

DO — (N1 N2 →)

Očekává v TOS počáteční hodnotu parametru cyklu a v NOS hodnotu ukončovací, tedy první hodnotu, s níž se již cyklus provádět nebude.

LOOP — (→)

Zvětší parametr cyklu o jedničku a testuje, zda je menší než ukončovací hodnota. Pokud je, vykoná se další běh cyklu, pokud není, cyklus se ukončí.

| — (→ N)

Uloží na TOS hodnotu parametru nejvnitřejšího cyklu.

Slova v lekcii nadefinovaná:

PREVTAB SACHOVNICE RADEK
RADSACH NASOBILKA NA RADOPS
OBSACH

Další základní konstrukci, která se vyskytuje prakticky ve všech programovacích jazycích, je cyklus s parametrem. Tento cyklus je v jazyku FORTH uzavřeno mezi slovy DO a LOOP. Slovo DO zjistí na UZ počáteční a ukončovací hodnotu cyklu, smaže je a nastaví hodnotu parametru cyklu na hodnotu počáteční.

FORTH stejně jako FORTRAN, většina verzí jazyka BASIC a řada dalších jazyků testuje mezi parametru cyklu až na konci cyklu, a proto každý cyklus provede alespoň jednou.

Lépe než suchá teorie nám funkci cyklu ukáží příklady. Prvním příkladem bude vytisknutí tabulky převodu mezi jednotlivými číselnými soustavami. Při definici slova, tisknoucího tužku, využijeme slova PREV, které jsme nadefinovali v sedmé lekcii.

: PREVTAB DO | PREV LOOP CR ;

Takto nadefinováno, očekává PREVTAB v TOS prvně převáděné číslo a v NOS prvé číslo, které se již převádět nebude. Provedeme-li tedy:

DEC 27 23 PREVTAB

objeví se na obrazovce:

10111 (B) = 27 (0) = 23 (D) = 17 (H)
11000 (B) = 30 (0) = 24 (D) = 18 (H)
11001 (B) = 31 (0) = 25 (D) = 19 (H)
11010 (B) = 32 (0) = 26 (D) = 1A (H)

Druhým příkladem, který si ukážeme, bude tisk šachovnice. Každé poličko této šachovnice budou tvořit 2x3 znaky. Bílá polička budou vytvořena mezerami, tmavá polička znaky X. Levé horní poličko bude bílé.

Kontrolní řešení k 10. lekcii:

SIGN DUP 0<> IF	DUP ABS / ENDIF
SIGN DUP 0<= SWAP	0= NEGATE +
SIN DUP 35 / IF	(TEST: X < 35 ?)
10 / 6 / ELSE	(ANO: 100 * SIN = 10 * X/6)
90 SWAP —	(NE: X = 90 - X)
2NA 70 /	(TOS = x*x/70)
100 SWAP — ENDIF	(100 * SIN = 100 - x*x/70)
COS DUP 55 / IF	
2NA 70 / 100 SWAP — ELSE	
90 SWAP — 10 / 6 / ENDIF	
TAN DUP 39 / IF	
50 / 1 ELSE	
DUP 51 <= IF	
2 33 — 100 SWAP — 5000 / ELSE	
90 SWAP — ENDIF ENDIF	

ŠACHOVNICE

Ø PŘÍZNAK BĚLOSTI POLÍČKA
Ø DO CYKLUS TISKNUJÍCÍ RÁDKY SACHOVNICE
Ø 2 DO CR RÁDEK SACH. TJ. DVA RÁDKY TISKU
Ø Ø DO CYKLUS TISKNUJÍCÍ RÁDEK ZNAKU
DUP IF TEST PŘÍZNAKU BARVY POLE
“ XXX ” ELSE PŘÍZNAK → ČERNÉ POLE
“ ENDIF PŘÍZNAK = Ø → BÍLÉ POLE
NOT LOOP ZMĚNA BARVY PRO DALŠÍ POLE
LOOP KONEC TISKU RÁDKY SACHOVNICE
NOT ZMĚNA BARVY PO DALŠÍ RÁDKY
LOOP KONEC TISKU SACHOVNICE
DROP VYMAZANÍ PŘÍZNAKU BARVY

Uvedený příklad byl nejsložitějším slovem, které jsme doposud nadefinovali. Vysvětlíme si na něm několik zásad, kterých býchom se v jazyku FORTH měli držet.

První zásadou je důsledné komentování jednotlivých kroků programu. O nutnosti a zásadním významu komentářů se lehce přesvědčíte tím, že si je zakryjete a pokusíte se smysl programu dešifrovat bez nich. Bez komentářů ponechávejte jen ty opravdu nejkratší a „samovysvětlující se“ definice.

Druhou zásadou je nedefinovat příliš dlouhá a složitá slova. Nejen, že jsou nepřehledná, ale snáze se v nich udělají a hůře nacházejí chyby. Kdybychom např. chtěli předchozí příklad rozdělit do několika definic, mohli bychom ho naprogramovat např. takto:

RADEK (TOS = BARVA 1. SLOUPCE)
Ø Ø DO (CYKLUS TISKNUJÍCÍ RÁDEK ZNAKU)
DUP IF (TEST PŘÍZNAKU BARVY POLE)
“ XXX ” ELSE (PŘÍZNAK <> Ø → ČERNÉ POLE)
“ ENDIF (PŘÍZNAK = Ø → BÍLÉ POLE)
NOT LOOP (ZMĚNA BARVY PRO DALŠÍ POLE)

RADSACH (TOS = BARVA 1. SLOUPCE)
2 Ø DO (1. RÁDEK SACHOVNICE = 2. R. ZNAKU)
CR RADEK (TISK RÁDKU ZNAKU)
LOOP (DALŠÍ RÁDEK ZNAKU)

SACHOVNICE Ø (PŘÍZNAK BĚLOSTI POLE)
Ø Ø DO (CYKLUS TISKNUJÍCÍ RÁDKY SACHOVNICE)
RADSACH (TISK RÁDKU SACHOVNICE)
NOT (ZMĚNA BARVY PO DALŠÍ RÁDKY)
LOOP (KONEC TISKU SACHOVNICE)
DROP (VYMAZANÍ PŘÍZNAKU BARVY)

Třetí zásadou je zavést si nějakou grafickou úpravu, vyjadřující vnořenosť jednotlivých programových struktur, a tuto pak používat.

Uvedená pravidla jsou tím závažnější, čím složitější problém se snažíme naprogramovat.

Vaším úkolem bude definice slova **NÁSOBILKA**, které vytiskne v úhledné formě násobíku čísla, které najde na TOS.

Druhým úkolem bude slovo **NA**, které hodnotu z NOS umocní exponentem z TOS. Kontrolní řešení:

: **NASOBILKA** 10 1 DO

(NÁSOBKY OD 1 DO 9)

CR (PRECHOD NA NOVÝ RADEK)

DUP (USCHOVÁNÍ NÁSOBENÉHO ČÍSLA)

I DDUP . . . * . . . = " *

LOOP

DROP (SMAZÁNÍ NÁSOBENÉHO ČÍSLA)

; : **NA** (UMOCŇOVÁNÍ — NOS = ZÁKLAD, TOS EXPONENT)

1 SWAP Ø DO

(PŘÍPRAVA MEZIVÝSLEDKU)

OVER * LOOP

(NOS = ZÁKLAD, TOS = MEZIVÝSLEDEK)

SWAP DROP (TOS = VÝSLEDEK)

; : **NA** (UMOCŇOVÁNÍ S KONTROLAMI — NOS = ZAKLAD, TOS - EXPONENT).

OVER Ø = IF

(TEST NULOVOSTI ZÁKLADU)

Ø <= IF

(ZÁKLAD NULOVÝ → TEST EXPONENTU)

“ **ARITMETICKÁ CHYBA**”

(EXP <= Ø → NEDEFINOVANÝ VÝSLEDEK)

QUIT ELSE

(HAVARIJNÍ UKONČENÍ VÝPOČTU)

Ø ENDIF

(EXP > Ø → VÝSLEDEK = Ø)

DUP Ø < IF

(TEST ZÁPORNOH EXPONENTU)

DROP Ø ELSE

(EXP < Ø → [VÝSL] < 1 → VÝSL = Ø)

DUP Ø = IF

(EXPONENT NULOVÝ ?)

DROP 1 ELSE

(ANO → VÝSLEDEK = 1)

1 SWAP Ø DO

(NE → NORMÁLNÍ UMOCNĚNÍ)

OVER * LOOP

(NOS = ZÁKLAD, TOS = MEZIVÝSLEDEK)

ENDIF (UKONČENÍ KONSTRUKCÍ IF)

ENDIF

ENDIF

SWAP DROP (TOS = VÝSLEDEK)

FORTH

Ing. R. Pečinovský, ČSc.

LEAVE — (→)

Nastaví ukončení cyklu při příštím vykonávání slova **LOOP**, nebo - **LOOP** tím, že položí hodnotu ukončovací rovnou hodnotě parametru cyklu.

Slova v lekci nadefinovaná:

PRVOČÍSLA I J LEAVE ARPR

Při programování velmi často potřebujeme cyklus s přírůstek jiným, než jedničkovým. Takový cyklus můžeme v jazyku FORTH realizovat pomocí slov **DO** a + **LOOP**. Slovo **DO** pracuje stejně, jako u cyklu uvedeného v minulé lekci. Slovo + **LOOP** k parametru cyklu připočte hodnotu, kterou nalezneme na TOS. Test, který pak provádí, záleží na znaménku přírůstku cyklu. Pokud je přírůstek kladný, je test stejný jako u slova **LOOP**, tedy „I < IMAX“. Pokud je přírůstek cyklu záporný, je test negaci původního, tedy „I > IMAX“. V případě, že je odpovídající podmínka (test) splněna, nic nebrání tomu, aby cyklus pokračoval s novou hodnotou parametru. Pokud podmínka splněna není, cyklus se ukončí a pokračuje se prvním slovem za slovem - **LOOP**.

Jako příklad použití takového cyklu si uvedeme definici slova, které spočte všechna prvočísla z intervalu definovaného na UZ.

: **PRVOČÍSLA**

(NOS = HORNÍ MEZ, TOS = SPODNÍ MEZ)

(PŘEDPOKLAD: SPODNÍ MEZ > 10)

DUP 2 MOD NOT IF

(TEST LICHOSTI SPODNÍ MEZE)

1+ ENDIF

(UŽ JE LICHÁ)

DO (TEST ČÍSEL Z DANÉHO INTERVALU)

-1

(PŘEPOKLADAME, ZE JE PRVOČÍSLEM)

I 3 / 3 DO

(DĚLÍME ČÍSLY OD 3 DO 1/3)

J I MOD IF

(TEST DĚLITENOSTI)

NOT LEAVE ENDIF

(JE DĚLITELNÉ → NENÍ PRVOČÍSLO)

2 +LOOP

(DĚLME DALŠÍM LICHÝM ČÍSEM)

IF (TOS = PŘÍZNAK PRVOČÍSLOSTI)

I . ENDIF

(PRVOČÍSLO → VYTISKNOUT)

2 +LOOP

(TEST DALŠÍHO LICHÉHO ČÍSLA)

Druhou zvláštností cyklu je, že u většiny verzí jazyka FORTH slovo **DO** ukládá ukončovací i počáteční hodnotu parametru na zásobník návratových adres, odkud si je slova **LOOP**, - **LOOP**, I, J, a **LEAVE** berou. Avšak pozor! Ze stejných důvodů, které jsme rozebrali u definice slova R v 8. lekci, nemůžeme nadefinovat:

: I R@ ; ,
jelikož by takto nadefinované slovo vrácelo v TOS UZ svou návratovou adresu!

Z toho vyplývá, že chceme-li užít ZNA uvnitř cyklu, musíme zařídit, aby toto slovo (**LOOP**, - **LOOP**, ...) našla ZNA ve stavu, v jakém ho zanechalo předchozí slovo z uvedené množiny nebo slovo **DO**. Odborně, chceme-li použít položku, kterou jsme na ZNA zanechali před vstupem do cyklu, musíme počítat s tím, že **DO** za tuto položku připsalo další dvě.

Pokud používáme ZNA pouze vně cyklu, nekladou na nás slova realizující cyklus žádná omezení a můžeme se ZNA pracovat tak, jak jsme byli doposud zvykli.

Pokusete se nadefinovat slovo **ARPR**, které spočte aritmetický průměr N vrchních položek UZ, kde N = (TOS).

Dále se pokusete navrhnut definici slov I, J a **LEAVE**.

Posledním úkolem bude nadefinovat slovo, které by vytisklo šachovnici, přičemž (TOS) = počet sloupců a (NOS) = počet řádků této šachovnice. Slovo **RADEK** nadefinujte takto:

: **RADEK CR Ø DO DUP IF**
" XXX" ELSE " ENDIF
NOT LOOP DROP ;

Kontrolní řešení:

: **ARPR DUP >R Ø DO**
+ **LOOP R> / ;**
: **I R> R@ SWAP >R ;**
(RYCHLEJŠÍ JE NADEFINOVAT JE STEJNĚ JAKO R@)
: **J R> R> R> R@ 2ROT >R >R SWAP >R ;**
: **LEAVE R> R> R> DUP >R >R DROP >R ;**
: **RADOB**

(RADEK OBECNÉ ŠACHOVNICE — NOS = PŘÍZNAK BARVY 1. SLOUPEČ, TOS = POČET SLOUPCI)

	DDUP	2	Ø	DO
F	F	F	F	F
NS	NS	NS	NS	NS
	F	F	F	F
	NS	NS	NS	NS
		2	2	2

RADEK LOOP ;

F F ;

(HODNOTY OZNAČENÉ ? JSOU NA UZ POUZE PRI PRVNIM BEHU CYKLEM)

: **OBSACH**

(OBECNÁ ŠACHOVNICE — TOS = POČET ŘÁDKŮ, NOS = POČET SLOUPCI)

Ø SWAP

(NNOS = NS, NOS = PŘÍZNAK BARVY 1. SL., TOS = NR)

Ø DO

(CYKLUS TISKNOUcí JEDNOTLIVÉ ŘÁDKY ŠACHOVNICE)

OVER NOT OVER RADOB

(ZMĚNA BARVY 1. POLE A NATÍSTENÍ JEDNOHO ŘÁDKU)

NOT LOOP

DROP DROP

(SMAZÁNÍ PŘÍZNAKU BARVY A POČTU SLOUPCI)

;

13. CYKLY S PODMÍNKOU

Nová slova:

BEGIN ... UNTIL

BEGIN — (→)

Slouží jako návěstí. Označuje začátek cyklu.

UNTIL — (F →)

Testuje (TOS) na jeho pravdivostní hodnotu. V případě „FALSE“ opakuje cyklus (= pokračuje znova od **BEGIN**), v případě „TRUE“ jej ukončí a pokračuje dál.

BEGIN ... WHILE ... REPEAT

BEGIN — (→)

Označuje začátek konstrukce.

WHILE — (F →)

Testuje (TOS) na jeho pravdivost.

(7)

FORTH

vostní hodnotu. V případě „TRUE“ se posloupnost slov mezi **WHILE** a **REPEAT** vykoná, v případě „FALSE“ se pokračuje prvním slovem za slovem **REPEAT**.

REPEAT — (→)

Je posledním slovem cyklu. Vrací výpočet zpět za **BEGIN** (funguje jako GOTO BEGIN).

Slova v lekci nadefinovaná:

EUKL NSN

Kromě cyklu s parametrem obsahují moderní programovací jazyky (BASIC ani FORTRAN k nim nepatří) i cykly, jejichž vykonávání je řízeno platnosti či neplatnosti nějaké podmínky. Obdobné možnosti jsou i součástí standardní verze jazyka FORTH.

První z těchto konstrukcí je cyklus **BEGIN ... UNTIL**. Slovo **BEGIN** cyklus pouze uvozuje a nemá pro jeho provádění jiný význam, než jako návěští. Slovo **UNTIL** testuje (TOS) a nechává cyklus opakovat tak dlouho, dokud není při testu (TOS) = „TRUE“.

Jako příklad si naprogramujeme slovo **EUKL**, které spočte podle Euklidova algoritmu největšího společného dělitele (NOS) a (TOS).

: **EUKL BEGIN**

(SPOČÍTÁ NEJVĚTŠÍHO SPOLEČNÉHO DĚLITELE
TOS A NOS)

DDUP > IF SWAP ENDIF

(TOS = VĚTŠÍ Z OBOU ČISEL)

OVER — (TOS = ROZDÍL OBOU ČISEL)

DDUP =

UNTIL (POKUD NE, OPAKUJ)

DROP (TOS = NEJVĚTŠÍ SPOLEČNÝ DĚLITEL)

Druhým z cyklů s podmínkou je cyklus „**BEGIN ... WHILE ... REPEAT**“. Tento cyklus, na rozdíl od cyklu předchozího, testuje (TOS) na konci, ale již na začátku cyklu. Slovo **BEGIN** zde opět hraje úlohu návěští, uvozujícího celou konstrukci. Mezi slovy **BEGIN** a **WHILE** je třeba spočítat podmínu a umístit její výsledek na TOS. Slovo **WHILE** testuje (TOS) na jeho logickou hodnotu a v případě „TRUE“ provede tělo cyklu končící slovem **REPEAT**, které vrátí běh programu zpět za **BEGIN**. Je-li (TOS) = „FALSE“, ukončí se cyklus a pokračuje se prvním slovem za slovem **REPEAT**.

Na ukázku si naprogramujeme stejný příklad s pomocí cyklu **BEGIN...WHILE ...REPEAT**:

: **EUKL BEGIN**

(SPOČÍTÁ NEJVĚTŠÍ SPOLEČNÝ DĚLITEL TOS,
NOS)

DDUP < > WHILE

(TĚLO SE PROVEDE POKUD NOS < > TOS)

**DDUP > IF
SWAP ENDIF**

(TOS = VĚTŠÍ Z OBOU ČISEL)

OVER —

(TOS = ROZDÍL OBOU ČISEL)

REPEAT

(OPAKUJ CYKLUS)

DROP

(TOS = NEJVĚTŠÍ SPOLEČNÝ DĚLITEL)

Pokuste se nadefinovat slovo **NSN**, které zanechá na TOS nejmenší společný násobek NOS a TOS. Zkuste obě možnosti realizace cyklu s podmínkou.

Kontrolní řešení:

: **NSN**

DDUP < IF SWAP ENDIF

(TOS = MENŠÍ Z OBOU ČISEL)

>R Ø BEGIN

(NOS = MENŠÍ ČISLO, TOS = ODHAD NSN)

OVER - (TOS = NOVÝ ODHAD)

DUP R@ MOD Ø

(JE DĚLITELNÝ DRUHÝM ČISLEM ?)

UNTIL

(DOKUD NENÍ OPAKUJ CYKLUS)

SWAP DROP R > DROP

(SMAZÁN OBOU ČISEL)

;

: **NSN**

DDUP < IF SWAP ENDIF

(TOS = MENŠÍ Z OBOU ČISEL)

>R DUP BEGIN

(NOS = MENŠÍ ČISLO, TOS = ODHAD NSN)

DUP R@ MOD

(JE DĚLITELNÝ DRUHÝM ČISLEM)

WHILE

(POKUD NE, SPOČÍTEJ DALŠÍ ODHAD)

OVER + (TOS = NOVÝ ODHAD)

REPEAT

SWAP DROP R > DROP

(SMAZÁN OBOU ARGUMENTŮ)

14. VNITŘNÍ STRUKTURA SLOVNÍKU

Nová slova:

VARIABLE xxx - (N →)

Definuje proměnnou **xxx** a nastaví její počáteční hodnotu = (TOS). Při vykonání slova **xxx** se na TOS uloží adresa paměťového místa, v němž je uložena hodnota této proměnné.

CONSTANT xxx - (N →)

Definuje konstantu **xxx** s hodnotou rovnou (TOS). Při vykonání slova **xxx** se tato hodnota uloží na TOS.

? (A →)

Vytiskne hodnotu na adresu A.

Slova v lekci nadefinovaná: **A B**

Ve slovníku jsou uložena všechna slova, která nás FORTH zná. Záznam každého slova můžeme rozdělit na dvě části — na hlavičku a na tělo slova. Hlavička začíná bajtem, udávajícím počet písmen názvu a obsahujícím ještě některé další informace, za ním následuje jméno slova, kodované v ASCII. Dvoubajková položka označená **SA** je spojovací adresa, což je adresa počátku hlavičky předchozího slova. Tato položka bývá často označována **LA** (Link Address).

DJ	„jméno“	SA	AVCP
----	---------	----	------

DJ – délka jména (počet znaků)

SA – spojovací adresa (2 bajty)

AVCP – adresa výkonné části překladače (2 bajty)

Hledá-li se nějaké slovo ve slovníku, začne se od posledního nadefinovaného slova. Není-li to hledané slovo, testuje se slovo předposlední a tak dále, až se hledané slovo najde, nebo až se narazí na první slovo slovníku, které má **SA=0**.

Poslední položka v hlavičce, označená AVCP, je adresa výkonné části překladače, což je program ve strojovém kódu. Ten chápe tělo slova jako pole parametrů, které má zpracovat. Tuto adresu budeme označovat apostrofy ('xxx').

V literatuře bývá adresa buňky, obsahující AVCP, označována CFA (Code pointer Field Address = adresa, na níž je uložen ukazatel na

podprogram ve strojovém kódu). Pro úplnost ještě dodám, že adresa prvního bajtu hlavičky, v níž je uložen počet písmen názvu, se značí NFA (Name Field Address), adresa buňky, v níž je uložena SA se označuje LFA (Link Field Address) a adresa počátku těla PFA (Parameter Field Address).

Pokud je slovo nadefinováno pomocí dvojtečkové definice, je jeho tělo tvořeno seznámením adres slov, která se mají vykonat. Poslední adresou je adresa slova **EXIT**, které nahrazuje vám jistě známé RETURN (v některých verzích je toto slovo označeno „S“).

Dvojtečka je jedním z překladačů. Jiné překladače mohou samozřejmě nadefinovat tělo slova, neboli parametry pro svoji výkonnou část, po svém.

Zde bych chtěl upozornit na to, že pokud budu hovořit o adrese slova, budu tím myslit vždy adresu poslední položky hlavičky, v níž je uložena AVCP. Budu-li hovořit o jiné adrese, vždy na to výslovně upozorním.

Po tomto úvodu přistoupíme k vlastním definičním slovům (kompilátorům, překladačům). Jak jsme si již řekli, každé definiční slovo (překladač) překládá jím definované slovo po svém. Dopsud známe pouze jediné definiční slovo — :. Slova definovaná pomocí dvojtečky mají do svého těla zapsány adresy slov tak, jak mají být postupně vykonána.

adresa	obsah	poznámka
101	4	délka jména
102	D	
103	D	jméno (v ASCII)
104	U	
105	P	
106	SA	SA
107		
108	"	AVCP
109		
110	.OVER.	
111		
112	.OVER.	
113		
114	.EXIT.	
115		
116	3	délka jména
117	R	
118	O	jméno
119	T	
120	101	SA
121		
122	"	AVCP
123		

Všimneme si ještě způsobu, kterým se používá překladače zapisuje. V textu je vždy uvedeno napřed jméno překladače (v našem případě :) a za ním jméno právě definovaného slova (např. **DDUP**). Tento způsob zápisu platí i pro všechny ostatní překladače.

Do standardní verze patří kromě : ještě překladače **VARIABLE** a **CONSTANT**. Definujeme-li nové slovo pomocí **VARIABLE**, vyhradí se ve slovníku za hlavičkou dva bajty, do nichž se uloží obsah TOS. Kdykoliv pak takto definované slovo vyzvoláme, uloží se na TOS adresa těchto dvou bajtů. Na tuto adresu pak můžeme ukládat nové hodnoty nebo je z ní naopak vyzvednout. Slovo tedy můžeme používat jako proměnnou. Použití tohoto překladače může být např. následovně:

1 VARIABLE A 2 VARIABLE B

A	@	B	@	+	A	!	A	?
(A).	1	1	1	3	3		(A).	

Po vykonání této posloupnosti slov nám počítac vytiskne na obrazovku číslo 3.

Příkladem slova, které bychom mohli nadefinovat pomocí překladače **VARIABLE**, je nám již známé slovo **BASE**.

Definujeme-li nové slovo pomocí překladače **CONSTANT**, vyhradí se ve slovníku také dva bajty, do nichž se uloží hodnota TOS, ale každé vyvolání takto definovaného slova uloží na TOS ne adresu, ale obsah výše uvedených dvou bajtů. Slovo se tedy dá použít jako konstanta.

Používání slov definovaných jako konstanty nám šetří paměť. Pokud používáme v programu nějaké číslo, uloží se do slovníku adresa slova **LIT** (viz 18. lekce) a za ní hodnota čísla, které chceme použít. Při vykonání uloží slovo **LIT** hodnotu, která za ním následuje, na TOS. Kolikrát nějaké číslo použijeme, kolikrát potřebujeme dva bajty navíc. Proto jsou nejpoužívanější čísla (0, 1, -1, 2) nadefinována jako slova jazyka FORTH pomocí překladače **CONSTANT**.

Než začnete čist dál, rozmyslete si, jaká bude odpověď počítače na řádek

BIN 1 . 2 . 4 .

Hotovo? Tak zde je řešení. Počítač na obrazovce vytiskne

FORTH 602:

1 10
4 CHYBNE NAPSANE SLOVO
fig-FORTH
1 10 4 ? MSG# 0

Proč? Slovo **BIN** přepnulo vstup a výstup na binární soustavu. Slovo 1 uložilo na TOS hodnotu 1, kterou slovo . vytisklo. Slovo 2 uložilo na TOS hodnotu 2 (slovo 2 je definováno jako konstanta), kterou opět slovo . vytisklo. Slovo 4 překladač ve slovníku nenašel a proto se ho pokusil interpretovat jako číslo. Avšak binární soustava zna pouze číslice 0 a 1 a proto počítač ohlásil chybu.

15. DEFINOVÁNÍ NOVÝCH PŘEKLADAČŮ

Nová slova:

- (X →)
Cárka – uloží (TOS) do slovníku.

<BUILD> - (→)
Vytvoří hlavičku nově definovaného slova.

DOES> - (→)
Ukončí komplikaci nově definovaného slova a nastaví ukazatel na výkonné části překladače.

C@ - (A → B(A))
Uloží do spodních osmi bitů TOS obsah bajtu na adrese A. Horních osm bitů nuluje.

C! - (B A →)
Uloží do bajtu na adrese A obsah spodních osmi bitů NOS.

C, - (B →)
Uloží do slovníku spodní bajt TOS.

Slova v lekci nadefinovaná:

CVARIABLE CCONSTANT

Tři základní překladače již známe. Jak ale nadefinovat překladače nové? Zde přichází na řadu dvě „magická“ slova <BUILD> a **DOES>**. Spičaté závorky na začátku prvního a na konci druhého slova symbolizují, že tato slova musíme vždy použít obě v jedné definici a v uvedeném pořadí.

Tato dvě slova teprve dělají FORTH Fortem. Dopsud se liší od ostatních programovacích jazyků pouze svým poněkud „divokým“ zápisem. Nyní však odhalíme jeho schopnost kvalitativně rozšířovat sám sebe.

Způsob implementace této dvou slov se v různých verzích jazyka FORTH poněkud liší. V našem výkladu budeme vycházet z přístupu, který použili autori systému FORTH 602.

FORTH

Ing. Rudolf Pečinovský, CSc.

Operace kolem nových překladačů můžeme rozdělit do tří fází, které si ilustrujeme na příkladu překladače **CONSTANT**. Pro strukturu budeme v dalším textu nový překladač značit NP a jím nadefinované nové slovo NS.

1. Definice NP pomocí překladače „“:
: CONSTANT <BUILD>, DOES> @ ;
2. Použití NP k definici NS:
3 CONSTANT TRI
3. Použití NS:
DEC TRI . BIN TRI .

Co se stalo? V první fázi jsme nadefinovali překladač **CONSTANT** a tím zařadili jeho definici do slovníku.

adresa	obsah	poznámka
501	8	délka jména
502	C	
503	O	
504	N	
505	S	
506	T	
507	A	
508	N	
509	T	
510	485	SA
511		
512	..	AVCP
513		
514	<BUILD>	
515		
516		
517	...	
518	.DOES>c.	
519		
520	JMP.DOES>e.	
521		
522		
523	@.	
524		
525	.EXIT.	
526		
527	3	délka jména
528	T	
529	R	
530	I	
531	'CONSTANT'=520	SA
532		
533	3	
534		

V druhé fázi jsme pomocí tohoto překladače nadefinovali slovo **TRI**. Činnost, vykonávaná pod bodem 2 bude následující:

- 3 — toto slovo je pochopeno jako číslo a jeho hodnota se uloží na TOS.
- CONSTANT** při vykonávání tohoto slova se postupně provede jeho definice (podle bodu 1).
- <BUILD> — zřídí hlavičku slova se jménem, které je zapsáno za slovem **CONSTANT**, v našem případě se jménem **TRI**. SA bude ukazovat na počátek hlavičky předchozího slova (v našem případě slova **CONSTANT**). Vyhradí se i místo pro AVCP, ale ta se prozatím nenastavuje.
- vezme TOS (=3) a přidá jeho hodnotu na konec slovníku, tedy v našem případě za místo vyhrazené pro AVCP.

DOES> — jak jste si možná všimli slovo **DOES>** se překládá jinak než běžná slova jazyka FORTH. Na rozdíl od nich zabírá ve slovníku dvě položky. První položku je adresa kompilační části slova **DOES>**, která ukončí fázi komplikace, tedy fázi, v níž definujeme NS (slovo **TRI**) a nastaví jeho AVCP na počátek druhé položky. Tato položka je ve strojovém kódu naprogramovaným skokem na exekuční část slova **DOES>**, což je program ve strojovém kódu, který teprve spustí výkonnou část překladače.

Ptáte se, proč tak složité? V minulé kapitole jsme si řekli, že AVCP ukazuje na podprogram ve strojovém kódu, ale výkonná část našeho překladače je psaná v jazyce FORTH. Proto musíme interpret nejprve „přepnout“ ze strojového kódu na FORTH. Na podprogram **DOES>**, realizující toto „přepnutí“, jsme nemohli skočit přímo proto, že bychom pak nevěděli, kde hledat výkonnou část našeho překladače.

Vráťme se ale k našemu příkladu. Ve třetí fázi NS **TRI** použijeme. Činnost bude následující:

DEC — nastaví se desítková soustava,
TRI — začne se vykonávat činnost definovaná překladačem. Na její počátek ukazuje nepřímo AVCP v hlavičce NS. Zde je to adresa skoku na exekuční část slova **DOES>**. Tato část:

- 1) Uloží adresu počátku těla NS na TOS.
- 2) Spustí výkonnou část překladače **CONSTANT**.

Prověde se tedy:
@ — vezme obsah na adresu, kterou najde na TOS a uloží jej na TOS místo této adresy. Po jeho vykonání bude tedy (TOS)=3.

EXIT — ukončí provedení výkonné části překladače a tím i slova **TRI**.

— vytiskne (TOS) na obrazovku,
BIN — nastaví vyjadřování čísel ve dvojkové soustavě,
TRI — vykoná se podobně jak bylo již popsáno, výsledkem je uložení čísla 3 na TOS.
— vytiskne (TOS) ve dvojkové soustavě.

Po vykonání celé sekvence se tedy na obrazovce objeví:

3 11 OK

kde „OK“ oznamuje, že počítač s úspěchem dokončil požadovanou činnost.

Shrňme si tedy probrané:

Pomoci dvojčekové definice můžeme nadefinovat nový překladač. Tato definice se skládá ze tří částí:

1) **Překladační část** (od jména NP po slovo <BUILD>). V této části je nadefinována činnost, která se má provést předtím, než se vytvoří hlavička tímto překladačem definovaného nového slova (NS). Tato část bývá velmi často prázdná.

2) **Kompilační část překladače** (od slova <BUILD> po slovo **DOES>** včetně).

Slovo <BUILD> vytvoří hlavičku NS, přičemž prozatím nenastavuje AVCP. Za slovem <BUILD> následuje popis činnosti, která se má vykonat během defino-

(9)

vání NS pomocí tohoto NP. Je to vlastně postup, jak NP vytváří NS. Nakonec slovo **DOES** nastaví AVCP a ukončí komplikaci.

3) Výkonná část překladače (od slova **DOES** do konce definice)

Tato část se začne vykonávat při použití NS. Začne „výkonnou částí“ slova **DOES**. Ta uloží adresu počátku těla NS na TOS a spustí vykonné část překladače, tj. činnost popsanou v definici NP za slovem **DOES**.

Pochopení lekce si zkuste ověřit na návrhu překladače **VARIABLE** a překladače **CVARIABLE** a **CCONSTANT**, které definují jednobajtovou proměnnou a konstantu.

Kontrolní řešení:

```
: VARIABLE <BUILDS , DOES> ;
: CVARIABLE <BUILDS C, DOES> ;
: CCONSTANT <BUILDS C, DOES> C@ ;
```

16. ZÁKLADNÍ DATOVÉ STRUKTURY

Nová slova:

ALLOT - ($N \rightarrow$)
(VYHRADI VE SLOVNÍKU N BAJTŮ)

Slova v lekci nadefinovaná:

VEKTOR **MATICE** **ARPRV** **ZAPLN**
() **VEK** **TABULKA**

V této lekci se již odpoutáme od teorie a vyšvětlíme si použití slov **<BUILDS** a **DOES** přímo na příkladech. NP jsou nejčastěji používány při definování nových datových struktur. Dejme tomu, že bychom potřebovali, aby nás FORTH uměl pracovat s vektory. Nadefinujeme si proto následující kompilátor:

```
: VEKTOR
    ( TOS = POČET POLOŽEK VE VEKTORU )
<BUILDS ( VYTVOŘENÍ HLAVICKY )
2* + ( 2 BAJTY NA KAŽDOU POLOŽKU )
ALLOT ( VYHRAZENÍ MÍSTA )
( VÝKONNÁ ČÁST - OČEKÁVÁ NA TOS POŘADÍ POLOŽKY )
DOES > ( NOS = POŘADÍ POLOŽKY, TOS = PFA )
SWAP ( NOS = PFA, TOS = POŘADÍ POLOŽKY )
2* ( TOS = VZDÁLENOST POLOŽKY
     OD POČÁTKU TĚLA V BAJTECH )
+ ( TOS = ADRESA HLEDANÉ POLOŽKY ) ;
```

Tento překladač očekává při definici NS na TOS počet položek, pro něž se má vyhradit místo ve slovníku. Každá položka zaujímá 2 bajty. Před použitím NS musíme na TOS umístit pořadí položky, které nás zajímají. Po vykonání NS pak na TOS obdržíme adresu této položky (vzhledem k očekávanému použití cyklu budeme položky číslovat od nuly do N-1, kde N je celkový počet položek ve vektoru).

Takto definovaný překladač při definici NS pouze vyhradí pro toto slovo místo. Chceme-li, aby při definici vektoru byla zároveň jeho prvkům přiřazena nulová počáteční hodnota, můžeme definici překladače upravit, např.:

```
: VEKTOR <BUILDS 0 DO 0 ,
LOOP DOES> SWAP 2* + ;
```

Sami si zkuste nadefinovat překladač, který kromě počtu složek v TOS očekává ještě v NOS počáteční hodnotu, kterou má přiřadit všem prvkům definovaného vektoru.

Takto definované vektory ovšem nekontroluje případná přetečení indexu. Pokud bychom chtěli přidat hlášení chyb při opuštění

FORTH

Ing. Rudolf Pečinovský, CSc.

mezi, museli bychom vektory nadefinovat např. následovně (verze vhodná pro ladění programu):

```
: VEKTOR
<BUILDS ( VYTVOŘENÍ HLAVICKY )
DUP , ( ULOŽENÍ DÉLKY VEKTORU )
0 DO 0 , LOOP ( NULOVÁ POČATEČNÍ HODNOTA )
DOES>
DDUP MEZE ( TEST: 0 <= INDEX <
               < HORNÍ MEZ )
2+ SWAP 2* +
( VŠE V POŘÁDKU, ULOŽ NA TOS
ADRESU ) ;
```

Každá kontrola je náročná nejen na práci programátora, ale i na kapacitu paměti a operační dobu. Proto se raději většinou žádné havarijní situace (přetečení mezi poli nebo výsledku aritmetických operací, podtečení zásobníku ap.) nehledají a vystříhání se těchto stavů je věcí programátora. Výhodné je nadefinovat si slova s kontrolami pro ladění programu a pro odladěnou verzi používat slova bez kontrol.

Ukažme si nyní, jak lze vektor podle poslední definice, použít v programu. Definujme:

```
10 VEKTOR VEK
( DEFINOVÁNÍ VEKTORU S 10 PRVKY )
: ARPRV 0 10 0 DO I VEK +
LOOP 10 / ;
( VÝPOČTE ARITMETICKÝ PRŮMĚR PRVKŮ VEK-
TORU VEK )
```

Nyní zadáme

ARPRV .

Systém by měl odpovědět
0 OK

V tomto příkladu se již ukázaly některé slabiny naší definice. Slovo **ARPRV** umělo pracovat pouze s vektorem **VEK** a s žádným jiným. Práce s obecným vektorem by se při této definici vektoru programovala těžko. Jednou z možností je postup typu:

```
: ARPRV ( NOS = PFA,
           TOS = POČET POLOŽEK )
>R
0 SWAP R@ 0
( NNNOS = PFA, NNNOS = SUM = 0 )
DO OVER I
( NNNOS = SUM, NOS = PFA,
   TOS = POŘADÍ POLOŽKY )
2* + ( POČÍTEJ ADRESU I-TÉ POLOŽKY )
+ ( PŘIPOČÍTI J K MEZISOUČTU )
LOOP R@ /
( VYDĚL SOUČTEM PRVKŮ )
```

Takto definované slovo bychom pak mohli použít v posloupnosti

0 VEK 10 ARPRV

Sami jistě vidíte, že toto řešení je poněkud těžkopádné. Pokusme se tedy vydat jinou cestou a upravit přímo definici překladače **VEKTOR**, např.:

```
: VEKTOR <BUILDS DUP , 2*
ALLOT DOES> ;
```

Nadefinujeme-li ještě slovo

```
: () SWAP 2* + ;
popr.
: () DDUP @ MEZE SWAP 2* + ;
```

které nám bude sloužit k podobným účelům, k nimž doposud sloužila výkonná část překladače **VEKTOR**, nic nám nebrání pracovat jak s vektorem jako celkem, tak s jeho jednotlivými složkami.

Tuto definici můžeme ještě zobecnit a nadefinovat si překladač **VEKTOR** tak, aby bychom mohli s vektory pracovat jednotně, ať půjde o vektory bajtů, dvoubajtových čísel nebo vektory vektorů.

```
: VEKTOR
<BUILDS ( NOS = POČET POLOŽEK,
           TOS = DÉLKA POLOŽKY )
DDUP C, C,
( POČET POLOŽEK I JEJICH DÉLK.
KA < 256 )
( TIM SI UMÔZNIME, ABYCHOM MOHLI POZDĚJI
  PRACOVAT S ŘETËZCI - BUDOU VYSVĚTLENY
  V 17. LEKCI - JAKO S VEKTORY ZNAKÙ )
* ALLOT ( VYHRAZENÍ MÍSTA V PAMËTI )
DOES>
```

Způsob uložení vektoru ve slovníku:

DJ	jmeno	SA	„VEKTOR“	délka	počet pol.
0. položka	1. položka			

```
: () ( NOS = POŘADÍ POLOŽKY,
      TOS = PFA vektoru )
DUP >R ( USCHOVÁNÍ ADRESY DÉLKY
           POLOŽKY = PFA )
C@ * ( TOS = VZDÁLENOST POČÁTKU
           POLOŽKY OD PFA+2 )
R> 2+ + ( ADRESA HLEDANÉ POLOŽKY )
```

Pro inicializaci vektoru bychom si pak mohli nadefinovat slovo, které očekává v TOS adresu počátku těla, tedy adresu položky, v níž je uložena délka vektoru:

```
: NULUJ ( TOS = PFA )
DUP 2+ >R ( USCHOVÁNÍ ADRESY NULTÉ
           POLOŽKY, TOS = PFA )
C@ R@ 1- C@ ( NOS = DÉLKA,
                 TOS = POČET POLOŽEK )
* R@ + R@ ( NOS, TOS - PARAMETRY PRO
           CYKLUS )
DO 0 I C1 LOOP ( VYNULOVÁNÍ VŠECH BAJTÙ
                  VEKTORU OD PFA+2 )
```

Vektor **VEK** bychom pak definovali a inicializovali takto:

```
2 10 VEKTOR VEK VEK NULUJ
Výhodou tohoto řešení je, že v případě, kdy není třeba vynulovat složky vektoru, nenuluji se.
```

Náš příklad s aritmetickým průměrem složek vektoru pak lze nadefinovat např. takto:

```
: ARPRV ( TOS = PFA )
>R 0 ( USCHOVÁNÍ PFA, TOS = 0 - INI-
       CIALIZACE SOUČTU )
R@ 1+ C@ 0 ( CYKLUS OD NULY DO POČTU
           PRVKÙ )
DO I () + LOOP
R> C@ / ( VYDĚLENÍ SOUČTU POČTEM
           PRVKÙ )
```

Slovo bychom pak použili v posloupnosti **VEK ARPRV**

Slovo **ARPRV** bude nyní pracovat stejně pro jakékoli vektor čísel. Obdobně jako kompliátor vektoru bychom mohli nadefinovat i kompliátor matic. V zájmu obecnosti nadefinujeme matici jako vektor vektoru, čímž si umožníme pracovat jak s celou matici, tak s jejími jednotlivými prvky, ale také s jejími jednotlivými sloupcy jako celky (= vektory).

V matematice se obvykle používají sloupcové vektory. Pokud by někdo chtěl pracovat s celými řádky, upraví si definici po svém.

: MATECE (NOS = POČET SLOUPCŮ, TOS = POČET ŘÁDKŮ)

<BUILD>

2x 2+ C, (SLOUPEC = (POČET ŘÁDKŮ x2 + 2) BAJTŮ)

DUP C, (POČET POLOŽEK = POČET SLOUPCŮ)

DUP >R #

DO 2 C, J C, (ČÍSLO = 2 BAJTY, POČET POLOŽEK = POČET SLOUPCŮ)

J 2* ALLOT

(VYHRAZENÍ MÍSTA PRO I-TÝ ŘÁDEK)

LOOP R> DROP

(SMAZÁNÍ POČTU SLOUPCŮ ZE ZNA)

DOES > ;

Definujeme-li tedy matici

10 10 MATECE TABULKA

vypočítáme aritmetický průměr posledního sloupcu této tabulky prostřednictvím posloupnosti slov

10 TABULKA () ARPRV

Na závěr si ještě ukážeme, jak nařezení me slovo (), které na TOS uloží adresu položky, jejíž řádek předáme v NNOS a sloupec v NOS, přičemž v TOS je PFA dané matici.

: () () ;

Adresu (i,j)-tého prvku matic TABULKA nám na TOS uloží posloupnost

I J TABULKA ()

17. OPERACE VÝSTUPU

Nová slova:

BL - - (→ 32) Konstanta, která uloží na TOS ASCII kód mezery.

'EMIT - - (→ .'(EMIT).) Systémová proměnná obsahující CFA slova, které provede slovo EMIT. Platí jen pro FORTH 602.

OUT - - (→ .'(OUT).) Systémová proměnná obsahující počet vytištěných znaků na řádce. Je inkrementována slovem EMIT. V systému FORTH 602 je navíc nulována slovem CR.

<# - - (D → D) Úvodní slovo posloupnosti pro formátovaný výstup čísel. Zpracovává pouze čísla ve dvojnásobné přesnosti! (Platí pouze pro systémy fig-FORTH a FORTH 602).

- - (D1 → D2) Do výstupního řetězce zapíše nejnižší platnou číslici čísla D1. Číslo D2 je podíl (D1/(BASE)).

#> - - (D → A PZ) Zakončí převod čísla smazáním čísla D a předáním adresy výstupního řetězce a počtu znaků k tisku ve formě vhodné pro TYPE.

HLD - - (→ .'(HLD).) Systémová proměnná, obsahující během konverze čísla na znakový řetězec adresu posledního vygenerovaného znaku.

FDL - - (→ .'(FDL).) Systémová proměnná vyčleněná pro potřeby formátování.

EXECUTE - - (CFA → ???) Vykoná slovo, jehož CFA najde na TOS.

FORTH

Ing. Rudolf Pecinovský, CSc.

S-> D

- (N → D) Převede (TOS) na číslo ve dvojnásobné přesnosti.

- EXECUTE: - (→ PFA)

COMPILE: - (→)

Apostrof - zjistí PFA slova, které jej následuje ve vstupním řetězci. Je-li systém v režimu EXECUTE, uloží ji na TOS. Je-li systém v režimu COMPILE, začlení ji do definice jako číslo. Z popisu je jistě zřejmé, že slovo ' se provede i během komplikace. Blíže si o podobných slovech povíme v 18. lekcii.

CFA

- (PFA → CFA)

Uloží na TOS CFA slova, jehož PFA najde na TOS.

Slova v lekci nařezená:

EMIT - - (Z →)

Vytiskne na obrazovku (obecněji na zadáné výstupní zařízení) znak, jehož ASCII kód nalezneme na TOS.

SPACE - - (→)

Vytiskne jednu mezery.

SPACES - - (N →)

Vytiskne max (0,N) mezer. .

TYPE - - (A PZ →)

Vytiskne PZ znaků textu, který začíná na adrese A.

COUNT - - (A → A+1 PZ)

Z adresy řetězce (= adresy, na níž je uložena jeho délka) vygeneruje parametry vhodné pro TYPE.

HOLD - - (Z →)

Vloží do výstupního řetězce znak, jehož ASCII kód nalezneme na TOS. Používá se při konverzi čísel na řetězec znaků.

SIGN - - (N D → D)

V případě, že N < 0, začlení do výstupního řetězce (prevod čísel na řetězec znaků) znaménko "-". Je-li N > 0, nedělá nic.

U. - - (U →)

Vytiskne (TOS) jako číslo bez znaménka = číslo v rozsahu 0 až 65 535.

R - - (N1 N2 →)

Vytiskne číslo N1 v zóně široké N2 pozic tak, aby bylo „doraženo“ k jejímu pravému okraji.

Další slova:

. MUL MUL! *, DM.R DM. B6 CAS

Tato lekce předpokládá, že jste seznámeni s vnitřní reprezentací čísel v počítači. Pokud tomu tak není, najdete potřebné informace v kterékoli učebnici programování mikropřenosorů.

Doposud jsme si vysvětlovali, jak „rafinovaně“ lze psát v jazyce FORTH programy. Každý program ale musí umět předat společné výsledky. Pro tento účel nám doposud sloužila pouze dvě slova, a to slovo . (tečka) a slovo . (tečka-uvozovky). Pro náročnější aplikace je to málo. Ukažme si proto nyní, jak lze vlastnosti jazyka FORTH využít pro efektivní a úhledný výstup údajů.

Základním slovem, kolem nějž se točí veškerý výstup na obrazovku a jí podobná zařízení, je slovo EMIT. Toto slovo vytiskne na zadáné zařízení znak, jehož ASCII kód najde na TOS.

Zde bych chtěl udělat malou odbočku pro uživatele systému FORTH 602. Tento systém totiž umožňuje velice jednoduše přepínat rutiny tisknoucí znaky. CFA slova, které má tisknout znak, je nutno uložit do proměnné 'EMIT. Slovo EMIT je totiž v tomto systému nařezeno

: EMIT 'EMIT @ EXECUTE ;

Rutiny se přepínají velice jednoduše. Chceme-li např., aby slovo EMIT provádělo námi nařezené slovo xxx, napišeme

' xxx CFA 'EMIT !

a od této chvíle probíhají všechny tisky prostřednictvím tohoto námi nařezeného slova.

Přepínáním rutin můžeme dosáhnout toho, že systém bude tisknout na tiskárnu nebo zároveň na tiskárnu i obrazovku, na některých počítačích (např. PMD 85) může začít používat větší nebo naopak menší znaky a řadu dalších užitečných maličkostí.

Zkusme si nyní nařezenovat dvě užitečná slova, a to slovo SPACE, které na obrazovku (obecněji na zadáné výstupní zařízení) vytiskne jednu mezeru a slovo SPACES, které vytiskne (TOS) mezer, avšak pouze v případě, že (TOS) >= 0. Dříve, než se podíváme na jedno z možných řešení, zkuste si tato slova nařezenovat sami.

: SPACE BL EMIT ;

: SPACES 0 MAX ?DUP IF 0

DO SPACE LOOP ENDIF ;

Slovo EMIT vytiskne pouze jeden znak. Chceme-li vytisknout více znaků, pomůžeme si cyklem. Nařezenoveme si slovo TYPE, které vytiskne (TOS) znaků z oblasti paměti začínající na adresu (NOS).

: TYPE (NOS = ADRESA POČÁTKU TEXTU)

(TOS = POČET ZNAKŮ K TISKU)

?DUP (PŘEDPOKLÁDÁME, že TOS >= 0)

IF OVER + SWAP

(PŘÍPRAVA PARAMETRŮ PRO CYKLUS)

DO I C@ EMIT LOOP

(VLASTNÍ TISK TEXTU)

ELSE DROP ENDIF

(SMAŽ ADRESU V PŘÍPADĚ TOS = 0)

;

Při tisku textů pomocí slova TYPE musíme vědět, kolik chceme tisknout znaků. Proto se v jazyku FORTH uchovávají texty podobně, jako jsme v 16. lekcii uchovávali vektory – počet znaků textu většinou v paměti přímo předchází vlastní text. Abychom ze znalosti adresy počátku takto uloženého textu mohli poskytnout parametry pro TYPE, nařezenoveme si slovo COUNT

: COUNT DUP 1+ SWAP C@ ;

Máme-li v paměti standardně uložený text, posloupnost COUNT TYPE nám ho vytiskne na obrazovku.

Zvláštní oblastí tisku je tisk čísel. Slova, která jsou v první části slovníku v úvodu této lekce, bychom si sice mohli nařezenovat také sami, ale jsou v některých bodech poněkud komplikovanější a vyžadují znalost některých systémových slov a proto je raději budeme používat daná.

Slova realizující výstup čísel se systému od systému poněkud liší. Budu proto vysvětlovat slova v podobě, jak jsou definována v systému FORTH 602, dodávaném 602. ZO Svatoplukem v Praze 6, který je u nás v republice v profesionální sféře verzí zdaleka nejrozšířenější. Prakticky shodně definuje tyto operace i fig-FORTH rozšířený u nás zejména mezi uživateli osobních počítačů Sinclair ZX-81 a Sinclair ZX-Spectrum (mimořádě můžete si jej přijít zdarma nahrát na schůzky

(11)

Slova v lekci nadefinovaná:

- [- (→) Slovo typu IMMEDIATE. Přepne systém z režimu COMPILE do režimu EXECUTE.
-] - (→) Přepne systém z režimu EXECUTE do režimu COMPILE.
- LIT - (→ X) Uloží na TOS položku, která následuje za jeho voláním, tedy na adresu, jež by byla za normálních okolností jeho návratovou adresou. Program pak pokračuje až od následující položky.
- COMPILE - (→) Až se bude provádět slovo, které je právě definováno, začlení CFA slova, jež nyní následuje ve vstupním řetězci, na konec slovníku – tedy do vytvářené definice.
- ; - (→) Popis činnosti viz lekce č. 5.
- LITERAL -
 - EXEC.: (X → X)
 - COMP.: (X →) Slovo typu IMMEDIATE. Pokud je systém v režimu COMPILE, začlení (TOS) do definice. Pokud je systém v režimu EXECUTE, neudělá nic.
- " - (→ Z) Slovo typu IMMEDIATE. Přečte slovo, které je následuje ve vstupním řetězci a uloží na TOS ASCII kód prvního znaku.

Tato lekce je určena těm, kteří se o FORTH zajímají poněkud více, než pouze informativně. Ostatní ji mohou s klidným svědomím přeskočit.

Ríkali jsme si, že FORTH pracuje ve dvou základních režimech; v režimu EXECUTE a v režimu COMPILE a vysvětlovali jsme si, v čem se tyto režimy liší. Existuje však množina slov, která se témito pravidlům vymyká. Její nejdůležitější podmožinou jsou tzv. slova typu IMMEDIATE. Tato slova se, na rozdíl od slov standardních, provedou i když se systém nachází v režimu COMPILE. Dopsoud jsme z této množiny poznali pouze slovo ' (apostrof). Zároveň do této množiny patří slova, realizující programové konstrukce.

Dříve, než přistoupíme k podrobnějšímu výkladu slov z této skupiny, měli bychom si nadefinovat dvě základní slova – slovo [, které přepne systém z režimu COMPILE do režimu EXECUTE a slovo], které naopak přepne systém z režimu EXECUTE do režimu COMPILE. Jednou z možnosti jsou definice

: [0 STATE ! ; IMMEDIATE
HEX

:] C0 STATE ! ;

Jak jste jistě postřehli, slovo [jsme ihned po nadefinování zařadili mezi slova typu IMMEDIATE. Proč? Nachází-li se systém v režimu COMPILE, slova se neprovádějí, ale pouze začleňují do slovníku. My však potřebujeme, aby se slovo [provedlo, protože jinak bychom se z tohoto režimu neuměli dostat (slovo ; opustí režim COMPILE právě prostřednictvím slova]). Proto je zařadíme mezi slova typu IMMEDIATE, která se provedou i v režimu COMPILE.

Na rozdíl od slova [potřebujeme slovo] vykonat v režimu EXECUTE a proto není důvod, proč by se mělo od běžných slov jazyka FORTH odlišovat.

Dříve, než si ukážeme definice dalších slov jazyka FORTH typu IMMEDIATE, nadefinujeme si slova

: LIT R> DUP 2+ >R @ ;
RA RA RA RA RA+2 (RA) RA+2
RA RA+2
: COMPILE R> DUP 2+ >R @ ;

FORTH

Ing. Rudolf Pecinovský, CSc.

Zde bych chtěl upozornit na jednu věc: přestože se definice slova COMPILE velice podobá definici slova LIT, nelze napsat :

COMPILE LIT , ;
Proč? Vzpomeňte si na naše problémy se slovy I, J, R, ap. Na návratové adrese slova LIT v poslední definici již není uloženo číslo, které máme přečíst, ale CFA slova , .

Tomu, komu to ještě není zcela jasné, doporučuji namalovat si poměry na obou zásobnících a ve slovníku na papír.

Nyní jsou již naše znalosti dostatečné k tomu, abychom byli schopni nadefinovat slovo ; . Definice může být např. následující:

: ; SP@ CSP @ =
(KONTROLA SHODNOSTI ZÁSOBNÍKU SE STAVEM PŘED POČÁTKEM DEFINICE)

IF COMPILE EXIT
(ZAČLENĚNÍ SLOVA EXIT NA KONEC DEFINICE)

SMUDGE (ZVIDITELNĚNÍ SLOVA PRO SYSTÉM)

[COMPILE] [(NASTAVENÍ REŽIMU EXECUTE)

ELSE ABORT ENDIF

(POČET PRVKŮ NA ZÁSOBNÍKU NESOUHLASIL – CHYBA)

; Zkusme si nyní nadefinovat další slovo z naší množiny, slovo LITERAL:

: LITERAL STATE @
(REŽIM=COMPILE ?)
IF COMPILE LIT , ENDIF

(ANO – PROVEĎ ČINNOST)

; IMMEDIATE

Pokud chceme slovo LITERAL a s ním i jakékoli slovo IMMEDIATE použít v definici, dostáváme se do problému. Slovo se totiž i v kompilačním režimu ihned provede a nelze je proto do žádné definice začlenit. Proto se zavedlo slovo [COMPILE], které do definice začlení následující slovo, i když je typu IMMEDIATE (viz definice slova ;).

19. VSTUP ÚDAJŮ BĚHEM PROGRAMU

Nová slova:

?TERMINAL - (→ Z)

Testuje klávesnice. FORTH 602: uloží na TOS kód ASCII stisknuté klávesy.

Není-li stisknuta žádná klávesa, uloží na TOS 0. fig-FORTH: byla-li stisknuta klávesa BREAK, uloží na TOS „TRUE“ (=1), nebyla-li tato klávesa stisknuta, uloží „FALSE“.

'TERM - (→ .('TERM).)

Systémová proměnná, obsahující CFA slova, které provede slovo ?TERMINAL. Platí jen pro FORTH 602.

KEY - (→ Z)

Ceká na stisknutí klávesy a uloží kód ASCII odpovídajícího znaku na TOS.

'KEY - (→ .('KEY).)

Systémová proměnná obsahující CFA slova, které provede slovo KEY. Platí jen pro FORTH 602.

EXPECT - (A N →)

Přečte z klávesnice N znaků a jejich kódy ASCII ukládá do paměti

od adresy A.

- (→ .(TIB).)

Proměnná obsahující adresu vstupní vyrovnávací paměti klávesnice (vstupního bufferu).

IN

- (→ .(IN).)

Proměnná obsahující počet znaků přečtených ze vstupního bufferu. Je nulována slovem QUERY a inkrementována slovem WORD.

HL

- (→ Z)

Konstanta, která uloží na TOS kód ASCII mezery.

WORD

- (Z →)

Přečte ze vstupního bufferu text ukončený znakem Z a uloží jej na konec „fyzického“ slovníku, přičemž do prvního bajtu uloží počet znaků přečteného textu (viz ukládání textu popsáne v 17. lekci) a na konec přidá nejméně dvě mezery. Znaky Z, vyskytující se před vlastním textem, ignoruje. Po přečtení textu inkrementuje proměnnou IN.

AGAIN

- (→)

Ukončuje programovou konstrukci: xxx BEGIN ... AGAIN ; realizující nekonečnou smyčku. Tuto smyčku nelze opustit jinak, než vyskočit ze slova xxx pomocí EXIT nebo nějakého jeho ekvivalentu.

>BINARY

- (D1 A1 → D2 A2)

Převede text na adresu adr1 na číslo, které připočte k číslu D1* (BASE) a výsledek D2 uloží na NOS. Na TOS pak zanechá adresu prvního nepřevoditelného znaku. Počet převedených čísel připočte k proměnné DPL. Ve fig-FORTH se toto slovo jmenuje (NUMBER).

DPL

- (→ .(DPL).)

Proměnná obsahující během konverze textu na číslo počet čísel vpravo od posledního výskytu desetinné tečky nebo čárky. Nastavuje se slovy >BINARY a NUMBER.

INTERPRET - (→)

Vnější interpret, který zpracovává text ve vstupní vyrovnávací paměti. Je-li systém v režimu EXECUTE každé slovo ihned provede, je-li v režimu COMPILE, začlení slovo (přesněji řečeno jeho CFA) do slovníku. Nenajde-li slovo ve slovníku, „pošle na něj“ slovo NUMBER. Pokud se slovo podaří interpretovat jako číslo, uloží je na TOS, popř. TOS.NOS (v režimu EXECUTE), popř. je zařadí do slovníku jako literál, tj. pomocí slova LIT (v režimu COMPILE). Číslo je vyjádřeno v jednoduché přesnosti, nevyškytne-li se v něm žádná desetinná tečka a je-li tedy (DPL)=1 (viz definice slova NUMBER).

V opačném případě je pochopeno jako číslo ve dvojnásobné přesnosti.

NUMBER

- (A → D)

Převede textový řetězec na adresu A+1 (očekává, že na adresu A je uložena délka řetězce a tento bajt ignoruje) na číslo dvojnásobné přesnosti s přihlédnutím k aktuální bázi. Pokud při transformaci narazí na desetinnou

(13)

A/1

Amatérské RÁDI

23

FORTH

nou tečku nebo čárku, uloží počet číslic vpravo od jejího posledního výskytu do proměnné DPL. Např. řetězce „10,3.“, „103.“ i „103“ převede na číslo 103, které uloží na TOS.NOS (číslo je ve dvojnásobné přesnosti) ale v prvním případě uloží do proměnné DPL číslo 1, ve druhém 0 a ve třetím -1 (žádná desetinná tečka).

NUMBER - (→ .(NUMBER).)
Systémová proměnná obsahující CFA slova NUMBER, které používá **INTERPRET**. Platí jen pro FORTH 602.

Slova v lekci nadefinovaná:

QUERY - (→)
Cte znaky z klávesnice až po **RETURN** (NEW LINE, ENTRY, EOL, ...), nejvýše však 80 znaků, a uloží je do vstupního bufferu (TIB). Vynuluje proměnnou IN.

INPUT - (→ n)
Přečte z klávesnice číslo a uloží je na TOS.

COMMAND - (→ ??)
Přečte jeden řádek z klávesnice a interpretuje jej.

NUMBER - (A → D)
Zobecnění tohoto slova. Interpretuje text na adresu A jako číslo, které smí obsahovat i znaky , : / -. Tyto znaky smí číslo obsahovat v systému FORTH 602.

Další slova:
ODHAD DOBY TEXTCON <M M< MEZI? NOP

Veškeré operace realizující vstup údajů z klávesnice jsou naprogramovány pomocí slov ?TERMINAL a KEY. Slovo ?TERMINAL můžeme použít např. pro generování příznaku ukončení nekonečné smyčky, jako např. v následujícím slově:

: **ODHAD DOBY**
." PO STISKNUTÍ TLAČÍTKA POČKEJ CHVÍLI," CR
." PAK JEJ STISKNI PODRUHÉ" CR
." A NA DVAKRÁT TAK DLOUHOU DOBU POTŘETÍ." CR CR
." NA ZÁVER SE DOZVÍS PROCENTUÁLNÍ ODCHYLUKU SVÝCH ODHADŮ." CR CR CR
."STISKNI TLAČÍTKO POPRVE"
BEGIN ?TERMINAL UNTIL
." – PODRUHÉ"
32 767 0 DO ?TERMINAL IF ! LEAVE ENDIF LOOP
." – POTŘETÍ"
32 767 0 DO ?TERMINAL IF ! LEAVE
ENDIF LOOP OVER 2* – 100 LROT /*
CR CR CR ." TVŮJ ODHAD DVOJNÁSOBKU DOBY SE LIŠÍL"
." ZHRUBA 0" . . %"
;

Slovo **KEY** na rozdíl od slova ?TERMINAL přeruší provádění programu do doby, než bude stisknutej nějaké tlačítko. Pomoci slova **KEY** je nadefinováno slovo **EXPECT**, kterého pak využívá slovo **QUERY**:

DECIMAL : QUERY TIB @ 80 EXPECT 0 IN ! ;

Slovo **QUERY** načte jeden řádek do vstupního bufferu, z něhož si jej pak vnější interpret postupně dešifruje pomocí slova **WORD**.

Aby byl následující výklad pochopitelnější, podíváme se napřed trochu hlouběji do systému. Následující řádky platí však pouze pro systémy fig-FORTH a FORTH 602.

Páteří celého systému je slovo **QUIT**, které se spustí po zapnutí ihned po inicializaci systému. Toto slovo je naprogramováno jako nekonečná smyčka. Jeho zestrojená definice vypadá asi takto:

: **QUIT**
STNDIO (PLATÍ POUZE PRO FORTH 602.
NASTAVÍ PROMĚNNÉ EMIT, KEY A TERM
NA STANDARDNÍ RUTINY)
BEGIN (ZAČÁTEK NEKONEČNÉ SMYČKY)
RP! (INICIALIZUJE ZNA)
QUERY (NAČTE RÁDKU DO VSTUPNÍHO
BUFFERU KLÁVESNICE)
INTERPRET (INTERPRETUJE RÁDKU
V BUFFERU)
REZIM EXECUTE? IF " OK" ENDIF
(RÁDEK USPĚŠNĚ INTERPRETOVÁN)
AGAIN

; Slovo **INTERPRET** je definováno přibližně takto:

: **INTERPRET**
BEGIN BL WORD
(VYHLEDEJ DALŠÍ SLOVO VE VSTUPNÍM
BUFFERU)
Nalezení toto slovo ve slovníku
Našel?
IF Režim EXECUTE?
IF Proved' je
ELSE Je typu IMMEDIATE?
IF Proved' je
ELSE Zakompliluje je ENDIF
ENDIF
ELSE HERE 'NUMBER @
EXECUTE
(V SYSTÉMU FIG-FORTH JE MÍSTO
POSLOUPNOSTI)
('NUMBER @ EXECUTE
JENOM NUMBER)
Režim EXECUTE?
IF Ulož číslo na zásobník
ELSE Začleň je do definice
ENDIF
ENDIF
AGAIN

; V této definici není ošetřena možnost jednoduché a dvojnásobné přesnosti čísel. Předpokládám, že to pro vás již nebude problém.

Mnozí z vás se budou ptát, jak se systém dostane z této nekonečné smyčky, kterou, na rozdíl od smyčky ve slově **QUIT**, opustit musí. Pomůže k tomu slovo, jež je obdobou slova **EXIT** a které je na konec přečteného textu automaticky zařazeno slovem **EXPECT**. Slovo **QUERY** pak tento text předá slovu **INTERPRET** ke zpracování (viz definice slova **QUIT**).

S našimi současnými znalostmi bychom již měli být schopni nadefinovat textovou konstantu, která svoji počáteční hodnotu přečte ve fázi definice průměr z klávesnice.

: **TEXTCON**
<BUILD 0 WORD
(PRESUNE TEXT Z TIB NA KONEC SLOVNÍKU)
HERE C@ 1+ ALLOT
(A ZAČLENÍ JEJ DO DEFINICE)
DOES> COUNT TYPE

; Definiční slovo **TEXTCON** po vytvoření hlavičky zavolá slovo **WORD**, přičemž označovacím je znak z ASCII kódem 0. Slovo **WORD** tedy přečte obsah vstupního bufferu až do konce a presune tento text za slovník, tedy do téla právě definovaného slova. Překladač **TEXTCON** můžeme použít např. následovně:

TEXTCON M< NAD PRIPUSTNOU MEZ
TEXTCON <M POD PRIPUSTNOU MEZ

Takto nadefinované konstanty vždy, když budou vyvolány, vytisknou svoji hodnotu = = přířazený text. Např. slovo

: **MEZ? DUP 0< IF DROP M<**
ELSE 0> IF <M ENDIF ENDIF

; vytiskne jeden z textů v závislosti na znaménku (TOS).

Při definování slov potřebujeme velmi často zjišťovat stisknuté tlačítko nebo naopak tisknout nějaký znak. V obou případech potřebujeme do programu začlenit ASCII kód dotedněho znaku. Jelikož si však většina z nás tabulkou ASCII nepamatuje a ani ji často nemá při ruce (od toho je počítáč!), musíme si pomoci jinak. Jednou z možností je

... [KEY] LITERAL ...

Co jsme provedli? Ve chvíli, kdy jsme potřebovali začlenit do definice kód požadovaného znaku, přepnuli jsme systém do režimu EXECUTE, znak jsme přečetli z klávesnice a **KEY** nám jeho kód uložilo na TOS. Poté jsme se přepnuli zpět do režimu COMPILE a pomocí slova **LITERAL** začlenili tento kód do definice.

Můžeme však nastoupit jinou cestu. Nadefinujeme si slovo

: " **BL WORD**
(PŘEČTI NÁSLEDUJÍCÍ SLOVO ZE
VSTUPNÍHO ŘETĚZCE)
HERE 1+ C@
(TOS = KÓD PRVNÍHO ZNAKU)
[COMPILE] LITERAL
(POKUD BUDE SLOVO POUŽITO V DEFINICI,
ZAČLENÍ DO NÍ TENTO KÓD. V OPAČNÉM
PŘÍPADĚ JE PONECHÁ NA TOS.)
IMMEDIATE

Slovo tisknoucí znaménko položky na TOS pak můžeme nadefinovat např. následovně:

: **ZNAM 0< IF " – ELSE " +**
ENDIF EMIT ;

Čtení a zpracování textu je jistě velice zajímavé a mohli bychom se mu věnovat daleko podrobněji, avšak přeci jenom nejčastěji potřebujeme číst čísla; proto bude zbytek kapitoly věnován jim.

Základní slova, pomocí nichž se definuje prakticky jakýkoliv vstup, jsou slova >**BINARY** a **NUMBER**.

Nadefinujeme si nyní slovo **INPUT**, které přečte z klávesnice číslo a jeho hodnotu uloží na TOS. Předpokládejme, že číslo bude v jednoduché přesnosti.

: **INPUT QUERY BL WORD HERE
NUMBER DROP** ;

Pokud bychom chtěli obecnější vstup, můžeme si nadefinovat slovo **COMMAND**:

: **COMMAND QUERY INTERPRET** ;

ktéře přečte řádek z klávesnice a provede jej, tj. „pošle na něj“ vnější interpret. Pomoci tohoto slova můžeme uprostřed programu změnit obsah zásobníku nebo i nadefinovat nové slovo. Toto slovo můžeme vkládat do definic do míst, v nichž bychom rádi umístili kontrolní tisky nebo jinak testovali správnost programu. Když je již program odláden, nemusíme jej ani přepisovat, stačí, když v jeho definici přepíšeme CFA slova **COMMAND** na CFA slova **NOOP**, které můžeme nadefinovat jednoduše : **NOOP** ;

Na závěr této lekce si ukážeme, jak bychom mohli nadefinovat zobecněné slovo **NUMBER**. Toto slovo přijme jako přípustný znak nejen desetinnou tečku a čárku, ale i další znaky, které se v různých formátech vyskytují – budou to znaky / : a -. Taktto definované **NUMBER** pak umí přečíst čísla (32; 32.2; 32,2), čas (13:28:12), datum (17/07/84) nebo telefonní číslo (332-3880) ve tvaru, který se běžně používá.

HEX

```

: NUMBER      ( TOS = ADRESA S PZ
    PŘEVÁDĚNÉHO TEXTU )
0 0 ROT      ( PŘIPRAVA PARAMETRU
    PRO SLOVO BINARY )
DUP 1+ C@ 2D =
    ( JE PRVNÍM ZNAKEM
    MINUS? )
DUP >R      ( PŘÍZNAK ZNAMÉNKA >R )
-          ( POKUD BYLO ČISLO
    ZÁPORNÉ, POSUŇ ADRESU
    0+1 )
    ( POZOR! FIG-FORTH DÁVA PŘI TRUE +1,
    TAKŽE MUSÍME MINUS ZMĚNIT NA PLUS )
-1          ( PŘÍZNAK
    ŠESTNÁCTIBITOVÉHO
    ČÍSLA = POČET MÍST VPRAVO
    OD POSLEDNÍHO VÝSKYTU
    NENUMERICKÉHO ZNAKU )
BEGIN       DPL !
    ( ULOŽ POČET MÍST
    VPRAVO OD POSLEDNÍHO
    VÝSKYTU NENUMERICKÉHO
    ZNAKU )
>BINARY     ( PREVED DALŠÍ ČÁST
    ČÍSLA )
DUP C@ BL -
    ( NENUMERICKÝ ZNAK =
    MEZERA? )
WHILE DUP C@
    ( NEJ - PODEZŘELÝ ZNAK
    JE NATOS )
3A -        ( JE TO DVOJEČKA? )
IF DUP C@ 2C <
    ( NE JE V ASCII PŘED
    ČARKOU )
SWAP 2F > OR
    ( NEBO ZA LOMÍTKEM? )
IF CR .. " NELZE PREVEST"
QUIT ENDIF
ENDIF      ( POZN.: V ASCII NÁSLEDUJÍ
    . . / ZA SEBOU )
0          ( BYL TO PŘÍPUSTNÝ ZNAK,
    VPRAVO OD NĚJ DOSUD NIC )
REPEAT
DROP      ( SMAŽ ADRESU )
R> IF DNEGATE ENDIF
    ( POKUD BYLO NA POČÁTKU
    - , PŘEVEĎ )
;
```

Často se stane, že uživateli rutiny dodávané se systémem nevyhovují a že by si rád nadefinoval rutiny vlastní. Uživatelé systému FORTH 602 mohou systému nařídit, aby místo standardních rutin používal jejich vyplňené (např. potřebují čist čísla v plovoucí čárce). Slovo **INTERPRET** totiž používá pro převod textového řetězce na číslo to slovo, jehož CFA je hodnotou proměnné **NUMBER**. Předdefinování je jednoduché:

```

' NUMBER      ( UMÍSTÍ NA TOS PFA
    NOVÉHO NUMBER )
CFA 'NUMBER !
    ( ULOŽ CFA TOHOTO SLOVA
    JAKO HODNOTU PROMĚNNÉ
    'NUMBER )

```

Obdobně lze předdefinovat i rutiny, na něž ukazují proměnné **'EMIT**, **'KEY**, **'TERM**.

20. LOGICKÉ SLOVNÍKY

Nová slova:

VOCABULARY - (→)

Definiční slovo – definuje nový logický slovník. Při použití takto definovaného slova se na tento slovník nastaví proměnná **CONTEXT**. Bývá zvykem definovat slovníky jako slova typu **IMMEDIATE**.

FORTH

Ing. Rudolf Pecinovský, CSc.

FORTH	- (→)	Základní logický slovník – slovo typu IMMEDIATE .
CONTEXT	- (→) (CONTEXT)	Proměnná, obsahující adresu logického slovníku, v němž INTERPRET hledá nejdříve.
CURRENT	- (→) (CURRENT)	Proměnná, obsahující adresu logického slovníku, do nějž se ukládají nově definovaná slova.
DEFINITIONS	- (→)	Nastaví proměnnou CURRENT na slovník, na nějž ukazuje proměnná CONTEXT .

V této leci se seznámíme s jednou zvláštností jazyka FORTH, kterou jsou logické slovníky.

Kdykoli jsme doposud hovořili o slovníku, měli jsme na mysli tzv. fyzický slovník, neboli místo v paměti, kam se ukládají definice nových slov. V této leci hovoříme o tzv. logických slovnících, což jsou datové struktury typu strom. Každé slovo patří do některého logického slovníku a SA v jeho hlavičce od- kazuje na předchozí slovo z téhož logického slovníku.

V každou chvíli jsou aktuální nejvýše dva logické slovníky. Jednak je to slovník, do nějž se ukládají nové definice, jednak slovník, v němž se začíná s hledáním slov. Na první slovník ukazuje proměnná **CURRENT**, na druhý slovník proměnná **CONTEXT**. Každé slovo se nejdříve hledá ve slovníku, na nějž ukazuje proměnná **CONTEXT** (tentotéž slovník budeme označovat (CONTEXT)), a v případě, že se nenaleze, prohledává se ještě slovník, na nějž ukazuje proměnná **CURRENT** (slovník (CURRENT)), tedy slovník, do nějž zařazujeme nové definice. Teprve když se slovo nepodaří najít ani v jednom ze slovníků, pokusí se **INTERPRET** interpretovat je jako číslo.

Zavedení logických slovníků přináší několik výhod. Za prvé vnitřní interpret prohledává menší množinu slov a zpracování vstupního řádku je proto rychlejší, za druhé v různých logických slovnících mohou být slova stejně pojmenovaná, aníž by slovo později definované zakrylo slovo starší.

Nové logické slovníky definujeme pomocí překladače **VOCABULARY**. Zde bych chtěl připomenout, že logický slovník je slovo jako každé jiné a patří proto do některého logického slovníku. Základním logickým slovníkem je slovník **FORTH**, který patří sám do sebe. Na něj se navazují všechny další slovníky.

Pro používání slovníku jsou důležité dvě věci: Jak víme, při provedení slova které je slovníkem, se provede výkonná část slova **VOCABULARY**, která zařídí, aby se na daný slovník nastavila proměnná **CONTEXT**. Druhou skutečností, kterou musíme mit na paměti, je, že slovníky bývá zvykem definovat jako slova typu **IMMEDIATE**, tzn. jako slova, která se provedou i uprostřed definice. Kdykoli tedy provedeme slovo, které je slovníkem, provede se výkonná část slova **VOCABULARY**, která zařídí, že od této chvíle začíná **INTERPRET** prohledáváním právě tohoto slovníku.

A nyní opět pozor! Také slovo : (dvoječka) nastavuje proměnnou **CONTEXT** a to na slovník (**CURRENT**), neboli dvoječka zařídí, že použitá slova se hledají v tomtéž slovníku, do nějž se ukládá právě definované slovo.

Nyní již tedy víme, jak lze nastavit slovník, v němž záčneme s hledáním použitých slov. Zbývá nám ještě naučit se jak změnit slovník, do nějž budeme ukládat nové definice. K tomu slouží slovo **DEFINITIONS**, které nastaví proměnnou **CURRENT** na slovník (**CONTEXT**).

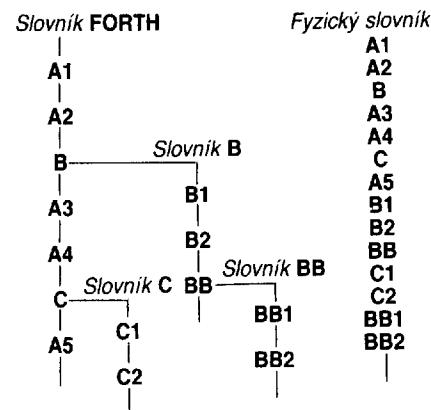
Používání slovníku se pokusím demonstrovat na následujícím příkladu:

```

( JE NASTAVEN LOGICKÝ SLOVNÍK FORTH )
: A1 ; : A2 ;
VOCABULARY B IMMEDIATE
: A3 ; : A4 ;
VOCABULARY C IMMEDIATE
: A5 ;
B DEFINITIONS : B1 ; : B2 ;
;
VOCABULARY BB IMMEDIATE
BB DEFINITIONS : BB1 ;
C DEFINITIONS
: C1 B B1 B2 A1 A2 ;
: C2 B BB BB1 A5 ;
BB DEFINITIONS
: BB2 FORTH C C2 ;

```

Ukážeme si nejdříve na obrázku, jak budou navzájem navázána jednotlivá slova a jak budou tato slova uložena ve fyzickém slovníku.



Jak je vidět z příkladu, logické slovníky tvoří stromovou strukturu. Ještě jednou připomenu, že spojovací adresa každého slova ukazuje na počátek hlavičky předchozího slova v témže logickém slovníku. Proto jsme při definici slova **C1** museli přepnout proměnnou **CONTEXT** nejprve na slovník **B**, protože jinak by počítač slova **B1** a **B2** nenalezl, pretože byla definována dříve než slovo **C1**.

Dvoječka v definici slova **C2** přepne **CONTEXT** zpět na slovník **C** a proto nemůže **INTERPRET** slovo **A5** najít, takže nepracuje-li právě v číselné soustavě se základem většinu než deset (pak by bylo možné **A5** interpretovat jako číslo), ohláší chybu.

Ve slově **C2** bychom si měli všimnout ještě jedné zvláštnosti. Při definici tohoto slova jsme potřebovali slovo ze slovníku **BB**, při (**CURRENT**) = (**CONTEXT**) = **C** je nám slovník **BB** nepřístupný. Proto jsme museli přepnout **CONTEXT** nejdříve na slovník **B** a pak teprve na slovník **BB**. Obdobně jsme si museli počítat, když jsme v definici slova **BB2** chтиeli použít slovo **C2**.

Pokud to nebylo z dosavadního výkladu dostatečně zřejmé, chtěl bych upozornit, že při práci s jakýmkoli logickým slovníkem jsou nám přístupná kromě slov z tohoto slovníku

► i všechna slova ze slovníku, v němž byl náslovník nadefinován, pokud byla vytvořena před definicí tohoto slovníku. Pokusím se tuto složitou větu vysvětlit konkrétněji. Pokud CURRENT nebo CONTEXT ukazují na slovník FORTH, mohu použít slova A5, C, A4, A3, B, A2, A1 a všechna slova ze slovníku FORTH, definovaná před slovem A2. Pokud ukazují na slovník C, mohu použít slova C2, C1, C a všechna slova ze slovníku FORTH, definovaná před slovem C. Ukazuje-li některá z výše jmenovaných proměnných na slovník BB, mohu použít slova BB2, BB1 a BB, dále slova ze slovníku B definovaná před slovem BB, tedy B2, B1 a B a nakonec i slova ze slovníku FORTH definovaná před slovem B, tedy slova A2, A1 a další, která byla nadefinována ještě dříve.

Na závěr této lekce bych vás chtěl „uklidnit“; logické slovníky se většinou nepoužívají tak divoce, jako v našem příkladě. Ten vám měl jenom usnadnit pochopení, na co všechno je třeba při práci s logickými slovníky dávat pozor. V běžné praxi slouží logické slovníky většinou k tomu, abychom ve fyzickém slovníku oddělili slova, která patří k různým projektům. Tím se zrychlí fáze komplikace, protože počítac nemusí prohledávat celý fyzický slovník, ale pouze logický slovník související s daným projektem. Standardní profesionální verze jazyka FORTH mívaly většinou tři logické slovníky – FORTH, ASSEMBLER a EDITOR. První obsahuje všechna slova, která jsme se doposud naučili a případně i některá další, druhý umožňuje práci v jazyku symbolických adres, o níž bude pojednávat příští lekce, a třetí poskytuje prostředky pro práci se zdrojovými texty, uloženými na disku v tzv. skrinách.

21. JAZYK SYMBOLICKÝCH ADRES (JSA)

Nová slova:

ASSEMBLER	- (→)	Logický slovník, v němž jsou uložena slova, umožňující programování JSA. Jsou typu IMMEDIATE!
CREATE	- (→)	Přečeze ze vstupního bufferu název nového slova a vytvoří jeho hlavičku s tím, že (CFA) = PFA. POZOR! Slovo je po vytvoření hlavičky ještě stále pro systém neviditelné!
NEXT	- (→ (NEXT))	Konstanta obsahující adresu počátku vnitřního interpretu.
DP	- (→ .(DP).)	Proměnná obsahující adresu prvního volného bajtu ve fyzickém slovníku. Její obsah je čten slovem HERE a modifikován slovem ALLOT.
ERROR	- (N →)	Vytiskne zprávu o chybě číslo N. Inicializuje UZ a ZNA.
;CODE	- (→)	Má podobný význam jako DOES> s tím rozdílem, že výkonná část překladače může být psána přímo ve strojovém kódu, popř. v JSA.
(;CODE)	- (→)	Výkonná část slova ;CODE, která je začleněna do definice překladače, aby při jeho provádění ukončila fázi komplikace

(16)

FORTH

Ing. Rudolf Pecinovský, CSc.

va „spustit“, aby po jejich vykonání pokračoval v plném slova, která tento slova vyvolalo.

Jak jsme si již řekli, do definice slova se ukládají CFA slov, která se mají vykonat. Vnitřní interpret tedy čte tyto CFA a provede skok na adresu, která je na CFA uložena. Předpokládám, že si ještě pamatujete, že na této adrese je uložena adresa počátku výkonné části překladače. Pokud slovo není naprogramováno v jazyku FORTH, ale v JSA, ukazuje jeho CFA nejčastěji na jeho PFA, kde začíná vlastní program.

Z tohoto budeme vycházet při definování slova CODE, které bude zastupovat dvojtečku v definicích slov programovaných v JSA. Slovo CODE musí provést dvě věci: utvořit hlavičku nově definovaného slova a přepnout CONTEXT na ASSEMBLER, abychom mohli používat slova v tomto logickém slovníku nadefinovaná. Definice slova CODE může být např. následující:

CODE CREATE (COMPILE) ASSEMBLER ;

V této definici bych chtěl upozornit na dvě zajímavé věci. První je slovo CREATE, které pracuje obdobně jako slovo <BUILDS v konstrukci <BUILDS ... DOES> (některé verze jazyka FORTH používají slovo CREATE místo slova <BUILDS, v některých, např. FORTH 602, lze v této konstrukci použít slova obě). Druhou věcí, která stojí za povšimnutí, je způsob začlenění slova ASSEMBLER do definice slova CODE. V minulé lekci jsme si říkali, že slovníky se definují jako slova typu IMMEDIATE a že se proto tato slova vykonají i během definice. Pokud nechceme, aby se slovo vykonalo, ale naopak je chceme začlenit do definice, musíme tak učinit explícitně prostřednictvím slova [COMPILE]. Slovo se pak vykoná až během provádění slova, do jehož definice bylo začleněno.

Po vykonání každého slova se musí program vrátit do vnitřního interpretu. Protože tento návrat bude ve všech slovech stejný, můžeme si nadefinovat slovo ;C, které jej připraví.

ASSEMBLER

: ;C NEXT JMP, CURRENT @ CONTEXT ! SMUDGE ;

Základním pravidlem, kterým se musíme řídit v JSA stejně jako v celém jazyku FORTH, je postfixová notace. Znamená to, že i instrukce JSA musíme psát obráceně, než jsme zvykli – napřed operandy a potom operátor. Druhým nepsaným pravidlem je, že názvy instrukcí obsahují jako poslední znak čárku, která symbolizuje, že toto slovo začlení zpracovaný objekt do slovníku.

Slovo ;C kromě toho, že začlenilo do definice strojový kód instrukce skonu na počátek vnitřního interpretu, muselo ještě nastavit zpět slovník CONTEXT, který slovo CODE nastavilo na ASSEMBLER. Proměnnou CONTEXT ovšem nenastavuje přesně na původní hodnotu, ale nastavuje ji na slovník CURRENT. Na závěr pak právě nadefinované slovo „zviditelní“.

Zkusme si na ukázku nadefinovat slovo R-, které odečte (NOS) od (TOS). Slovo nadefinujeme v JSA procesoru 8080.

CODE R-
H POP, D POP, (V REGISTRU HL JE
TOS, V DE NOS)
L A MOV, E SUB, A L MOV,
(ODEČET LSB)
H A MOV, D SBB, A H MOV,
(ODEČET MSB)
H PUSH,
;C

Předpokládám, že těm z vás, kteří znají JSA 8080, nečinilo porozumění programu žádné potíže. Protože nejvíce implementaci jazyka FORTH je u nás právě na systémech s mikroprocesorem 8080 nebo blízkým Z80 popř. U880D, budeme se v dalším výkladu držet i jejich JSA.

nového slova a nastavila CFA tohoto slova na výkonnou část použitého překladače. (Platí pouze pro fig-FORTH.)

Slova v lekci nadefinovaná:

CODE - (→)
Zastává funkci dvojtečky pro slova definovaná v JSA. Nastavuje CONTEXT na slovník ASSEMBLER.

;C - (→)
Ukončuje definici slova naprogramovanou v JSA. Nastavuje CONTEXT zpět na (CURRENT).

SEGMENT - (→)
Součást slovníku ASSEMBLER. Podrobněj viz text.

ERROR - (→)
Součást slovníku ASSEMBLER. Očekává v registru HL kód chyby. Vyvolá proceduru ERROR jazyka FORTH.

SUBROUTINE - (B →) komp.
(N1 ... Np →) výk.

Očekává v TOS počet parametrů definované procedury. Výkonná část pak zařídí, aby se při naprogramování vyloučil této procedury automaticky uložily její parametry za příkaz CALL.

PROCEDURE - (→)
Překladač – NS nadefinuje jako proceduru volatelnou z JSA pomocí CALL..

: - (→)
Popis – viz nová slova v 5. lekci.

Další slova:

CSUM **CSUM-GEN** **CSUM-ceck**

Prestože je FORTH velice mocným jazykem, nemůžeme o něm zdaleka tvrdit, že umí vše. Vždy se najdou některé nové aplikace, na něž nám dosavadní slovní zásoba nestačí. Jindy slova naprogramovaná v jazyku FORTH pracují pro nás příliš pomalu a my bychom si je potřebovali nadefinovat přímo v jazyce symbolických adres (JSA) použitého procesoru. Obdobně bychom si často potřebovali nadefinovat slova, která ošetřují některé periférie či čidla. Proto profesionální verze jazyka FORTH umožňují programovat i v JSA. Překladač z JSA do strojového kódu je samozřejmě naprogramován v jazyku FORTH. To má jednu velikou výhodu: Při programování v JSA máme stále k dispozici celý aparát jazyka FORTH, který představuje prostředek takové síly, jakou nám neposkytne žádný z dostupných asemblerů ani makroasemblerů. Některé z jeho možností si ukážeme v této kapitole.

Programování v JSA vyžaduje již některé vědomosti o vnitřním interpretu a detailnější znalost jazyka. Jelikož se jednotlivé verze jazyka FORTH od sebe někdy dost podstatně liší, zaměřím se na u nás nejrozšířenější verzi jazyka, FORTH 602.

Nejdříve si vysvětlíme rozdíl mezi pojmy vnější a vnitřní interpret. Vnější interpret je řádným slovem jazyka FORTH (INTERPRET), a čte vstupní řetězec z vyrovnávací paměti, kam jsme data nahráli buď přímo z klávesnice nebo z disku, nebo z vyhrazené části paměti či jiného zdroje. Hovořili jsme doposud o interpretu, měli jsme na mysli vždy tento vnější interpret.

Vnitřní interpret naproti tomu čte z definic adresy slov, která se mají vykonat, a tato slo-

Ukážeme si nyní některé obraty, které lze při programování v JSA použít. Při programování v JSA totiž vůbec nemusíme zůstat jen v assembleru, ale jak jsem již řekl v úvodu lekce, můžeme použít celý aparát jazyka FORTH.

Představme si, že několik programů používá společnou část, která oštepuje případy, kdy nastala chyba. Ve všech těchto programech bychom tedy rádi v některém místě provedli skok na tučnou část. Abychom mohli do programu začlenit skok, musíme nejprve umístit na TOS adresu, na niž se bude skákat. S obdobným problémem bychom se mohli setkat i při volání nějaké dříve nadefinované procedury. Můžeme jej vyřešit např. tak, že si nadefinujeme překladač PROCEDURE.

ASSEMBLER DEFINITIONS

: PROCEDURE

0 VARIABLE

(VYTVOŘÍ HLAVIČKU

NOVÉHO SLOVA –

PROCEDURE)

-2 DP +! (POSUNE DP ZPĚT NA PFA
TOHOTO SLOVA)

Jak vidíte, tento překladač je nadefinován poněkud nestandardně. Rozeberme si proto jeho činnost podrobněji:

0 VARIABLE

vytvoří hlavičku nové definovaného slova s tím, že výkonná část překladače bude shodná s výkonnou částí překladače VARIABLE, a že v prvních dvou bajtech těla bude uložena nula.

-2 DP +! – posune ukazatel volné parametru ve fyzickém slovníku zpět na počátek těla slova. Při příštém ukládání do slovníku se tedy nula, zapsaná v minulém kroku, přepíše.

Vytvořili jsme tedy slovo, jehož AVCP ukazuje na výkonnou část překladače VARIABLE. Ta uloží na TOS adresu těla slova, tedy adresu, na niž chceme skákat. Na této adrese proto musí začínat program ve strojovém kódu. Proto jsme posunuli ukazatel volné parametry DP zpět na počátek těla.

POZOR! Slovo je pro systém viditelně ihned po vytvoření hlavičky, může tedy volat i samo sebe.

Nadefinujeme si nyní slovo ERROR, které lze vyvolat z programu ve strojovém kódu a které spustí slovo ERROR jazyka FORTH.

ASSEMBLER DEFINITIONS

: PROCEDURE ERROR

LHLD,

(VE FÁZI DEFINICE OČEKÁVÁ
NA TOS KÓD CHYBY, NECHÁ
TENTO KÓD ULOŽIT DO
REGISTRU HL A PŘI

VYKONÁVÁNÍ SLOVA

JEJ ULOŽÍ NA TOS)

H PUSH, FORTH, ERROR CFA

(ULOŽÍ NA TOS CFA SLOVA
ERROR JAZYKA FORTH

BĚHEM DEFINICE)

LHLD, H PUSH,

(ULOŽ TUTO CFA NA TOS
BĚHEM PROVÁDĚNÍ)

' EXECUTE (POKRAČUJ VYPLNĚNÍ SLOVA EXECUTE)

CFA

JMP

Opět si celou sekvenci rozebereme po řádcích:

- Nové slovo budeme ukládat do slovníku ASSEMBLER.
- Pomocí překladače PROCEDURE nadefinujeme hlavičku slova.
- Začlení do definice instrukci, která při provádění slova uloží hodnotu, která je nyní na TOS, do registrového páru HL.

FORTH

Ing. Rudolf Pecinovsky, CSc.

M A MOV,

(PRVNÍ SČÍTANEC DO A)

BEGIN, B DCR, NZ

(KONEC ?)

WHILE, H INX, M ADD,

(DOKUD B = 0)

REPEAT, (POKRAČUJ ZPĚT NA „BEGIN“)

(NÁVRAT)

Takto nadefinované slovo můžeme nyní použít:

CODE: CSUM-GEN

(GENERÁTOR
KONTROLNÍHO SOUČTU)

E POP, H POP,

(PŘEVZETI PARAMETRU
ZUZ)

CSUM CALL,

(REG. A = KONTROLNÍ
SOUČET)

A M MOV,

(PŘESUŇ SOUČET ZA
SČÍTANÉ BAJTY)

:C

CODE: CSUM-CHECK

(KONTROLA SPRÁVNOSTI
KONTROLNÍHO SOUČTU)

E POP, H POP,

(PŘEVZETI PARAMETRU
ZUZ)

CSUM CALL,

(A = SKUTEČNÝ SOUČET)

M CMP,

(JE SOUČET SHODNÝ
S NÁSLEDUJÍCÍM BAJTEM?)

NZ IF, 9 ERROR ENDIF,

(NENÍ → CHYBA)

:C

(VŠE V POŘÁDKU)

Předpokládám, že program je jasný. Chtěl bych jen upozornit na to, že čárka ve slovech BEGIN, REPEAT, atd. není chybou tisku, ale že má odlišit programové konstrukce, které používáme v JSA, od konstrukcí používaných v jazyku FORTH.

Nyní bychom již mohli lehce odhadnout, proč se překladač z JSA, který používáme v jazyku FORTH, říká „strukturovaný makroassembler“. Strukturován proto, že v něm můžeme programovat strukturované, a makroassembler proto, že nám umožňuje vytvářet tzv. makra, což jsou v našem případě slova, která na základě vstupních parametrů sama vygenerují patřičnou část programu (viz překladač SUBROUTINE).

ZÁVĚR

V poslední lekci našeho kursu bych vás chtěl seznámit s rozšířením jazyka FORTH v naší republice a s možností získání překladače.

Poslední a asi nejdůležitější podnět k novitěmu rozšíření jazyka FORTH byl dán v roce 1981 vydáním srpnového čísla časopisu Byte, které bylo celé věnováno tomuto do té doby teměř neznámému jazyku. Od té doby prudce vzrástlo počet implementací tohoto progresivního jazyka na všechny družiny a typech počítačů. Jednou z prvních verzí jazyka FORTH v naší republice byl BD-FORTH, naprogramovaný ing. Dědinou, CSc., v ÚTIA ČSAV pro mikroprocesor 8080 a později upravený Petrem Novákem na ČVUT FEL a implementovaný i na počítače SMEP a Sinclair ZX-81 pod názvem mini-FORTH. Tento FORTH byl donedávna v naší republice i verzi nejrozšířenější, zejména díky uživatelům mikropočítačů ZX-81.

Ve světě nejrozšířenější verzi je fig-FORTH, který vyuvinula organizace FORTH interested group. Její koncepci jazyka pře-

(17)

- Začlení do definice instrukci, která při provádění slova ERROR uloží obsah páru HL na TOS.
- Slovo je viditelné. Proto nastavíme CONTEXT na FORTH, takže první slovo, které interpret objeví, bude ERROR z logického slovníku FORTH. Jeho CFA uložíme na TOS. POZOR! Tato adresa se objeví na TOS pouze při programování slova ERROR. O to, aby se tam objevila i při vykonávání tohoto slova, se musí postarat vlastní program ve strojovém kódu, který bude vytvořen dále.
- Program v JSA. Slovo LHLD, očekává svůj argument na TOS a začlení do programu odpovídající instrukci s tímto parametrem. Při vykonávání slova ERROR se proto do registru HL uloží CFA slova ERROR jazyka FORTH a následující instrukce pak uloží obsah HL, tedy tuto CFA, na TOS.
- Uloží na TOS PFA slova EXECUTE.
- Uloží adresu počátku slova EXECUTE na TOS (slovo EXECUTE je napsáno ve strojovém kódu, proto je adresa jeho počátku uložena v jeho CFA).
- Začlení do programu skok na počátek slova EXECUTE.

Podobně můžeme „míchat“ práci v jazyku FORTH s prací v assembleru neustále. U minipočítačů se např. často předávají parametry procedurám tak, že se jejich adresy nebo hodnoty zapíší za volání procedury. I když toto řešení není u procesoru 8080 obvyklé, použijeme je jako ukázkou možností jazyka FORTH:

ASSEMBLER DEFINITIONS

: SUBROUTINE

<BUILD> C, (ZAPAMATUJE SI POČET
PARAMETRŮ)
(TĚLO NADEFINUJEME
V JSA)

DOES>

DUP 1+ CALL,
(ZAČLENÍ VOLÁNÍ SEBE DO
PROGRAMU)
C@ 0 DO , LOOP
(ZA TOTO VOLÁNÍ DODA
PARAMETRY)

Tento překladač uloží do prvního bajtu těla NS počet parametrů procedury. Při vykonávání NS začlení napřed do programu volání procedury NS. Protože v prvním bajtu těla je uložen počet parametrů a skutečný program začíná až o bajt dál, musí adresu, kterou obdrží na TOS od slova DOES> napřed upravit. Zbytek výkonné části překladače SUBROUTINE pak tvoří cyklus, který za toto volání začlení parametry, které očekává na UZ (pozor na pořadí!).

Slovo nadefinované pomoci překladače SUBROUTINE budeme používat jako jiná slova jazyka FORTH – potřebujeme-li je vykonat, napišeme pouze jeho jméno. Naproti tomu slovo nadefinované překladačem PROCEDURE musíme použít v sekvenci:

slovo CALL, nebo slovo JMP,

Programujeme-li v JSA v jazyku FORTH, míváme často k dispozici nejen instrukce JSA, ale i řadu dalších slov, která umožňují používat i pro programování v JSA programové konstrukce, na které jsme zvyklí z „čísťeho“ jazyka FORTH. Nadefinujeme si např. slovo CSUM, které seče všechny bajty ze zadaného pole a uchová v registru A spodních osm bitů tohoto součtu.

PROCEDURE: CSUM

(HL = ADRESA PRVNÍHO
BAJTU. B = POČET BAJTŮ)

vzala řada dalších firem, které ho pak prodávají pod vlastními názvy. Dalo by se říci, že fig-FORTH je ve světě neoficiálním standardem. K jeho velkému rozšíření nemálo měrou jistě přispěla i skutečnost, že se jeho tvůrci vzdali jakýchkoli licenčních nároků.

Jinou velice známou verzi jazyka je **poly-FORTH**, dodávaný společností FORTH inc., kterou založil sám tvůrce jazyka Ch. T. Moore. Poly-FORTH sice ve světě představuje špičku, ale pro řádového uživatele je již zbytečně dokonalý. Jen pro ilustraci uvedu, že v zájmu co nejrychlejšího vyhledávání slov ve slovníku používá osmí oddělených slovníků (mohli bychom je nazvat pseudofyzickými), do nichž ukládá slova v závislosti na prvním písmenu názvu. Při hledání slova pak vnitřní interpret potřebuje v ideálním případě zkонтrolovat pouze jednu osminu slov daného logického slovníku. Aby bylo hledání slov ještě rychlejší, nepamatuje si systém celé názvy slov, ale z každého slova si pamatuje pouze počet písmen a první tři písmena názvu. Celý systém obsahuje řadu dalších rafinovaností, které mohou docenit majitelé šestnáctibitových počítačů, ale pro malé systémy, které v naší republice převažují, je až nevýhodný.

Uživatelé počítačů SHARP a jím podobného systému SOS budou možná znát systém **KNITH-FORTH**. Tento systém je FORTH jenom podle názvu. Z původního jazyka zbyly jen některé vnitřní rysy. Nemá slova <BUILDS a DOES> a i jinak působí spíše amatérsky.

U nás je v současné době v profesionální sféře nejvíce rozšířován systém **FORTH 602**, distribuovaný 602. ZO Svazarmu v Praze 6. Vychází koncepcně ze systému fig-FORTH, v některých směrech jej však podstatně zdokonaluje a rozšiřuje. Klasický FORTH, tedy i fig-FORTH, je stavěn na počítače, které mají diskové jednotky a většina nediskových verzí jazyka je proto v některých směrech poněkud těžkopádná. Autori systému FORTH 602 naopak vycházejí z toho, že většina uži-

FORTH

Ing. Rudolf Pecinovsky, CSc.

vatelů v naší republice diskové jednotky nemá a proto se jej snaží vybavit některými možnostmi, které nepřítomnost diskových jednotek do jisté míry nahradí. Zároveň se ale snaží neodlísit se výrazně od standardního fig-FORTH, aby bylo možné bez problémů přebírat uveřejňované programy. FORTH lze v současné době implementovat na jakýkoli počítač s minimálně 16 kB paměti. V současné době se dodává pro počítače SAPII, PMD85, EG3003 a TNS. Existují i „diskové“ verze, které fungují pod operačními systémy CP/M, Mikros, ISIS II. a NEWDOS. Kromě základního interpretru obsahuje řadu dalších užitečných podsystémů, jako je obrazovkový editor, strukturovaný makroassembler, prostředky pro zpětné dešifrování, trasování a ladění programu, aritmetiku v plovoucí čárci, soubory grafických podprogramů, přidavné programové a datové struktury a řadu dalších užitečných doplňků. Mezi nejužitočnější vlastnosti tohoto systému patří možnost krokování laděných slov s průběžným výpisem obou zásobníků a zadaných oblastí paměti s možností vstupovat do programu po každém kroku. Pro uživatele, kteří nemají diskety, existuje podsystém, který je umožňuje simulovat v paměti RAM. Celý systém obsahuje okolo 400 slov.

Na podzim 1984 byla ve vývoji verze, která bude umět provádět několik programů současně. Kromě toho byl zahájen vývoj tzv. cílového překladače, který by měl sloužit pro vygenerování programu pro nejmenší aplikace, realizované např. pouze základní deskou JPR1. Tento překladač začlení do vysledného programu pouze těla použitých slov, tedy žádné hlavičky ani slova, která potřebuje pouze systém, nebudou zbytečně zabírat

„drahou“ paměť. Takovýto program vychází ve většině případů kratší, než odpovídající program, napsaný přímo ve strojovém kódu.

Majitelé osobních mikropočítačů Sinclair si mohou zdarma nahrát překladače FORTH (v několika verzích) na setkání Klubu uživatelů osobních počítačů každou první sobotu v měsíci v Praze 6, Pod Juliskou 2.

A rada na závěr? Programy, které vytváříte, pečlivě dokumentujte. Zejména u jazyků s hutným zápisem, ke kterým FORTH patří, se vám čas strávený nad podrobnou dokumentací určitě vrátí. Zádny program nepřetržitě dlouho ve své původní podobě. Velice záhy učítě potřebu udělat několik „nepodstatných“ změn! Zejména v případě, kdy se k programu vracíte po delší odmlce, velice snadno přehlédnete řadu důležitých detailů, jejichž pravý význam jste mezikápně zapomněli, a systém vás to vyfresta.

Nevytvářejte dlouhá slova. FORTH je velice kompaktní, takže často již druhý den ztratíte přehled o tom, jak jste problém vlastně naprogramovali. Mějte neustálé na paměti, že jedno dlouhé slovo se vymýší, programujte, testujte a ladí mnohem dlele, než několik krátkých. FORTH a moderní programování vůbec nejsou přáteli složitých programů. Moderní filozofie programování razí zasadu co nejjednodušších procedur, vykonávajících pouze jednu presně definovanou činnost.

Jako každý jazyk se i FORTH nejlépe naučíte tím, že jej budete používat. Nepouštějte se však hned z počátku do konstrukce projektů, na něž nejste teoreticky a prakticky dobře připraveni. Neznalý programátor udělá radu chyb, anž by o nich věděl a uměl je najít. A většinou chybu nechápe jako chybu vlastní, a pokládá ji za chybu systému. A systém se mu za to v nejneocenitelnějších chvílích mstí různými haváriemi.

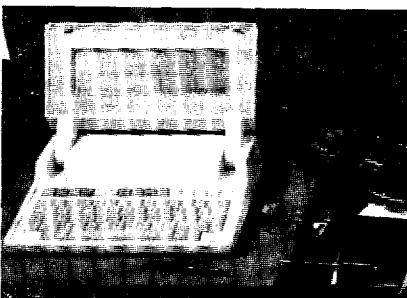
Na závěr vám přejí, aby vás vás systém „poslouchal“ a aby vám přinesl hodně zdravu ve vaší práci.

ZE SVĚTA

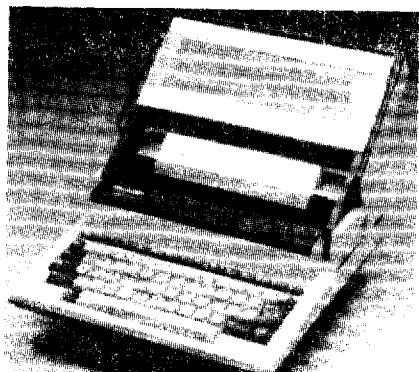
HEWLETT PACKARD 110

Mikropočítač HP 110 pracuje s mikroprocesorem 8086 (kmitočet 5,33 MHz). Má ROM 384 kB a RAM 272 kB. Výklopný displej LCD má 16 řádků po 80 znacích nebo grafiku 480 × 128 bodů. Jako periferie lze použít tiskárnu ink-jet a minifloppy jednotku 710 kB, která pracuje i na baterie. Interfejs RS-232C, HP-IL, operační systém MS-DOS. Počítač vydří pracovat na baterie 16 hodin a údaje zůstávají v paměti počítače jeden rok po jeho vypnutí.

rh



Mikropočítač HP 110



Mikropočítač SHARP PC-5000

SHARP PC-5000

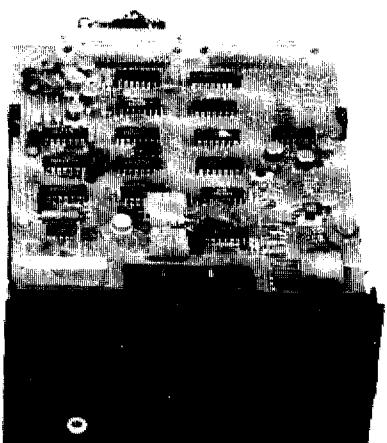
Mikropočítač používá mikroprocesor 8088, paměť ROM 192 kB, RAM 128 až 256 kB. Součástí počítače je displej LCD 8 řádek po 80 znacích nebo grafika 640 × 80 bodů. Dále obsahuje tepelnou tiskárnou 80 znaků v řádce.

K mikropočítači lze připojit magnetickou bublinovou paměť 128 kB, minifloppy dvojice 2 × 360 kB nebo kazetový magnetofon a přes RS232C další periferie. Operační systém MS-DOS, základní programovací jazyk BASIC s grafikou. Počítač pracuje na síti i na baterie 4 × 1,5 V.

Mini floppy z BLR

BLR vyrábí kromě floppové diskové jednotky ES5074 (3,2 MB, 9 kp) i malou jednotku ES5088 pro pružné magnetické disky 133 × 133 mm s kapacitou 109,4 kB. Jednotka pracuje s 300 otáčkami za minutu, napájení + 12 V/0,9 A a 15 V/0,8 A, hmotnost 1,5 kp. Ideální pro malé osobní mikropočítače.

rh



Mini floppy ES5088