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 jazyk má však řadu nevýhod, z nichž nejvýznamnější je asi jeho pomalost. O nevýhodách tohoto jazyka ostatně podrobněji 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 kursu bychom vás chtěli s jazykem FORTH nejen podrobně seznámit, ale částečně vás i přesvědčit, že možná právě tento jazyk vám pomůže realizovat všechny vaše nápady.

1. ZÁSOBNÍK

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

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 můžete 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 následujícími zkratkami:

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

NOS — následující za TOS (Next On Stack) — mohli bychom říci položka 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. Uvědomuji si, že tento způsob zápisu si protiřeči s termíny „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, přestože si, jak jsem již poznamenal, s terminologií protiřeči.

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

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

2. FILOSOFIE JAZYKA FORTH

První, co začátečníka na jazyku FORTH většinou zaráží, 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 postavená 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 **zásobník 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 tvořen posloupností tzv. **slov,** oddělených mezerami nebo znaky „Nová řádka". Slovo může být označeno (pojmenováno) jakýmkoliv znakem kódu 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 vymaže a uloží tam případné výsledky.

Definice všech slov, neboli popisy ke slovům přiřazených činností, 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.

Interpretr 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, ohlásí chybu.

Před započetím programování v jazyku FORTH je zapotřebí provést důkladnou analýzu úlohy. Řešený problém si rozložíme na hlavní části, z nichž každou pak znovu 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 nadefinujeme slova, realizující nejjednodušší činnosti. Pomocí těchto slov a slov dříve známých nadefinujeme realizaci činnosti složitějších atd., až na konec nadefinujeme jedno slovo, řešící celý problém. Tento postup se nazývá „programování zdola nahoru" (bottom-up programming).

Specifickou vlastností jazyka FORTH je, že tato postupně tvořená slova nejen definujeme, ale také zároveň odlaďujeme. 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 realizovat „ručně".

Mnohé jiné jazyky, např. BASIC, nás nenutí provádět bezpodmínečně tuto teoretickou přípravu a svojí strukturou 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 nekonečným odlaďováním věčně chybných programů.

3. METODICKÉ POZNÁMKY

Základní interpretr 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)

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 v dvojnásobné přesnosti. Zaujímá dvě položky na zásobníku (tj. 4 bajty). Jeho rozsah je od -2 147 483 648 do 2 147 483 647.
F — Pravdivostní hodnota (flag). Nula je chápána jako **„FALSE"** (nepravda), nenulová hodnota jako **„TRUE"** (pravda). Máme-li někam uložit pravdivostní 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 stejné 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 myslet 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) uvádím pro snazší porozumění a větší názornost stav zásobníku po vykonání každého slova pod tímto slovem, a odděluji jej plnou čarou. U dvojtečkových definic (budou vysvětleny) uvádím stav zásobníku, který je očekáván před jejich plněním, pod jménem nově definovaného slova.

Pokud bude využíván i ZNA, budou zobrazení 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 doporučuji zhotovení tzv. RUP, což je zkratka názvu „ruční univerzální počítač". Pro tento účel si opatřete sadu dominových kamenů nebo jiných, jim podobných kostek. V nouzi postačí i z kartonu nastříhané obdélníky. Zásobník pak bude představovat řada pod sebou vyrovnaných kamenů (kostek, obdélníků), na nichž budou napsány hodnoty patřičných položek. Uložení položky na vrchol zásobníku budeme realizovat tak, že její hodnotu nebo symbolické označení napíšeme 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 prvé 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.

FORTH

4. ARITMETICKÉ OPERACE

Nová slova:

+ — (N1 N2 → (N1 + N2))
Sečte (TOS) a (NOS) a výsledek uloží na TOS.

— — (N1 N2 → (N1-N2))
Odečte (TOS) od (NOS), výsledek uloží na TOS.

***** — (N1 N2 → (N1*N2))
Vynásobí (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 svoji 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), sečte 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), odečte (TOS) od (NOS) a výsledek (=3) uloží na TOS.

Následuje slovo „ ***** " které vezme (TOS) (=3) a (NOS) (=5), vynásobí je a výsledek (=15) uloží na TOS.

Nyní již zbývá pouze tento součin umocnit. Jelikož mocnění na druhou není mezi základními slovy jazyka, musíme je obejít. Jednou z možností je zduplikovat TOS pomocí slova DUP a takto 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 odděluje 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 odladění programu nemuselo být na závadu, ale jeho hlavním handicapem je neustálé 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 přesvědčivé 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 využití 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 čárce jsou mnohem rychlejší. V případě potřeby však lze nadefinovat i slova, která pracují s jakýmikoliv jinými typy dat (vícenásobná přesnost, plovoucí čárka, komplexní čísla, . . .), která jsou v dané chvíli potřeba. O všech těchto možnostech se postupně dozvíte v průběhu kurzu.

Aby se vám trochu zažila práce se zásobníkem, zkuste si naprogramovat výpočet a vytištění 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	8	—	DUP	DUP	*	*	2	4	-	/	.
13	13	5	5	5	5	125	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 2 - 2 3 * 6 + * 9 3 – DUP * / .
2. 3 6 + 3 6 7 * – * .
3. 2 3 6 4 7 * – DUP * * * + .
4. 3 1 – DUP * DUP * .

FORTH

Ing. R. Pecinovský, CSc.

5. DEFINICE NOVÝCH SLOV — I

Nová slova:

: xxx — (→)
Začátek dvojtečkové definice slova xxx. Nastavení režimu COMPILE.

; — (→)
Konec dvojtečkové 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 „:" (dvojtečka). První slovo, které následuje za dvojtečkou (popř. za jí následující mezerou), se chápe 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).

Řekněme, 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 „:" (dvojtečka). Ve slovníku jsme si přečetli, že označuje „začátek dvojtečkové 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 počítači pouze oznámit, 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 kompilační režim (režim COMPILE), ve kterém určí posloupnost slov, jež se budou provádět, až bude mít počítač provést slovo právě definované.

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

Slovo „;" (středník) přikompiluje do slovníku ekvivalent známého RETURN, neboli činnost, kterou je třeba vykonat při návratu z procedury (slova), a ukončí kompilaci = nastaví zpět režim 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 zobrazené 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 preferují rychlost 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
:  4NA  DUP  3NA  *  ;      :  4NA  DUP  *  DUP  *  ;
-------------------------   -------------------------
   N    N    N    N**4        N    N    N**2 N**2 N**4
        N    N**3                 N              N**2
:  5NA  DUP  4NA  *  ;      :  5NA  DUP  DUP  *  DUP  *  *  ;
-------------------------   -------------------------------
   N    N    N    N**5        N    N    N    N    N    N    N**5
        N    N**4                 N    N    N**2 N**2 N**4
                                  N              N**2
```

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

1+ 1– 2+ 2– 2* ,

která přičtou (odečtou) k TOS jedničku (dvojku), resp. vynásobí 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  5  6
```

6. DEFINICE NOVÝCH SLOV — II

Nová slova:

SWAP — (N1 N2 → N2 N1)
Zamění (TOS) a (NOS).

NOT — (F → $\overline{F}$)
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)
Exclusivní 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 činností, 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. Slovník se zásadně prohledává ODZADU, tedy od naposledy nadefinovaných slov. Najde-li počítač hledané slovo a je-li v režimu EXECUTE, ihned toto slovo vykoná. Je-li v režimu COMPILE, přikompiluje 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) zařídí, 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, ohlásí 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 míněna 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 jasně vyplývá, proč se o něm někdy hovoří jako o třetím zásobníku. Pokuste 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)

FORTH

ing. R. Pecinovský, CSc.

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 vodítko 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 ČÍSELNÉ 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 níž 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 soustavě. Pokud se nám toto slovo nelíbí, např. pro svou délku, není problém tentýž úkon pojmenovat jinak, např.:

```
: DEC DECIMAL ;
```

Od této chvíle je systému lhostejné, 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 vynásobí 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 níž je uložena hodnota této proměnné. Pro ty hloubavější jenom dodám, že adresa slova **BASE**, označovaná .BASE., je něco jiného než adresa, na níž je uložena hodnota proměnné BASE. Podrobněji se o tom dozvíte ve 14. lekci.

Slovo „ ! " (vykřičník) 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 nenastaví 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 znovu 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 provedli slovo **BIN.** Proto jsme v druhé definici museli číslo 16 vyjádřit ve dvojkové soustavě.

Pomocí uvedených slov můžeme v jazyku FORTH jednoduše realizovat převody mezi různými číselnými soustavami. Nadefinujme 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 „(" (otevírací 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 vytištění tohoto čísla bude samozřejmě obnovena původní soustava.

Posledním úkolem této lekce bude malé zamyšlení. 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
  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 jakékoliv 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 používá k 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 zadaném výstupním zařízení název použitého procesoru.

FORTH

ing. R. Pecinovský, CSc.

8. ZÁSOBNÍK NÁVRATOVÝCH ADRES

Nová slova:

>R — (X →) ZNA: (→ X)
Uloží (TOS) UZ na TOS ZNA.

R> — (→ X) ZNA: (X →)
Uloží (TOS) ZNA na TOS UZ.

Slova v lekci nadefinovaná:

R@ — (→ X) ZNA: (X → X)
Nedestruktivní čtení ZNA.

EXIT — (→)
Předčasné ukončení slova — RETURN uprostřed definice.

OVER — (N1 N2 → N1 N2 N1)
Zkopíruje (NOS) na TOS.

DDUP — (N1 N2 → N1 N2 N1 N2)
Zduplikuje dvě vrchní položky na UZ.

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

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

3DUP — (N1 N2 N3 → N1 N2 N3 N1 N2 N3)
Zduplikuje tři vrchní položky na UZ.

3PICK — (N1 N2 N3 → N1 N2 N3 N1)
Zkopíruje (NNOS) na TOS.

Již v úvodu jsem hovořil o tom, že FORTH používá 2 zásobníky, přičemž prvý je systémový a druhý uživatelský. Doposud 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 prováděném. 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 „neodborným" užitím bychom mohli jejich činnost ohrozit. Ještě jednou tedy opakuji: PŘI POUŽITÍ ZNA JE TŘEBA NEJVĚTŠÍ OPATRNOSTI!

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 prvku zpět, jsou **>R** a **R>**. Pokusme se pomocí nich nadefinovat slovo **OVER:**

```
: OVER  >R    DUP   R>    SWAP  ;
  N1 N1 | N2 N1 | N2 N1 |  N1
  N2    |    N1 |    N1 |  N2
        |       |    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, aniž by ji v ZNA zrušilo.

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

```
: R@    R>    DUP   >R   ;
  X  RA | X | RA | X | RA | X
  RA    |   | RA |   |    | RA
```

```
: R   R>  R>  DUP  >R  SWAP  >R  ;
  X RA| X RA|  RA|  RA| X | X | X | X
  RA  |   X |  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 ⟶)
Testuje (TOS) na jeho pravdivostní hodnotu. V případě „TRUE" se posloupnost slov mezi IF a ENDIF vykoná, v případě „FALSE" se přeskočí a pokračuje se rovnou prvním slovem za ENDIF.

ENDIF — (⟶)
Označuje konec konstrukce IF . . . ENDIF.

NEGATE — (N ⟶ -N)
Vynásobí (TOS) číslem -1.

DROP — (X ⟶)
Smaže (TOS)

QUIT — (⟶)
Havarijní ukončení výpočtu — obdoba příkazu STOP. Další příkazy čte z klávesnice.

Slova v lekci nadefinovaná:

ABC — (N ⟶ |N|)
Uloží na TOS absolutní hodnotu (TOS)

?DUP — (X ⟶ X ?X?)
Zduplikuje (TOS) pouze je-li různý od nuly.

MAX — (N1 N2 ⟶ max (N1, N2))
Uloží na TOS hodnotu většího z obou argumentů.

MIN — (N1 N2 ⟶ 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   max(N1,N2)  max(N1,N2)
  N2   N2    N2      N1   min(N1,N2)
       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 <= (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   (HM≤N) (HM≤N)  ( HM≤N
  HM    HM  (HM≤N)   N    (N<0)   nebo
        N                         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).

Vaším 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 PÍŠEME 2KRÁT ČÍSLO, MUSÍME MÍT ZARUČENOU SPRÁVNOU ČÍSELNOU SOUSTAVU)

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

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

Nová slova:

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

IF- — (F →)
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 pokračuje prvním slovem za slovem **ENDIF**

ELSE—(⟶)
Ukončuje první posloupnost. Pokračuje se prvním slovem za **ENDIF**

ENDIF-(⟶)
Ukončuje konstrukci.

*/- (N1 N2 N3 ⟶ (N1*N2)/N3)
Uloží na TOS hodnotu výrazu (NNOS)*(NOS)/(TOS), přičemž mezivýsledek je počítán ve dvojnásobné přesnosti.

Slova nadefinovaná v této lekci:

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 „doslovná" 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 váš úkol.

Uvedený příklad budiž 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:

SIN

x < 35	x = 35
sin = x/6Ø	x = 9Ø-x sin = 1-x*x/7ØØØ

COS

x = 55	x > 55
cos = 1-x*x/7ØØØ	x = 9Ø-x cos = x/6Ø

TAN

x < = 39	x < = 51	x > 51
tg = 5Ø/x	tg = (2*x-33)/57	tg = 5Ø/(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 zajisté pamatujete, FORTH umí v základní verzi pracovat pouze s celými čísly.

11. CYKLUS S PARAMETREM I

Nová slova v této lekci:

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.

I — (⟶ N)
Uloží na TOS hodnotu parametru nejvnitřnějšího cyklu.

Slova v lekci nadefinovaná:

PREVTAB SACHOVNICE RADEK RADSACH NASOBILKA NA RADOBS OBSACH

Další základní konstrukcí, která se vyskytuje prakticky ve všech programovacích jazycích, je cyklus s parametrem. Tělo cyklu 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 meze 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 vytištění tabulky převodu mezi jednotlivými číselnými soustavami. Při definici slova, tisknoucího tuto tabulku, využijeme slova **PREV,** které jsme nadefinovali v sedmé lekci.

```
: PREVTAB DO I PREV LOOP CR ;
```

Takto nadefinováno, očekává **PREVTAB** v TOS prvé 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 políčko této šachovnice budou tvořit 2×3 znaky. Bílá políčka budou vytvořena mezerami, tmavá políčka znaky X. Levé horní políčko bude bílé.

```
: ŠACHOVNICE
  Ø                ( PŘÍZNAK BĚLOSTI POLÍČKA)
  8 Ø DO           ( CYKLUS TISKNOUCÍ ŘÁDKY ŠACHOVNICE)
    Ø 2 DO CR      ( ŘÁDEK ŠACH. TJ. DVA ŘÁDKY TISKU)
      8 Ø DO       ( CYKLUS TISKNOUCÍ ŘÁDEK ZNAKŮ)
        DUP IF     ( TEST PŘÍZNAKU BARVY POLE)
          ." XXX " ELSE
                   ( PŘÍZNAK <> Ø → ČERNÉ POLE)
          ."     " ENDIF
                   ( PŘÍZNAK = Ø → BÍLÉ POLE)
        NOT LOOP   ( ZMĚNA BARVY PRO DALŠÍ POLE)
      LOOP         ( KONEC TISKU ŘÁDKY ŠACHOVNICE)
      NOT          ( ZMĚNA BARVY POČ. DALŠÍ ŘÁDKY)
    LOOP           ( KONEC TISKU ŠACHOVNICE)
  DROP             ( VYMAZÁNÍ PŘÍZNAKU BARVY)
;
```

Uvedený příklad byl nejsložitějším slovem, které jsme doposud nadefinovali. Vysvětlíme si na něm několik zásad, kterých bychom 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)
  8 Ø DO         ( CYKLUS TISKNOUCÍ ŘÁDEK ZNAKŮ)
    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. ŘÁDEK ŠACHOVNICE = 2 Ř. ZNAKŮ)
  CR RADEK
                 ( TISK ŘÁDKU ZNAKŮ)
  LOOP           ( DALŠÍ ŘÁDEK ZNAKŮ)
;

: SACHOVNICE Ø
                 ( PŘÍZNAK BĚLOSTI POLE)
  8 Ø DO         ( CYKLUS TISKNOUCÍ ŘÁDKY ŠACHOVNICE)
  RADSACH        ( TISK ŘÁDKU ŠACHOVNICE)
  NOT            ( ZMĚNA BARVY POČ. DALŠÍ ŘÁDKY)
  LOOP           ( KNEC TISKU ŠACHOVNICE)
  DROP           ( VYMAZANÍ PŘÍZNAKU BARVY)
;
```

Kontrolní řešení k 10. lekci:

```
: SIGN   DUP  0<  IF  DUP  ABS  /  ENDIF  ;
: SIGN   DUP  0<= SWAP 0>=  NEGATE  +  ;
: SIN •  DUP  35  <  IF                  ( TEST: X < 35 ? )
         1Ø   *  6  /  ELSE              ( ANO: 1Ø * SIN = 1Ø * X/6 )
         9Ø   SWAP  —                    ( NE: X = 9Ø — X )
         2NA  7Ø  /                      ( TOS = x*x / 7Ø )
         1ØØ  SWAP  —  ENDIF             ( 1ØØ * SIN = 1ØØ — x*x / 7Ø)
: COS    DUP  55  =  IF
         2NA  7Ø  /  1ØØ  SWAP  —  ELSE
         9Ø   SWAP  —  10  *  6  /  ENDIF
: TAN    DUP  39  <=  IF
         5Ø   /  ELSE
         DUP  51  <=  IF
         2*   33  —  1ØØ  *  57  /  ELSE
         9Ø   SWAP  —  5ØØØ  SWAP  /  ENDIF  ENDIF
```

FORTH

Ing. R. Pecinovský, CSc.

Třetí zásadou je zavést si nějakou grafickou úpravu, vyjadřující vnořenost 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 násobilku 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   ( PŘECHOD NA NOVÝ ŘÁDEK)
  DUP  ( USCHOVÁNÍ NÁSOBENÉHO ČÍSLA)
  I DDUP . ." * " . ." = "
  * .
  LOOP
  DROP ( SMAZÁNÍ NÁSOBENÉHO ČÍSLA)
;

: NA ( UMOCŇOVÁNÍ — NOS = ZÁKLAD,
     TOS  EXPONENT)
  1 SWAP 0 DO
  ( PŘÍPRAVA MEZIVÝSLEDKU)
  OVER * LOOP
  ( NOS = ZÁKLAD, TOS = MEZIVÝSLEDEK)
  SWAP DROP   ( TOS = VÝSLEDEK)
;

: NA ( UMOCŇOVÁNÍ S KONTROLAMI —
     NOS = ZÁKLAD.  TOS - EXPONENT).
  OVER 0= IF
  ( TEST NULOVOSTI ZÁKLADU)
  0<= IF  ( ZÁKLAD NULOVÝ → TEST EXPONENTU)
  ." ARITMETICKÁ CHYBA"
  ( EXP <= 0 → NEDEFINOVANÝ VÝSLEDEK)
  QUIT ELSE
  ( HAVARIJNÍ UKONČENÍ VÝPOČTU)
  0 ENDIF
  ( EXP > 0 → VÝSLEDEK = 0)
  DUP 0< IF
  ( TEST ZÁPORNOSTI EXPONENTU)
  DROP 0 ELSE
  ( EXP < 0 → |VÝSL| < 1 → VÝSL = 0)
  DUP 0= IF
  ( EXPONENT NULOVÝ?)
  DROP 1 ELSE
  ( ANO → VÝSLEDEK = 1)
  1 SWAP 0 DO
  ( NE → NORMÁLNÍ UMOCNĚNÍ)
  OVER * LOOP
  ( NOS = ZÁKLAD, TOS = MEZIVÝSLEDEK)
  ENDIF  ( UKONČENÍ KONSTRUKCÍ IF)
  ENDIF
  ENDIF
  SWAP DROP   ( TOS = VÝSLEDEK)
;
```

12. CYKLUS S PARAMETREM II

Nová slova:

DO ... +LOOP

+LOOP — (N →)
Zvětší parametr cyklu o (TOS) a testuje, zda je menší než ukončovací hodnota cyklu (N < 0), resp. (N ≥ 0) větší nebo roven ukončovací hodnotě cyklu. Pokud je test „TRUE", vykoná se další běh cyklu, pokud je „FALSE" cyklus se ukončí.

— (→ N)
Uloží na TOS hodnotu parametru druhého nejvnitřnějšího cyklu (V TÉ SAMÉ DEFINICI!)

MOD — (N1 N2 → ZB)
Uloží na TOS zbytek po dělení N1/N2 (Dělení modulo).

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 přírůstkem jiným, než jedničkovým. Takovýto 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 nalezne 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 negací 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 takovéhoto 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
  ( PŘEDPOKLÁDÁME, ŽE JE PRVOČÍSLEM)
  I 3 / 3 DO
  ( DĚLÍME ČÍSLY OD 3 DO I/3)
  J I MOD IF
  ( TEST DĚLITENOSTI)
  NOT LEAVE ENDIF
  ( JE DĚLITELNÉ → NENÍ PRVOČÍSLO)
  2 +LOOP
  ( DĚLME DALŠÍM LICHÝM ČÍSLEM)
  IF  ( TOS = PŘÍZNAK PRVOČÍSELNOSTI)
  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 rozebírali u definice slova R v 8. lekci, nemůžeme nadefinovat:

```
: I R@ ;
```

jelikož by takto nadefinované slovo vracelo v TOS UZ svoji návratovou adresu!

Z toho vyplývá, že chceme-li užít ZNA uvnitř cyklu, musíme zařídit, aby tato slova (**LOOP, +LOOP** ...) našla ZNA ve stavu, v jakém ho zanechalo předchozí slovo z uvedené množiny nebo slovo **DO.** Obdobně, 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 zvyklí.

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

Dále se pokuste navrhnout 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 0 DO DUP IF
." XXX" ELSE ."     " ENDIF
NOT LOOP DROP ;
```

Kontrolní řešení:

```
: ARPR DUP >R 0 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 ;
: RADOBS
  (ŘÁDEK OBECNÉ ŠACHOVNICE — NOS = PŘÍZNAK BARVY 1. SLOUPCE, TOS = POČET SLOUPCŮ)
```

	DDUP	2	0	DO
F	F	F	F	F
NS	NS	NS	NS	NS
	F	F	F	?F
	NS	NS	NS	?NS
		2	2	

RADEK	LOOP	;
F	F	
NS		
?F		

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

: OBSACH
  ( OBECNÁ ŠACHOVNICE — TOS = POČET ŘÁDKŮ, NOS = POČET SLOUPCŮ)
  0 SWAP
  ( NNOS = NS, NOS = PŘÍZNAK BARVY 1. SL., TOS = NR)
  0 DO
  ( CYKLUS TISKNOUCÍ JEDNOTLIVÉ ŘÁDKY ŠACHOVNICE)
  OVER NOT OVER RADOBS
  (ZMĚNA BARVY 1. POLE A NATIŠTĚNÍ JEDNOHO ŘÁDKU)
  NOT LOOP
  DROP DROP
  ( SMAZÁNÍ PŘÍZNAKU BARVY A POČTU SLOUPCŮ)
;
```

13. CYKLY S PODMÍNKOU

Nová slova:

BEGIN ... UNTIL

BEGIN — (→)
Slouží jako návěští. Označuje začátek cyklu.

UNTIL — (F →)
Testuje (TOS) na jeho pravdivostní hodnotu. V případě „FALSE" opakuje cyklus (= pokračuje znovu od **BEGIN**), v případě „TRUE" jej ukončí a pokračuje dál.

BEGIN ... WHILE ... REPEAT

BEGIN — (→)
Označuje začátek konstrukce.

WHILE — (F →)
Testuje (TOS) na jeho pravdi-

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 platností či neplatností 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 ČÍSEL)
  OVER — ( TOS = ROZDÍL OBOU ČÍSEL)
  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) ne 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 podmínku 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 ČÍSEL)
  OVER —
  ( TOS = ROZDÍL OBOU ČÍSEL)
  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 ČÍSEL)
  >R Ø BEGIN
  ( NOS = MENŠÍ ČÍSLO, TOS = ODHAD NSN)
  OVER + ( TOS = NOVÝ ODHAD)
  DUP R@ MOD Ø=
  ( JE DĚLITELNÝ DRUHÝM ČÍSLEM?)
  UNTIL
  ( DOKUD NENÍ OPAKUJ CYKLUS)
  SWAP DROP R> DROP
  ( SMAZÁNÍ OBOU ČÍSEL)
;
```

```
: NSN
  DDUP < IF SWAP ENDIF
  ( TOS = MENŠÍ Z OBOU ČÍSEL)
  >R DUP BEGIN
  ( NOS = MENŠÍ ČÍSLO, TOS = ODHAD NSN)
  DUP R@ MOD
  ( JE DĚLITELNÝ DRUHÝM ČÍSLEM)
  WHILE
  ( POKUD NE, SPOČÍTEJ DALŠÍ ODHAD)
  OVER + ( TOS = NOVÝ ODHAD)
  REPEAT
  SWAP DROP R > DROP
  (SMAZÁNÍ OBOU ARGUMENTŮ)
;
```

14. VNITŘNÍ STRUKTURA SLOVNÍKU

Nová slova:

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

? (A→)
Vytiskne hodnotu na adrese A.

Slova v lekci nadefinovaná: **A B**

Ve slovníku jsou uložena všechna slova, která náš 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 nim následuje jméno slova, kódované v ASCII. Dvoubajtová položka označená SA je spojovací adresa, což je adresa počátku hlavičky předchozího slova. Tato položka bývá často označována i 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 seznamem 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 myslet 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. Doposud 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**	jméno (v ASCII)
103	**D**	
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**	jméno
118	**O**	
119	**T**	
120	**101**	SA
121		
122	**':'**	AVCP
123		

Všimneme si ještě způsobu, kterým se použití 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 vyvoláme, 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. následovné:

```
1 VARIABLE A        2 VARIABLE B
A  @  B  @  +  A  !  A  ?
.(A).  1  1  1  3  3     .(A).
          .(B).  2     .(A).
```

Po vykonání této posloupnosti slov nám počítač 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, tolikrát potřebujeme dva bajty navíc. Proto jsou nejpoužívanější čísla (0, 1, −1, 2) nadefinována jako slova jazyka FORTH pomocí překladače **CONSTANT**.

Než začnete číst 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 zná pouze číslice 0 a 1 a proto počítač ohlásil chybu.

15. DEFINOVÁNÍ NOVÝCH PŘEKLADAČŮ

Nová slova:

, −(X →)
Čárka – uloží (TOS) do slovníku.

<BUILDS −(→)
Vytvoří hlavičku nově definovaného slova.

DOES> −(→)
Ukončí kompilaci nově definovaného slova a nastaví ukazatel na výkonnou část 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 přicházejí na řadu dvě „magická" slova **<BUILDS** a **DOES>**. Špičaté 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 Forthem. Doposud se lišil od ostatních programovacích jazyků pouze svým poněkud „divokým" zápisem. Nyní však odhalíme jeho schopnost kvalitativně rozšiřovat sám sebe.

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

FORTH

Ing. Rudolf Pecinovský, CSc.

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

1. Definice NP pomocí překladače „:":
: CONSTANT <BUILDS , 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**	jméno
507	**A**	
508	**N**	
509	**T**	
510	**485**	SA
511		
512	**':'**	AVCP
513		
514	**.<BUILDS.**	
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**	jméno
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*)

<BUILDS — 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 vyhražené 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 položkou je adresa kompilační části slova **DOES>**, která ukončí fázi kompilace, 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 složitě? 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.

Vraťme 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**.

Provede se tedy:

@ — vezme obsah na adrese, 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:

Pomocí dvojtečkové definice můžeme nadefinovat nový překladač. Tato definice se skládá ze tří částí:

1) *Předkompilační část* (od jména NP po slovo **<BUILDS**). 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 **<BUILDS** po slovo **DOES>** včetně). Slovo **<BUILDS** vytvoří hlavičku NS, přičemž prozatím nenastavuje AVCP. Za slovem **<BUILDS** následuje popis činnosti, která se má vykonat během defino-

(9)

Ing. Rudolf Pecinovský, CSc.

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

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 výkonnou 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 →)
(VYHRADÍ VE SLOVNÍKU N BAJTŮ)

Slova v lekci nadefinovaná:

VEKTOR MATICE ARPRV ZAPLN () (,) VEK TABULKA

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

```
: VEKTOR
        ( TOS = POČET POLOŽEK VE VEKTORU )
  <BUILDS     ( VYTVOŘENÍ HLAVIČKY )
  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 zajímá. 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 nekontrolují případná přetečení indexu. Pokud bychom chtěli přidat hlášení chyb při opuštění mezí, museli bychom vektory nadefinovat např. následovně (verze vhodná pro ladění programu):

```
: VEKTOR
  <BUILDS     ( VYTVOŘENÍ HLAVIČKY )
  DUP ,       ( ULOŽENÍ DÉLKY VEKTORU )
  0 DO 0 , LOOP
              ( NULOVÁ POČÁTEČNÍ 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.) nehlídají a vystřihání 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 / ;
  ( VYPOČTE 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, NNOS = SUM = 0 )
  DO OVER I
             ( NNOS = SUM, NOS = PFA,
             TOS = POŘADÍ POLOŽKY )
  2* +       ( POČÍTEJ ADRESU I-TÉ POLOŽKY )
  +          ( PŘIPOČTI JI 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* + ;
```

popř.

```
: () 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, abychom 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 DÉL-
              KA < 256 )
  ( TÍM SI UMOŽNÍME, ABYCHOM MOHLI POZDĚJI
  PRACOVAT S ŘETĚZCI – BUDOU VYSVĚTLENY
  V 17. LEKCI – JAKO S VEKTORY ZNAKŮ)
  * ALLOT     ( VYHRAZENÍ MÍSTA V PAMĚTI )
  DOES>
;
```

Způsob uložení vektoru ve slovníku:

DJ	jméno	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 C! LOOP
              ( VYNULOVÁNÍ VŠECH BAJTŮ
              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, nenulují 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
            PRVKŮ )
  DO I () + LOOP
  R> C@ /   ( VYDĚLENÍ SOUČTU POČTEM
            PRVKŮ )
;
```

Slovo bychom pak použili v posloupnosti

VEK ARPRV

Slovo **ARPRV** bude nyní pracovat stejně pro jakýkoli vektor čísel.

Obdobně jako kompilátor vektoru bychom mohli nadefinovat i kompilátor matice. V zájmu obecnosti nadefinujeme matici jako vektor vektoru, čímž si umožníme pracovat jak s celou maticí, tak s jejími jednotlivými prvky, ale také s jejími jednotlivými sloupci jako celky (= vektory).

FORTH

Ing. Rudolf Pecinovský, CSc.

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

```
: MATICE      ( NOS = POČET SLOUPCŮ, TOS =
                POČET ŘÁDKŮ)
  <BUILDS
  2* 2+ C,    ( SLOUPEC = (POČET ŘÁDKŮ *2 +
                + 2) BAJTŮ )
  DUP C,      ( POČET POLOŽEK = POČET
                SLOUPCŮ )
  DUP >R Ø
  DO 2 C, J C,
                ( ČÍSLO = 2 BAJTY, POČET POLO-
                ŽEK = POČET SLOUPCŮ )
  J 2* ALLOT
                ( VYHRAZENÍ MÍSTA PRO I-TÝ ŘÁ-
                DEK )
  LOOP R> DROP
                ( SMAZÁNÍ POČTU SLOUPCŮ ZE
                ZNA )
  DOES> ;
```

Definujeme-li tedy matici

10 10 MATICE TABULKA

vypočítáme aritmetický průměr posledního sloupce této tabulky prostřednictvím posloupnosti slov

10 TABULKA () ARPRV

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

```
: (,) () () ;
```

Adresu (i,j)-tého prvku matice **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.

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 kompilace. Blíže si o podobných slovech povíme v 18. lekci.

CFA – (PFA → CFA)
Uloží na TOS CFA slova, jehož PFA najde na TOS.

Slova v lekci nadefinovaná:

EMIT – (Z →)
Vytiskne na obrazovku (obecněji na zadané výstupní zařízení) znak, jehož ASCII kód nalezne na TOS.

SPACE – (→)
Vytiskne jednu mezeru.

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 nalezne na TOS. Používá se při konverzi čísel na řetězec znaků.

SIGN – (N D → D)
V případě, že N < 0, začlení do výstupního řetězce (převod 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 kterékoliv učebnici programování mikroprocesorů.

Doposud jsme si vysvětlovali, jak „rafinovaně" lze psát v jazyce FORTH programy. Každý program ale musí umět předat spočtené 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 vlastností 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 zadané 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 nadefinováno

```
: EMIT 'EMIT @ EXECUTE ;
```

Rutiny se přepínají velice jednoduše. Chceme-li např., aby slovo **EMIT** provádělo námi nadefinované slovo **xxx**, napíšeme

```
' xxx CFA 'EMIT !
```

a od této chvíle probíhají všechny tisky prostřednictvím tohoto námi nadefinovaného 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 nadefinovat dvě užitečná slova, a to slovo **SPACE**, které na obrazovku (obecněji na zadané výstupní zařízení) vytiskne jednu mezeru a slovo **SPACES**, které vytiskne (TOS) mezer, avšak pouze v případě, že (TOS) > = Ø. Dříve, než se podíváte na jedno z možných řešení, zkuste si tato slova nadefinovat sami.

```
: SPACE BL EMIT ;
: SPACES Ø MAX ?DUP IF Ø
  DO SPACE LOOP ENDIF ;
```

Slovo **EMIT** vytiskne pouze jeden znak. Chceme-li vytisknout více znaků, pomůžeme si cyklem. Nadefinujeme si slovo **TYPE**, které vytiskne (TOS) znaků z oblasti paměti začínající na adrese (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. lekci 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**, nadefinujeme 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 nadefinovat také sami, ale jsou v některých bodech poněkud komplikovanější a vyžaduji znalost některých systémových slov a proto je raději budeme považovat za daná.

Slova realizující výstup čísel se systém od systému poněkud liší. Budu proto vysvětlovat slova v podobě, jak jsou definována v systému FORTH 602, dodávaném 602. ZO Svazarmu 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 (mimochodem můžete si jej přijít zdarma nahrát na schůzky

FORTH

Klubu uživatelů osobních počítačů každé liché úterý od 17 hodin v Praze 6, Pod Juliskou 2). Pokud byste používali některou z ostatních verzí jazyka FORTH, mohou zde být tato slova definována odlišně (TN-FORTH) nebo nejsou definována vůbec (BD-FORTH, mini-FORTH). Zde se právě ukazuje nesmírná tvárnost jazyka FORTH, protože každý uživatel si může tato slova ve chvíli, kdy přebírá programy vytvořené pod jiným systémem, předefinovat nebo dokonce dodefinovat podle potřeby. Řekněte, který jiný jazyk vám toto umožní?

Základním znakem celého souboru je to, že pracuje pouze s čísly ve dvojnásobné přesnosti. Toto řešení je sice poněkud pomalejší, ale v programech, v nichž záleží na rychlosti, nesmí být nikdy tolik tisků, aby doba jejich provádění nějak podstatně ovlivňovala dobu výpočtu. Na druhou stranu nám toto řešení ušetří paměť pro obdobná slova, která bychom museli pro tisk čísel ve dvojnásobné přesnosti (nepoužívají se zase tak zřídka) dodefinovat.

Základním slovem celého převodu je slovo # (mříž), které vygeneruje nejnižší platnou číslici z čísla v TOS.NOS (připomínám, že významnější dva bajty, tedy bajty obsahující informaci o znaménku, jsou v (TOS). Pro ty zvídavější dodám, že toto slovo vydělí vstupní parametr základem číselné soustavy, zbytek po dělení převede na patřičný znak a přidá do výstupního řetězce (tento řetězce se generuje **odzadu!**) a celou část podílu uloží jako výstupní parametr do TOS.NOS.

Celý převod je nutno zarámovat mezi slova <# a #>, která provedou vše potřebné před započetím vlastního převodu a po jeho ukončení, takže na konci můžeme celý řetězec vytisknout slovem **TYPE**.

Prvním slovem, které bychom si měli nadefinovat, je slovo #**S**, které vyvolává slovo # tak dlouho, dokud je co převádět, neboli dokud je výsledný podíl různý od nuly.

```
: #S BEGIN # DDUP OR 0=
UNTIL ;
```

Nyní by pro nás již měla být hračka nadefinovat si třeba slovo **U.**, které bude umět správně vytisknout jakoukoliv adresu, tzn. že i čísla větší než 32 767 bude považovat za kladná:

```
: U. 0 ( PŘEVEDE VSTUPNÍ PARAMETR
       NA ČÍSLO VE DVOJNÁSOBNÉ
       PŘESNOSTI )
  <# #S #> TYPE SPACE ;
       ( A VYTISKNE JEJ )
```

Pokud bychom chtěli tisknout čísla se znaménkem, potřebovali bychom k tomu poněkud bohatší soubor slov. Prvním z potřebných slov by bylo slovo **HOLD**, které začlení do vystupujícího textu znak, jehož ASCII kód najde na TOS. Zároveň dekrementuje proměnnou **HLD**, která obsahuje adresu posledního znaku, začleňovaného do výstupního řetězce (znovu připomínám, že číslo se generuje odzadu). Možná definice je tedy:

```
: HOLD -1 HLD +! HLD @ C! ;
```

Pomocí slova **HOLD** můžeme nadefinovat i slovo **SIGN**, které v případě potřeby přidá před celý řetězec znaménko „–". Jedna z možností je např.:

```
HEX : SIGN LROT 0 IF 2D HOLD
ENDIF ;
       ( 2D JE ASCII KÓD ZNAKU „–" )
```

Nyní si již můžeme ukázat, jak lze nadefinovat slovo **.** (tečka). Nesmíme ovšem ztratit ze zřetele, že dříve, než otevřeme vlastní konverzi, musíme si zapamatovat znaménko a převést číslo na kladné, např.:

```
: . DUP ABS 0 <# #S SIGN #>
TYPE SPACE ;
```

Přejděme nyní od slov základních ke slovům „rafinovanějším". Věc, kterou potřebujeme velice často a kterou většina verzí jazyka BASIC neumí, je tisk čísel do tabulky tak, aby byly shodné řády pod sebou – jednotky pod jednotkami, desítky pod desítkami, atd. Nadefinujeme si proto slovo **.R**, které očekává v NOS.NNOS tištěné číslo a na TOS šířku kolonky v takovéto tabulce. Toto slovo vytiskne před vlastní číslo tolik mezer, aby dané číslo bylo vytištěno až u pravého kraje vymezené oblasti. Pro zjednodušení budeme počítat, že se nám číslo do vymezené oblasti vejde. Pokud by se vejít nemělo, nevytiskne se před něj žádná mezera a číslo nám bude vpravo z vymezené oblasti „vyčnívat".

```
: .R      ( NOS = TIŠTĚNÉ ČÍSLO )
          ( TOS = ŠÍŘKA VYMEZENÉ OBLASTI )
  >R      ( USCHOVÁNÍ ŠÍŘKY OBLASTI )
  DUP     ( USCHOVÁNÍ ZNAMÉNKA )
  ABS 0   ( PŘEVOD NA KLADNÉ ČÍSLO )
          ( VE DVOJNÁSOBNÉ PŘESNOSTI )
  <# #S SIGN #>
          ( VLASTNÍ KONVERZE )
  R OVER - SPACES
          ( TISK MEZER PŘED ČÍSLEM )
  TYPE    ( VLASTNÍ TISK ČÍSLA )
;
```

Nyní zkusíme nahlédnout do světa desetinných čísel. Jak si jistě sami odvodíte, počítání s celými čísly je vlastně totéž jako počítání s čísly v pevné desetinné čárce s jediným rozdílem, a to že násobení bychom museli předefinovat. Zavedli bychom si proměnnou

```
1 VARIABLE MUL
```

do níž bychom si ukládali čísla, jimiž je třeba vydělit celočíselný součin, abychom dostali žádaný výsledek. Tuto proměnnou bychom nastavovali slovem (očekáváme, že do systémové proměnné **FDL** jsme si uložili počet desetinných míst)

```
: MUL! 10 FDL @ NA MUL ! ;
```

a nebo obecněji s uvažováním rozdílných bazí

```
: MUL! BASE @ FDL @ NA MUL
! ;
```

Operaci násobení bychom pak nadefinovali

```
: */ MUL @ */ ;
```

Pokud bychom však chtěli výsledky svých výpočtů vytisknout, jistě by se nám nelíbilo, kdyby v tomto tisku chyběla desetinná čárka. Není však nic snazšího, než nadefinovat:

```
HEX
: DM.R    ( NOS = ČÍSLO K TISKU )
          ( TOS = ŠÍŘKA VYHRAZENÉ OBLASTI )
  >R      ( USCHOVÁNÍ ŠÍŘKY OBLASTI )
  DUP     ( USCHOVÁNÍ ZNAMÉNKA )
  S->D    ( PŘEVEDENÍ DO DVOJNÁSOBNÉ
          PŘESNOSTI )
          ( SPUŠTĚNÍ VLASTNÍ KONVERZE )
  FDL @ ?DUP IF 0 DO # LOOP
  ENDIF
          ( KONVERZE DESETINNÉ ČÁSTI )
  2C HOLD
          ( VLOŽENÍ DESETINNÉ ČÁRKY )
  #S SIGN #
          ( DOKONČENÍ KONVERZE )
  R OVER - SPACES
          ( UMÍSTĚNÍ VE VYMEZENÉ
          OBLASTI )
  TYPE    ( VLASTNÍ TISK ČÍSLA )
;
: DM. 0 DM.R SPACE ;
```

A na závěr „chuťovku", která by vám měla alespoň částečně odkrýt netušené možnosti jazyka FORTH i v oblasti formátovaného tisku čísel. Představte si, že bychom měli vytisknout přesný čas – např. 23:03:51. Bohužel, chceme-li čas s přesností na sekundy, nemůžeme použít zobrazení s jednoduchou přesností, které nám umožňuje uchovávat pouze čísla do 65 535. Jak sami jistě lehce spočítáte, sekund je ale v jednom dni 86 400. Budeme tedy uchovávat počet sekund uběhlých od půlnoci v čísle v dvojnásobné přesnosti a chceme je vytisknout. Má to ale jeden háček – sekundy a minuty se počítají v šedesátkové soustavě a tisknou v desítkové. Lehká pomoc. Nadefinujeme si šestkovou soustavu a celý výstup pak bude vypadat následovně:

```
HEX
: B6 6 BASE ! ;
          ( DEFINICE ŠESTKOVÉ SOUSTAVY )
: :00     ( TOS = ČÍSLO, Z NĚJŽ ODDĚLÍM
          POČET SEKUND, POPŘ. MINUT )
  DEC #   ( KONVERZE JEDNOTEK )
  B6 #    ( KONVERZE DESÍTEK )
  3A HOLD
          ( VLOŽENÍ DVOJTEČKY PŘED DVOJČÍSLÍ )
;
: CAS     ( TOS.NOS = POČET SEKUND
          OD PŮLNOCI )
  BASE @ >R
          ( ULOŽENÍ STARÉ BÁZE )
  <# :00 :00
          ( KONVERZE SEKUND A MINUT )
  DEC #S
          ( KONVERZE HODIN )
  # TYPE
          ( TISK CELÉHO ÚDAJE )
  R BASE !
          ( OBNOVENÍ STARÉ BÁZE )
;
```

Odhalování dalších možností již ponechám na vaší fantazii.

Sami si nyní zkuste nadefinovat slovo **S–> D**, které převede (TOS) na číslo ve dvojnásobné přesnosti, které umístí do TOS.NOS.

Kontrolní řešení:

```
: S-> D DUP 0 IF -1 ELSE 0
ENDIF ;
```

18. SLOVA TYPU IMMEDIATE

Nová slova:

IMMEDIATE – (—>)
Označí naposledy definované slovo jako slovo typu IMMEDIATE, tedy jako slovo, které se provede i v režimu COMPILE.

SP@ – (—> A)
Uloží na TOS adresu původního TOS.

CSP – (—> .(CSP).)
Proměnná, do níž slovo : ukládá adresu TOS před vykonáváním definice. Počet položek na UZ musí před definicí a po ní souhlasit – tím se provádí nepřímo kontrola uzavřenosti programových konstrukcí.

[COMPILE] – (—>)
Začleňuje následující slovo do definice. Používá se k začlenění slov typu IMMEDIATE do definic.

STATE – (—> .(STATE).)
Proměnná obsahující informaci o režimu, v němž se systém nachází. (STATE) = 0 označuje režim EXECUTE, (STATE) ≠ 0 (v systémech FORTH 602 a fig-FORTH je typickou hexadecimální hodnotou C0) označuje režim COMPILE.