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 adrese, 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.

Říkali 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 těmto pravidlům vymyká. Její nejdůležitější podmnožinou jsou tzv. slova typu **IMMEDIATE**. Tato slova se, na rozdíl od slov standardních, provedou i když se systém nachází v režimu COMPILE. Doposud 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 možností 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 důvodu, 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 klávesnici. 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)
Čeká 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 – (→ .(**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 popsané 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 adrese 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 číslic připočte k proměnné DPL. Ve fig-FORTH se toto slovo jmenuje (**NUMBER**).

DPL – (→ .(DPL).)
Proměnná obsahující během konverze textu na číslo počet číslic 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, nevyskytne-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 adrese A+1 (očekává, že na adrese A je uložena délka řetězce a tento bajt ignoruje) na číslo dvojnásobné přesnosti s přihlédnutím k aktuální bázi. Pokud při transformaci narazí na desetin-

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 – (→)
Čte 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 adrese 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 ZÁVĚR SE DOZVÍŠ PROCENTU-
    ÁLNÍ ODCHYLKU SVÝCH ODHADŮ."
    CR CR CR
  ."STISKNI TLAČÍTKO POPRVÉ"
  BEGIN ?TERMINAL UNTIL
  ." – PODRUHÉ"
  32 767 0 DO ?TERMINAL IF I
    LEAVE ENDIF LOOP
  ." – POTŘETÍ"
  32 767 0 DO ?TERMINAL IF I
    LEAVE
  ENDIF LOOP OVER 2* – 100
    LROT */
  CR CR CR ." TVŮJ ODHAD
    DVOJNÁSOBKU DOBY SE LIŠIL"
    ." ZHRUBA O" . ." %"
;
```

Slovo **KEY** na rozdíl od slova **?TERMINAL** přeruší provádění programu do doby, než bude stisknuté nějaké tlačítko. Pomocí 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.

FORTH

Páteří celého systému je slovo **QUIT**, které se spustí po zapnutí ihned po inicializaci systému. Toto slovo je naprogramováno jako nekonečná smyčka. Jeho zestručněná 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 ŘÁDKU DO VSTUPNÍHO
           BUFFERU KLÁVESNICE )
  INTERPRET ( INTERPRETUJE ŘÁDKU
              V BUFFERU)
  REŽIM EXECUTE? IF ." OK" ENDIF
           ( ŘÁDEK ÚSPĚŠNĚ INTERPRETOVÁN )
  AGAIN
;
```

Slovo **INTERPRET** je definováno přibližně takto:

```
: INTERPRET
    BEGIN BL WORD
    ( VYHLEDEJ DALŠÍ SLOVO VE VSTUPNÍM
    BUFFERU )
      Nalezni toto slovo ve slovníku
      Našel?
      IF Režim EXECUTE?
        IF Proveď je
        ELSE Je typu IMMEDIATE?
          IF Proveď je
          ELSE Zakompiluje 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 přímo z klávesnice.

```
: TEXTCON
  <BUILDS 0 WORD
    ( PŘESUNE 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 omezovačem je znak z ASCII kódem 0. Slovo **WORD** tedy přečte obsah vstupního bufferu až do konce a přesune tento text za slovník, tedy do těla 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 = = přiřazený text. Např. slovo

```
: MEZÍ? 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 dotyčného znaku. Jelikož si však většina z nás tabulku ASCII nepamatuje a ani ji často nemá při ruce (od toho je počítač!), 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 jakýkoli 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 ;
```

které přečte řádek z klávesnice a provede jej, tj. „pošle na něj" vnější interpret. Pomocí 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 odladěn, 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 – . Takto 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    ( PŘÍPRAVA PARAMETRU
                PRO SLOVO BINARY )
   DUP 1+ C@ 2D =
                ( JE PRVNÍM ZNAKEM
                MÍNUS?)
   DUP >R     ( PŘÍZNAK ZNAMÉNKA >R )
   -            (POKUD BYLO ČÍSLO
                ZÁPORNÉ, POSUŇ ADRESU
                O+1 )
   ( POZOR! FIG-FORTH DÁVA PŘI TRUE +1,
   TAKŽE MUSÍME MÍNUS 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    ( PŘEVEĎ DALŠÍ ČÁST
                ČÍSLA )
     DUP C@ BL -
                ( NENUMERICKÝ ZNAK =
                MEZERA? )
   WHILE DUP C@
                ( NENÍ - PODEZŘELÝ ZNAK
                JE NA TOS )
     3A -       ( JE TO DVOJTEČKA? )
     IF DUP C@ 2C <
                ( NE  JE V ASCII PŘED
                ČÁRKOU
        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 vylepšené (např. potřebují číst čísla v plovoucí čárce). Slovo **INTERPRET** totiž používá pro převod textového řetězce na číslo to slovo, jehož CFA je hodnotou proměnné **NUMBER**. Předefinování je jednoduché:

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

Obdobně lze předefinovat 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 lekci 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 lekci hovoříme o tzv. logických slovnících, což jsou datové struktury typu strom. Každé slovo patří do některého logického slovníku a SA v jeho hlavičce odkazuje 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** (tento slovník budeme označovat (CONTEXT)) a v případě, že se nenalezne, 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 vnější interpret prohledává menší množinu slov a zpracování vstupního řádku je proto rychlejší, za druhé v různých logických slovnících mohou být slova stejně pojmenovaná, aniž 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 mít 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.

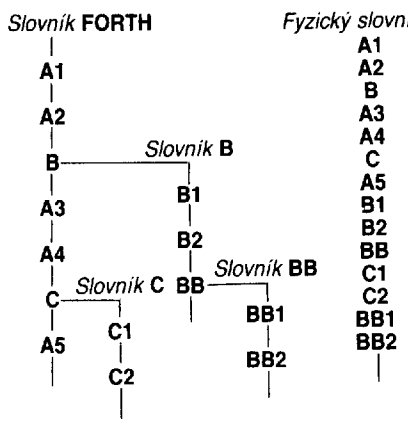
A nyní opět pozor! Také slovo **:** (dvojtečka) nastavuje proměnnou **CONTEXT** a to na slovník (CURRENT), neboli dvojtečka zařídí, že použitá slova se hledají v tomtéž slovníku, do nějž se ukládá právě definované slovo.

Nyní již tedy víme, jak lze nastavit slovník, v němž začneme 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, přestože byla definována dříve než slovo **C1**.

Dvojtečka v definici slova **C2** přepne **CONTEXT** zpět na slovník **C** a proto nemůže **INTERPRET** slovo **A5** najít, takže nepracujeme-li právě v číselné soustavě se základem větším než deset (pak by bylo možné A5 interpretovat jako číslo), ohlásí 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 počínat, když jsme v definici slova **BB2** chtěli 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

»
(1

FORTH

Ing. Rudolf Pecinovský, CSc.

i všechna slova ze slovníku, v němž byl náš slovník 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 nadefinovaná 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 kompilace, protože počítač nemusí prohledávat celý fyzický slovník, ale pouze logický slovník související s daným projektem. Standardní profesionální verze jazyka FORTH mívají 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 – (⟶)
Přečte 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 interpretru.

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 překládače může být psána přímo ve strojovém kódu, popř. v JSA.

(;CODE) – (⟶)
Výkonná část slova **;CODE**, která je začleňována do definice překladače, aby při jeho provádění ukončila fázi kompilace nového slova a nastavila CFA tohoto slova na výkonnou část použitého překládače. (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**. Podrobněji viz text.

ERROR – (⟶)
Součást slovníku **ASSEMBLER**. Očekává v registru HL kód chyby. Vyvolá proceduru ERROR jazyka FORTH.

SUBROUTINE – (B ⟶) komp.
(N1 ....N_p ⟶) výk.
Očekává v TOS počet parametrů definované procedury. Výkonná část pak zařídí, aby se při naprogramování vyvolání této procedury automaticky uložily její parametry za příkaz CALL.

PROCEDURE – (⟶)
Překládač – NS nadefinuje jako proceduru volatelnou z JSA pomocí CALL, .

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

Další slova:

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

Přestože 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 assemblerů ani makroassemblerů. Některé z jeho možností si ukážeme v této kapitole.

Programování v JSA vyžaduje již některé vědomosti o vnitřním interpretru 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-li 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 slova „spustí", aby po jejich vykonání pokračoval v plnění slova, která tato 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 explicitně 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 zvyklí – 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 skoku na počátek vnitřního interpretru, 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 implementací 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.

FORTH

Ing. Rudolf Pecinovský, CSc.

Ukážeme si nyní některé obraty, které lze při programování v JSA použít. Při programování v JSA totiž vůbec nemusíme zůstávat 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 ošetřuje případy, kdy nastala chyba. Ve všech těchto programech bychom tedy rádi v některém místě provedli skok na tuto část. Abychom mohli do programu začlenit skok, musíme nejprve umístit na TOS adresu, na níž 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 nově 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 paměti ve fyzickém slovníku zpět na počátek těla slova. Při příštím 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 níž chceme skákat. Na této adrese proto musí začínat program ve strojovém kódu. Proto jsme posunuli ukazatel volné paměti **DP** zpět na počátek těla.

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

Nadefinujme 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
  H PUSH,     JEJ ULOŽÍ NA TOS )
  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 VYPLNĚNÍM
              SLOVA EXECUTE )
  CFA
  JMP
```

Opět si celou sekvenci rozebereme po řádcích:

1. Nové slovo budeme ukládat do slovníku **ASSEMBLER**.
2. Pomocí překladače **PROCEDURE** nadefinujeme hlavičku slova.
3. Začlení do definice instrukci, která při provádění slova uloží hodnotu, která je nyní na TOS, do registrového páru HL.
4. Začlení do definice instrukci, která při provádění slova **ERROR** uloží obsah páru HL na TOS.
6. 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.
8. 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.
10. Uloží na TOS PFA slova **EXECUTE**.
11. Uloží adresu počátku slova **EXECUTE** na TOS (slovo **EXECUTE** je napsáno ve strojovém kódu, proto je adresa jeho počátku uložena v jeho CFA.
12. 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 ukázku možností jazyka FORTH:

```
ASSEMBLER DEFINITIONS
: SUBROUTINE
<BUILDS 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 DODÁ
              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 dále, 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 pomocí překladače **SUBROUTINE** budeme používat jako jiná slova jazyka FORTH – potřebujeme-li je vykonat, napíšeme 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, mívame 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 „čistého" jazyka FORTH. Nadefinujme si např. slovo **CSUM** , které sečte všechny bajty ze zadaného pole a uchová v registru A spodních osm bitů tohoto součtu.

```
PROCEDURE: CSUM
              ( HL = ADRESA PRVNÍHO
              BAJTU, B = POČET BAJTŮ )
  M A MOV,
              ( PRVNÍ SČÍTANEC DO A )
  BEGIN, B DCR, NZ
              ( KONEC ? )
  WHILE,      ( DOKUD B = 0
    H INX, M ADD,
              PŘIČTI DALŠÍ SČÍTANEC K „A" )
  REPEAT,     ( POKRAČUJ ZPĚT NA
              „BEGIN", )
  RET,        ( NÁVRAT )
```

Takto nadefinované slovo můžeme nyní použít:

```
CODE: CSUM-GEN
              ( GENERÁTOR
              KONTROLNÍHO SOUČTU )
  E POP, H POP,
              ( PŘEVZETÍ PARAMETRU
              Z UZ )
  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,
              ( PŘEVZETÍ PARAMETRU
              Z UZ )
  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 překladači z JSA, který používáme v jazyku FORTH, říká „strukturovaný makroassembler". Strukturovaný proto, že v něm můžeme programovat strukturovaně, 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 lavinovitému rozšíření jazyka FORTH byl dán v roce 1981 vydáním srpnového čísla časopisu Byte, které bylo celé věnováno tomuto do té doby téměř neznámému jazyku. Od té doby prudce vzrůstal počet implementací tohoto progresivního jazyka na všech druzích a typech počítačů. Jednou z prvních verzí jazyka FORTH v naší republice byl **BD-FORTH**, naprogramovaný ing. Dědinou, CSc., v ÚTIA ČSAV pro mikroprocesory 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 verzí je **fig-FORTH**, který vyvinula organizace FORTH interested group. Její koncepci jazyka pře-

(17

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 nemalou měrou jistě přispěla i skutečnost, že se jeho tvůrci vzdali jakýchkoli licenčních nároků.

Jinou velice známou verzí 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 řadového uživatele je již zbytečně dokonalý. Jen pro ilustraci uvedu, že v zájmu co nejrychlejšího vyhledávání slov ve slovníku používá osmi oddělených slovníků (mohli bychom je nazvat pseudofyzickými), do nichž ukládá slova v závislosti na prvním písmenu názvu. Při hledání slova pak vnější interpret potřebuje v ideálním případě zkontrolovat 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 jim podobného systému SOS budou možná znát systém **KNITH-FORTH**. Tento systém je FORTH jenom podle názvu. Z původního jazyka zbyly jen některé vnější rysy. Nemá slova <BUILDS a DOES> a i jinak působí spíše amatérsky.

U nás je v současné době v profesionální sféře nejvíce rozšiřován systém **FORTH 602**, distribuovaný 602. ZO Svazarmu v Praze 6. Vychází koncepčně 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. Autoři systému FORTH 602 naopak vycházeli z toho, že většina uživatelů v naší republice diskové jednotky nemá a proto se jej snažili vybavit některými možnostmi, které nepřítomnost diskových jednotek do jisté míry nahradí. Zároveň se ale snažili neodlišit 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 SAPI1, 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 čárce, soubory grafických podprogramů, přídavné programové a datové struktury a řadu dalších užitečných doplňků. Mezi nejužitečnější 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 disky, 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 výsledného 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.

FORTH

Ing. Rudolf Pecinovský, CSc.

Majitelé osobních mikropočítačů Sinclair si mohou zdarma nahrát překladače FORTH (v několika verzích) na setkáních 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. Žádný program nepřetrvá dlouho ve své původní podobě. Velice záhy ucítíte potřebu udělat několik „nepodstatných" změn. Není nepodstatných změn! Zejména v případech, kdy se k programu vracíte po delší odmlce, velice snadno přehlédnete řadu důležitých detailů, jejichž pravý význam jste mezitím zapomněli, a systém vás za to vytrestá.

Nevytvářejte dlouhá slova. FORTH je velice kompaktní, takže často již druhý den ztratíte přehled o tom, jak jste problém vlastně naprogramovali. Mějte neustále na paměti, že jedno dlouhé slovo se vymýšlí, programuje, testuje a ladí mnohem déle, než několik krátkých. FORTH a moderní programování vůbec nejsou přáteli složitých programů. Moderní filozofie programování razí zásadu co nejjednodušších procedur, vykonávajících pouze jednu přesně definovanou činnost.

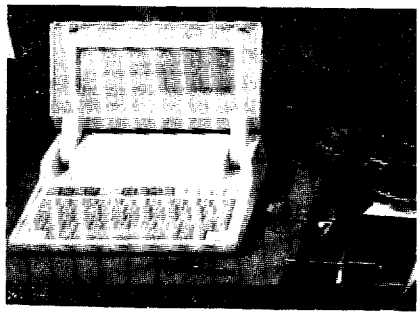
Jako každý jazyk se i FORTH nejlépe naučíte tím, že jej budete používat. Nepouštějte se však hned z počátku do konstrukcí a projektů, na něž nejste teoreticky a prakticky dobře připraveni. Neznalý programátor udělá řadu chyb, aniž by o nich věděl a uměl je najít. A většinou chybu nechápe jako chybu vlastní,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

Na závěr vám přeji, aby vás váš systém „poslouchal" a aby vám přinesl hodně zdaru ve vaší práci.

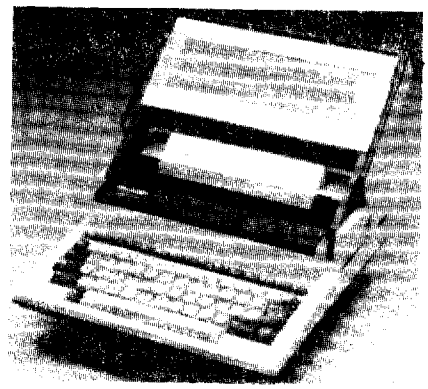
ZE SVĚTA

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 periférie 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 vydrží pracovat na baterie 16 hodin a údaje zůstávají v paměti počítače jeden rok po jeho vypnutí. **rh**



Mikropočítač HP 110



Mikropočítač SHARP PC-5000

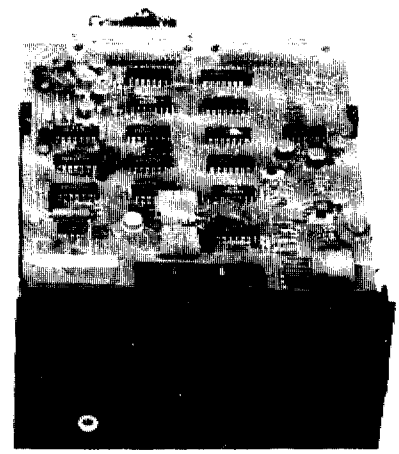
SHARP PC-5000

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 tiskárnu 80 znaků v řádce.

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

Mini floppy z BLR

BLR vyrábí kromě floppy 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 otáček 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